NEGOTIATING WITH QUALITATIVE PREFERENCES: METHODS FOR GENERATING BIDS EFFECTIVELY

by

Reyhan Aydoğan

B. S., in Computer Engineering, Dokuz Eylül University, 2003

M. S., in Computer Engineering, Boğaziçi University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2011

ACKNOWLEDGEMENTS

I thank my thesis supervisor Assoc. Prof. Pınar Yolum for her invaluable support during my thesis as well as her guidance, patience, and motivation. This thesis would not have been possible without her. I want to express my heartfelt thanks to Prof. Catholijn M. Jonker and Asst. Prof. Koen Hindriks. It is a great experience to work with them. I thank Prof. H. Levent Akın, Prof. Ethem Alpaydın, Prof. Taner Bilgiç, and Prof. Oğuz Dikenelli for being on my committee and giving useful feedback, which improved this thesis greatly.

I would like to express my gratitude to my family for their endless love, self-sacrifice and support as well as helping me to form my personality. I owe them everything. I want to dedicate this thesis to my parents Çetin Aydoğan and Emine Aydoğan. Special thanks to my dear mother Emine Aydoğan and my sister Özlem Aydoğan for motivating and encouraging me to complete my graduate education.

I want to thank everyone that I met through my graduate study. I am proud to meet them all. Particularly, I thank my friends Akın Günay, Assoc. Prof. Arda Yurdakul, Asst. Prof. Arzucan Özgür, Ayça Kundak, Barış Gökçe, Başak Aydemir, Dr. Canan Pembe, Dr. Çetin Meriçli, Demet Ayvaz, Deniz Gayde, Esma Kılıç, Gül Çalıklı, Hande Zırtıloğlu, Asst. Prof. Hatice Köse, Dr. Mehmet Gönen, Mirela Popa, Dr. Murat Şensoy, Özgür Kafalı, Recai Yalgın, Dr. Suzan Üsküdarlı, Tekin Meriçli, Tim Baarslag, Wietske Visser, Dr. Zhenke Yang, and members of AILAB for their support, motivation and the unique friendship that we share throughout these years.

Parts of this thesis have been published in different venues [1–8]. We thank the reviewers of these venues for their useful comments. This thesis has been supported by Boğaziçi University Research Fund under grant BAP5694. The Scientific and Technological Research Council of Turkey (TÜBİTAK) has partially supported me financially during my thesis. Finally, I would like to thank all my teachers, who have shared their knowledge and ideas with me over the years, and enlightened my path.

ABSTRACT

NEGOTIATING WITH QUALITATIVE PREFERENCES: METHODS FOR GENERATING BIDS EFFECTIVELY

This thesis studies automated bilateral service negotiation in which a consumer and a producer agent negotiate on a particular service in a distributed environment. The key challenges of automated service negotiation that are addressed here are the automatic generation of the service offers and the evaluation of the counter-offers. The negotiating agents need to represent and reason about their user's preferences in order to negotiate effectively on behalf of their users. Contrary to quantitative representations of preferences that are widely used in the literature, we advocate qualitative preference representations such as CP-nets. CP-nets enable users to represent their preferences in a compact and qualitative way but they almost always represent only a partial ordering. To cope with this limitation, this thesis develops a number of heuristics to be applied on CP-nets to estimate a total ordering of outcomes in terms of utilities from the partial ordering induced from a given CP-net. Consequently, the negotiating agent is able to employ existing utility-based negotiation strategies by means of estimated utilities. Our experimental results show that one can adopt effective heuristics on CP-nets to negotiate with a high performance in a reasonable time.

A negotiating agent also needs to understand its opponent's needs in order to generate accurate offers leading to successful negotiations. However, in many negotiation settings the participant's preferences are private. Accordingly, this thesis develops a novel preference prediction algorithm to understand the opponent's preferences from bid exchanges during the negotiation. This algorithm is enhanced with the use of an ontology so that similar service offers can be identified and treated similarly. Further, as the negotiation proceeds, the negotiating agent is able to revise its belief about the opponent's preferences. As a result, the agent generates well-targeted offers that are more likely to be acceptable by the opponent. This results in successful negotiations in which the participants reach a consensus faster and detect failures early.

ÖZET

NİTEL TERCİHLERLE PAZARLIK: ETKİN OLARAK ÖNERİLER OLUŞTURMAK İÇİN YÖNTEMLER

Bu tez, dağıtık ortamda üretici ve tüketici etmenlerin belli bir servis üzerine pazarlık yaptığı iki taraflı otomatik servis pazarlığını çalışmaktadır. Otomatik servis pazarlığının bu tezde çalışılan ana hususları otomatik bir şekilde önerilerin oluşturulması ve karşı önerilerin değerlendirilmesidir. Pazarlık yapan etmenler, bunların üstesinden gelmek ve kullanıcıları-nın yerine etkin şekilde pazarlık yapabilmek için, kullanıcılarının tercihlerini göstermeye ve bunların üzerine muhakeme yürütmeye gereksinim duyarlar. Literatürde yaygın bir şekilde kullanılan nicel tercih gösterimlerinin aksine, CP-net gibi nitel tercih gösterimlerini desteklemekteyiz. CP-net'ler tercihlerimizi derlitoplu ve nitel bir şekilde göstermemize imkan sağlamakla birlikte çoğu zaman sadece kısmi bir sıralama sağlarlar. Bu tez bu kısıtlamayla başetmek için, verilen bir CP-net'ten elde edilen kısmi sıralamadan faydalık açısından tam bir sıralama elde etmek amacıyla CP-net'ler için birtakım buluşsal yöntemler geliştirmektedir. Böylece, pazarlık yapan etmen faydaları tahmin ederek var olan fayda temelli pazarlık stratejilerini kullanabilir. Deneysel sonuçlarımız yüksek performansla makul, kısa sürede pazarlık etmek için CP-net'ler üzerinde etkin buluşsal yöntemleri kullanılabileceğini göstermektedir.

Pazarlık yapan etmen başarılı pazarlıklara yol açan doğru öneriler oluşturmak için rakibinin ihtiyaclarını da anlamaya gereksinim duyar. Fakat, çoğu pazarlık ortamında katılımcıların tercihleri gizlidir. Bundan dolayı, bu tez pazarlık sırasında öneri değiştokuşundan rakibin tercihlerini anlamak için yeni bir tercih tahmin algoritması geliştirmektedir. Bu algoritma ontoloji kullanımı ile geliştirilmiştir; böylece benzer servis önerileri teşhis edilebilmekte ve benzer bir şekilde muamele edilebilmektedir. Ayrıca, pazarlık ilerledikçe, pazarlık yapan etmen rakibinin tercihleri hakkındaki görüşünü yenileyebilmektedir. Sonuç olarak, etmen rakibinin kabul edilebileceği iyi hedef-lenmiş önerileriler oluşturmaktadır. Bu da katılımcıların daha hızlı fikir birliliğine ulaştığı ve olası başarısızlığı erkenden fark ettiği başarılı pazarlıklar ile sonuçlanmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$Dom(X_i)$	All possible values for the i^{th} attribute
G	The set of most general hypotheses
$ID3$	A basic decision tree algorithm
$P(C)$	The prior probability of an issue belonging to class C
$P(x)$	The evidence that is the probability of observing x
$P(C\|x)$	The posterior probability that is the probability of the given issue x belonging to class C
$P(x\|C)$	The likelihood of observing x when the given issue belongs to class C
S	The set of most specific hypotheses
X	A finite set of attributes
Ω	A set of all possible outcomes
Φ	Threshold for generalization of a single attribute value
θ	Threshold for generalization of the specific set

LIST OF ACRONYMS/ABBREVIATIONS

AAH	Average All Heuristic
C&D	Conjunctive and Disjunctive
CEA	Candidate Elimination Algorithm
CP-net	Conditional Preference Networks
CPT	Conditional Preference Table
DCEA	Disjunctive Candidate Elimination Algorithm
DH	Depth Heuristic
DLH	Depth Level Heuristic
GAI	Generalized Additive Independence
ML	Majority Lexicographical
PBayesian	Producer with Bayesian Classifier
PCEA	Producer with Candidate Elimination Algorithm
PDCEA	Producer with Disjunctive Candidate Elimination Algorithm
PID3	Producer with ID3
POH	Preferred Outcomes Heuristic
PRCEA	Producer with Revisable Candidate Elimination Algorithm
PRH	PageRank Heuristic
RCEA	Revisable Candidate Elimination Algorithm
TH	Taxonomic Heuristic
UCP-net	UCP-Networks
WCP-net	Weighted Conditional Preference Network

1. INTRODUCTION

Decision making constitutes an important part of our life. Most of the daily-life activities provide us a set of alternatives and we need to choose one of them. For instance, consider Alice who would like to go on a holiday in another city. There may be two possible transportation alternatives that she can choose from: she may travel by bus or by plane. She needs to choose one of them. The good thing is that the decision lies with Alice. That is, she can make a choice based on her preferences alone. If Alice prefers short trips, she might decide to take a plane. Conversely, if she prefers long trips, she might decide to take a bus.

However, the decision is not easily found when the decision is made by more than one participant. In such cases, the participants may (and probably would) have some conflicts in their interests. To illustrate this, consider that Bob is also going on holiday with Alice. Assume that Bob would like to go to abroad for their holiday while Alice prefers to go on holiday in their country. Moreover, Alice prefers long trips so she would like to take a bus. On the contrary, Bob likes short trips; hence he would prefer to take a plane. As seen, Alice and Bob have conflicts on their holiday destination and transportation. To be able to go on holiday together, they need to find a mutually acceptable agreement on their holiday. When the participants have conflicts in their interests, *negotiation* takes place [10, 11].

Negotiation is the process in which a number of participants interact to have a consensus — a mutually acceptable agreement — on a particular matter [10, 11]. Following our holiday scenario, Alice and Bob may agree on going on a holiday in their country and taking a plane for their trip, so that at least part of their requests are satisfied. The conflict of interest as outlined above is common in many settings. If Alice went ahead to buy the plane tickets, she might realize that the travel agent sells the tickets for a much higher price than what she would like to pay. This time, she needs to negotiate with the travel agent to find a solution that satisfies both them.

The simplest negotiation takes place between two parties on a single issue, such as price. These two parties interact to settle on a mutually acceptable value for that single issue. The more complex negotiations take place over multiple issues [11, 12]. For instance, the negotiating parties may have some conflicts about service quality metrics or even the service itself in addition to the price. Thus, they need to negotiate over the content of the service [13]. Without a doubt, a multi-issue negotiation is more complex than a single issue negotiation but it may take an advantage of the Win-Win solutions for the parties [14]. In a typical single-issue negotiation when one negotiating party wins, the other party loses. However, in a multi-issue negotiation the importance of the issues may vary for the parties. A negotiating party may consider a particular issue more important while its opponent may take another issue into account. Thus, it is possible to have trade-offs between issues, making consensus more plausible. For instance, the delivery time may be the main concern for a particular consumer whereas the price of the service may be more important for the producer. If the consumer pays more for fast delivery, both parties are at an advantage as far as their preferences are concerned. Therefore, there is no strict Win-Lose situation in a multi-issue negotiation unlike a single issue negotiation [14]. In this thesis, we deal with multi-issue negotiations between two negotiating parties, particularly a consumer and a producer.

Negotiating with another participant requires a number of interactions between the participants, especially when there are multiple issues to be agreed on. Think that negotiation is an arduous and time-consuming process, and some may believe that they cannot negotiate well and it would be better if someone else negotiates on behalf of them. For this purpose, we can use automated negotiation tools in which software negotiate on behalf of their users. For the above scenario, Alice's software would elicit Alice's preferences and accordingly negotiate on Alice's behalf. That is, it would generate offers and evaluate the counter-offers with respect to Alice's preferences. Finally, it would have a consensus on the travel package if possible. We expect such a software to represent its user, understand her preferences, reason and act on her behalf. These key behaviors that we expect from such a software match them ideally with software agents.

Software agents are autonomous software that are designed to achieve a particular objective while interacting with its environment as well as other agents [15–17]. An

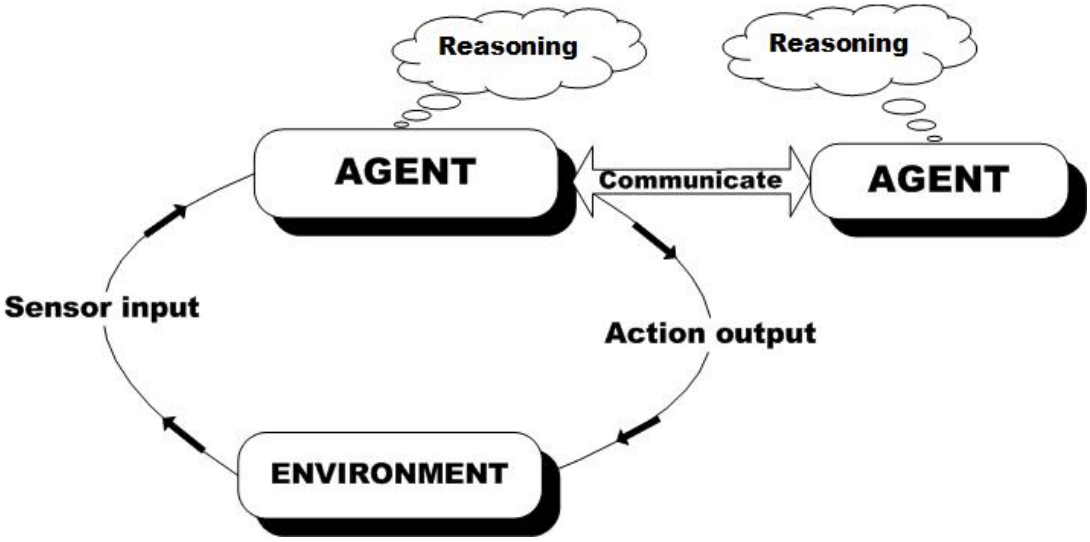


Figure 1.1. An agent interacting with its environment and other agent.

abstract view of an agent is drawn in Figure 1.1. As seen from the figure, agents perceive, reason, act and communicate with the environment as well as the other agents [17, 18]. An important property that is associated to agents is *autonomy*; that is, an agent is free to choose its actions and has control on its behavior at runtime [16, 17]. Another property is *heterogeneity* that enables us to design and construct agents in different ways [18]. In open and dynamic environment, the internal details of the agents are mostly different. That is, they can keep their user's information in a different format, can have different decision mechanisms and so on. All of the above statements support the fact that agents are successful paradigms for autonomous and intelligent software that act on behalf of their users such as those needed for flexible and automated applications.

We realize automated negotiation in this thesis by employing agents that perceive the negotiation environment, reason on their users' preferences, observe the other participant's offers, make decisions, propose their offers and communicate with the other participants. Such an automated negotiation can be adapted for many real-life problems. For example, assume that we are trying to arrange a meeting with our colleague. Both of our schedules may have several constraints and our preferences on the meeting time may conflict. Under these circumstances, finding an appropriate time may become difficult and time-consuming. It would be great if an agent negotiates with our colleague's agent to find a mutually acceptable time on our behalf.

In such a system, a negotiating agent can make an offer, accept the counter-offer made by the other agent or withdraw. As far as a negotiating agent's behavior is concerned, the main challenges are the automatic generation of its offers and the decision of which counter-offer is acceptable. In order to automatically generate well-targeted offers, a negotiating agent needs to model and reason about its user's preferences. Hence, representing and reasoning about preferences constitute an irreplaceable part of automated negotiation.

Before reasoning about the user's preferences, we need to elicit them from the user. In this phase, a number of questions related to the user's preferences are asked to the user. Preferences can be expressed quantitatively as well as qualitatively. While it is common to represent and reason about the user's preferences quantitatively, there are several issues to be taken into account. First, the space of all possible alternatives, *outcome space*, grows exponentially with the number of issues and their possible values. It may be infeasible to ask a user to order or rank all outcomes when the outcome space is large. Second, the user may have difficulty in assessing her preferences in a quantitative way [19]. Since representing someone's preferences with numerical values is not natural for a human, the users are unwilling to express their preferences quantitatively. Third, it is difficult to find a mathematical model for representing some preferences such as conditional preferences in which there are preferential dependencies between attributes. The above leads us to the fact that it is more effective and intuitive to use a qualitative preference model.

The immediate question is how to use qualitative preference models with negotiation. Most of the existing negotiation approaches [20–24] use quantitative preferences such as utilities. They assume that the agent has its user's preferences as a utility function mapping the outcomes to a real value, mostly scaled between zero and one. These approaches generate the offers automatically with respect to the utility of the outcomes. For instance, the agent may first generate the offer whose utility is the highest for its user and concede over time. To do this, it needs to compare the utility of the possible outcomes. When we have a concrete metric such as utilities to compare the outcomes, it is relatively easy to make an offer but it is not clear how offers can be generated with qualitative preferences. How does an agent reason about these preferences and generate

its offers automatically with respect to these qualitative preferences? Accordingly, this thesis considers two qualitative preference representations: conjunctive and disjunctive constraints and conditional preference networks (CP-nets) [19, 25] and presents how an agent negotiates with these preferences.

Furthermore, we study how an agent refines its offers to improve the negotiation and complete negotiation early. To do this, the negotiating agent needs to not only consider its own preferences but also take into account the other agent's preferences. Otherwise, no matter how good the generated offer is for the negotiating party, it will not be accepted by the other agent. To find a mutually beneficial outcome for both negotiating parties, the parties need to have sufficient knowledge about the negotiation domain and to take the other agent's preferences into account [23].

However, preferences of parties are almost always private and hence cannot be accessed by others. If a negotiating party shares its preferences with its opponent, this information can be exploited by the opponent [23]. For example, if the producer knows that the consumer can pay up to 100 USD for a service, it may not offer a price lower than 100 USD, although without that knowledge it can lower the price to 80 USD. As a result of revealing true preferences, this negotiation might end up with a lower gain for one of the agents. Thus, the negotiating parties may not prefer to reveal their preferences completely. Another problem may be that the preferences of the agents may be large. Because of the communication cost, these preferences may not be shared [26]. The best that can happen is that negotiating parties use additional domain knowledge or may *predict* each others' preferences through interactions over time.

An agent may improve the negotiation process by using additional domain knowledge in its strategy. If a consumer finds a four-star hotel expensive, one can infer that she would find a five-star hotel even more expensive, using the domain knowledge that the more stars a hotel has, the more expensive it would be. Or if a consumer prefers a vacation near a sea, when that can not be provided, it might make sense to offer a vacation in a hotel with a pool rather than in a hotel near the mountains. These inferences can be made possible if we have access to some domain knowledge.

Besides using domain knowledge for making inferences, it can be beneficial to use the domain knowledge in predicting the opponent agent's preferences and generating counter-offers accordingly. For this purpose, a preference prediction algorithm enhanced with the use of ontologies need to be developed. Principally, this algorithm might use the offers made by the negotiating parties during the negotiation in order to find a generic model reflecting the opponent agent's preferences. Incorporating the ontology reasoning to this prediction algorithm may improve the performance of the prediction about the opponent's preferences. In this way, the negotiation process may improve by using meta-information about the negotiation domain in predicting opponent's preferences.

As agents consider each others' preferences, they can provide better-targeted offers and thus enable faster negotiation. Predicting and understanding the opponent agent's preferences reduce the search space for the alternatives, which leads to more efficient negotiation. This may come up with an earlier consensus, which also reduces the communication cost. Hence, guessing the opponent agent's preferences through interactions over time and reasoning on these predicted preferences plays a key role for effective negotiations.

1.1. Research Challenges and Contributions

Contrary to quantitative preference representations that are widely used in the literature, we advocate qualitative representations such as CP-nets [19, 25] for preference representation and reasoning in negotiation. Although CP-nets provide us a natural and intuitive way of expressing the users' preferences including the conditional preferences, they can tolerate partial ordering. That is, there are some outcomes that we cannot compare. On the other hand, a negotiating agent should be able to compare each outcome and choose the most preferred one as an offer. One challenge is to develop mechanisms for an agent to negotiate with its user's partial preferences. This thesis pursues the ways of negotiating with partial preferences.

While using qualitative preferences such as CP-nets in negotiation, we also consider how we can reuse the existing negotiation approaches that are originally developed for quantitative preferences such as utilities without having the quantitative preference

information explicitly. Therefore, we propose several heuristic approaches that map the qualitative preferences represented via CP-nets to the quantitative preferences such as utilities. The estimated utilities can be used by any negotiation approach based on utilities unless the approach use the structure of the utility functions explicitly.

Here, an important point is to be able to evaluate the performance of these heuristics fairly. If we knew the user's real utilities assigned to the outcomes, we could evaluate the negotiation outcomes gained by the heuristic approaches with respect to the user's real utilities. On one hand, we can ask the user for her quantitative preferences and compare each negotiation outcome reached by the heuristics with respect to these quantitative preferences. On the other hand, one needs to ensure that the ordering of outcomes is consistent with the partial ordering induced from the CP-net. We need to find a convenient quantitative preference representation that is able to satisfy the CP-net relations and allows us to easily verify whether the total ordering provided by this representation is consistent with the ordering induced from a given CP-net.

Fortunately, UCP-nets [27] have the aforementioned properties. Thus, UCP-nets can serve as ground truth in our setting. To do this, we develop an environment in which the same user expresses her preferences both quantitatively (UCP-net) and qualitatively (CP-net). While the agent uses the estimated utilities by a particular heuristic from the given CP-net during the negotiation, the negotiation outcome is evaluated with respect to the utility in accordance with the given UCP-net consistent with the CP-net. Further, our simulation environment also allows us to compare the performance of agents when they apply heuristics on their users' qualitative preferences as CP-nets and negotiate with estimated utilities versus when they have their users' real total preference orderings as UCP-nets.

As mentioned before, it is important for a negotiating agent to understand the other agent's preferences from the offers proposed by the agents during the negotiation and generate well-targeted offers that are more likely to be acceptable by the other agent. The immediate question is how the negotiating agent can understand its opponent's preferences from the offers proposed during the negotiation. The agent may use the proposed offers as a training instances and apply an algorithm to model the opponent

agent's preferences.

But modeling other's preferences in negotiation is complicated for various reasons. First, the participant does not know how the other agent represents its preferences. For instance, the agent may use constraints or utility functions to represent its preferences. Alternatively, the agent may use an ordering of the alternatives for its preferences. Second, the agent does not know whether preferential interdependencies among issues exist. This uncertainty leads the agent to make some assumptions about its opponent's preferences or even its behavior in the negotiation [23, 24]. On the other hand, these assumptions may not work as expected in open and dynamic environments. Consider a producer agent that tries to learn a consumer's preferences and assumes that the issues are independent. This assumption may be consistent with some consumer's preferences but it will fail in others. Thus, it would be better to try to learn a generic model that is helpful for generating well-targeted offers without making assumptions about the agent's preferences. Third, the data becomes available as the negotiation progresses and many times. The number of training instances may be inadequate to predict the opponent's preferences accurately. Therefore, our aim is to acquire a generic model leading the negotiating agent to understand the other agent's need accurately enough to generate counter-offers effectively rather than to model the exact preferences.

Accordingly, this thesis develops a preference prediction algorithm based on concept learning in order to predict the opponent's preferences and generate well-targeted offers based on these predictions that enable the parties to negotiate effectively. This algorithm considers the offers and the counter-offers proposed by the negotiating parties in order to find a generic model reflecting the opponent's preferences. Since these offers become available during the negotiation, the developed preference prediction algorithm is incremental. As we mentioned before, the negotiating agent can reason on the domain knowledge using an ontology to improve the performance of the preference predictor. With an ontology, we can capture information about the relations between issues and use these relations when generating offers. Following the previous example, if a holiday in Turkey in the summer cannot be supplied, it would be better to find a similar holiday (e.g., a holiday in Greece in the summer) than a randomly-picked holiday. Hence, our prediction algorithm is enhanced with the use of ontologies so that similar service

requests can be identified and treated similarly.

1.2. Organization of the Dissertation

The rest of this paper is organized as follows. In Chapter 2, we present an overview of several preference representations that we specifically use in this thesis. We also discuss the pros and cons of each representation from a negotiation perspective. In Chapter 3, we provide a brief explanation of negotiation basics and present our service negotiation architecture in which a consumer and a producer agent negotiate in order to reach an agreement.

In Chapter 4, we study how the consumer agent negotiates with qualitative preferences such as conjunctive and disjunctive constraints and CP-nets. We present a number of heuristics for negotiating effectively with CP-nets. In Chapter 5, we evaluate the proposed heuristics for CP-nets in an experimental setting and compare the performance of CP-nets with heuristics with the performance of UCP-net in negotiation.

In Chapter 6, we present a preference prediction algorithm based on concept learning, namely Revisable Candidate Elimination Algorithm, in order to predict the other agent's preferences during the negotiation and generate well-targeted offers leading faster negotiations. In Chapter 7, we evaluate the proposed preference prediction algorithm for negotiation in an experimental setting and compare the performance of RCEA with other learning approaches such as CEA, DCEA, ID3 and Bayes' classifier in negotiation. In Chapter 8, we discuss our work with references to the literature and present an overview of this thesis with the directions for further research.

2. PREFERENCE REPRESENTATIONS

We mostly reason about our preferences in decision making problems where we need to choose an outcome from a set of alternatives (possible outcomes). A preference relation can be considered as a binary relation denoted by $\succeq$ on a set of all possible outcomes (Ω) such that for the outcomes $o_1, o_2 \in \Omega$, $o_1 \succeq o_2$ means that o_1 is equally or more preferred over o_2 [28]. If this relation is total, we are able to compare each outcome pairs. Otherwise, there are some outcome pairs that are incomparable (partial ordering). It is worth to note that $\succeq$ represents a weak ordering and we can show a strong (strict) ordering such that $o_1 \succ o_2$ if and only if $o_1 \succeq o_2 \wedge o_2 \not\succeq o_1$ where $o_1 \succ o_2$ means that o_1 is strictly preferred over o_2 [28, 29].

There are a variety of preference representations. Chevaleyre *et al.* categorize preference representations in terms of their structure as follows: [29]:

- Cardinal preference structure: In this category, we have an evaluation function mapping each outcome to either a numerical value or a totally ordered qualitative scale such as very good, good and so on.
- Ordinal preference structure: This category involves a binary relation $\succeq$ described before so that outcomes are not assigned to a numerical numbers but are related to some of the other outcomes with the relation $\succeq$. They also specify that this relation is reflexive and transitive.
- Binary preference structure: In this category, we divide the space of all possible outcomes into two partitions: good and bad states.
- Fuzzy preference structure: We have a fuzzy membership function indicating the degree to which an outcome (o_i) is preferred over another outcome (o_j).

In general we can categorize the preference representations as quantitative and qualitative preference representations. We can represent our preferences by simply ordering the outcomes qualitatively such as "A low price is preferred over a high price", `Low` $\succ$ `High`. In contrast to qualitative preferences, we quantify our preferences by assigning numerical values such as utilities to the outcomes in quantitative preference representa-

tions. For example, we can specify that the utility of `Low` is 1.0 where the utility of `High` is 0.25. This provides us a measure to compare the outcomes explicitly but from the user's point of view, it is an arduous task to define such numeric values. Furthermore, it may be difficult to represent preferences quantitatively while we have complex preferential dependencies among attributes. In contrast to quantitative preferences, representing preferences qualitatively is more natural and intuitive for the user.

In this section, we briefly examine four preference representations. First two are qualitative while others are quantitative.

2.1. Conjunctive and Disjunctive (C&D) Constraints as Preferences

It is possible to express someone's preferences via conjunctive and disjunctive constraints since preferences express "the relative or absolute satisfaction of an individual when faced with a choice between different alternatives" [29]. In fact, this preference representation describes a set of acceptable outcomes. That is, it divides the space of all possible outcomes into two partitions: acceptable and unacceptable outcomes. Thus, the preference structure fits the third category: binary preference structure.

We define that X is a finite set of attributes, $X = \{X_1, X_2, ..., X_n\}$ and every attribute X_i has a domain $Dom(X_i)$ of possible values. An outcome o_i is a complete assignment of all attributes such that $o_i =< v_1, v_2, ..., v_n >$ where each $v_i \in Dom(X_i)$. A conjunctive constraint c_i is a complete assignment of the attributes such that $c_i =< cv_1, cv_2, ..., cv_n >$ where each $cv_i \in \{Dom(X_i) \cup$ "?"$\}$ and "?" means that any domain value is acceptable for that attribute. A disjunctive constraint is the union of conjunctive constraints such that $dc_i = c_1 \cup c_2 \cup ... \cup c_m$ where $m > 0$. An outcome o_i is acceptable if and only if it satisfies at least one of the conjunctive constraints.

To illustrate this, consider a wine domain with these three attributes: `Region`, `Color` and `Sugar Level`. For simplicity, wine region can be `Bordeaux`, `Chianti` and `California`. There are three possible wine colors: `Red`, `Rose` and `White`. The domain for the sugar level involves `Dry`, `OffDry` and `Sweet`. A possible outcome can be any complete assignments of these three attributes. For instance, <`Chianti`, `White`, `Sweet`>

represents a sweet and white wine produced in Chianti.

If the user prefers any red and dry wine, the user's preferences can be represented via a conjunctive constraint such as $<?, \texttt{Red}, \texttt{Dry}>$. Note that a conjunctive constraint is the conjunction of each individual constraint on a single attribute. Here, $<?, \texttt{Red}, \texttt{Dry}>$ implies that ($\texttt{Region}=? \wedge \texttt{Color}=\texttt{Red} \wedge \texttt{Sugar}=\texttt{Dry}$) where $\wedge$ denotes "and" operator. If and only if all of the individual constraints are satisfied by the outcome, we say that the conjunctive constraint is satisfied and that the outcome is acceptable. Otherwise, that outcome is unacceptable.

Under some circumstances, a conjunctive constraint would be insufficient to represent some preferences requiring more than one acceptable outcome description. For example, the user may prefer any red and dry wine or any French wine. To represent such preferences, we use a disjunctive constraint such as $<?, \texttt{Red}, \texttt{Dry}> \cup <\texttt{French}, ?, ?>$ implying ($\texttt{Region}=? \wedge \texttt{Color}=\texttt{Red} \wedge \texttt{Sugar}=\texttt{Dry}$) $\vee$ ($\texttt{Region}=\texttt{French} \wedge \texttt{Color}=? \wedge \texttt{Sugar}=?$). It is the disjunction of two conjunctive constraints — the union of two acceptable service descriptions. If at least one of the conjunctive constraints is satisfied, the disjunctive constraint is satisfied. If none of the conjunctive constraints are satisfied by the outcome, that outcome is unacceptable.

Although it is easy to represent preferences as disjunctive and conjunctive constraints, we are not able to express that an outcome o_i is preferred over another outcome o_j. In some negotiation domains, it may be reasonable to use disjunctive and conjunctive constraints classifying acceptable and unacceptable outcomes. However, it may be insufficient in some domains where the user may prefer some outcomes over others. Consider that two outcomes are acceptable for the user but the user will be happier if she get the first outcome. Under these circumstances, we need a representation enabling to express such preferences. The next preference representation, CP-net is able to express such preferences as well as handle conditional preferences.

2.2. Conditional Preference Networks (CP-nets)

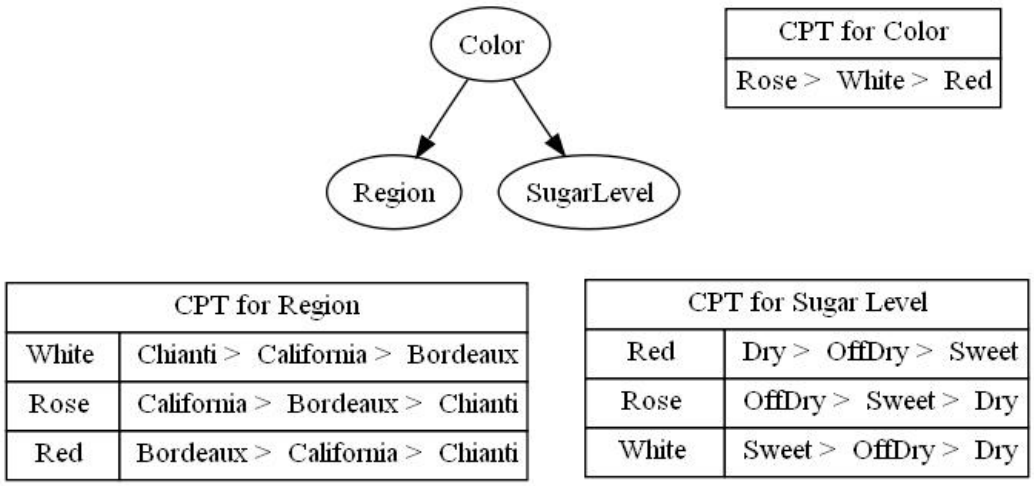
CP-nets are graphical models for representing qualitative preferences in a compact way [19, 25]. As far as combinatorial domains are concerned, the number of all possible outcomes may be so huge that it may be infeasible to ask a user to represent her preferences in an explicit way by ordering or ranking all possible outcomes. Note that the number of all possible outcomes is equal to $\prod_{i=1}^{n} |Dom(X_i)|$ where $Dom(X_i)$ denotes all possible values for the i^{th} attribute and an outcome consists of n attributes. Under these circumstances, it is important to express preferences in a compact way. Compact representation reduces the amount of information gathered from the user while it requires some assumptions such as preferential independence about the preferences.

We say that Y (a subset of X) is preferentially independent of Z (a subset of X) if the values of Z does not affect the preferences on Y where X denotes the set of all possible attributes, and Y and Z are disjoint subsets ($X = Y \cup Z$ and $Y \cap Z = \emptyset$). That is, if and only if for every $y_1, y_2 \in Dom(Y)$ and $z_1, z_2 \in Dom(Z)$, we have $y_1 z_1 \succeq y_2 z_1$ if and only if $y_1 z_2 \succeq y_2 z_2$ [19, 25, 28].

Conditional preferential independence can be defined analogously. We say that Y is conditionally preferentially independent of Z given an assignment w (condition) to W where Y, Z and W are disjoint sets of X. In other words, if and only if for every $y_1, y_2 \in Dom(Y)$ and $z_1, z_2 \in Dom(Z)$, we have $wy_1z_1 \succeq wy_2z_1$ if and only if $wy_1z_2 \succeq wy_2z_2$ [19, 25, 28].

CP-nets capture the qualitative conditional preferential independence and each preference statement is interpreted under the *ceteris paribus*, "all else being equal" assumption. That is, if we say that y_1 is preferred over y_2, this means that if all other attributes are the same, we prefer y_1 to y_2 [30].

A CP-net is a directed graph in which each node represents an attribute (variable) and each edge denotes preferential dependency between nodes under the "all else being equal" assumption. Here, if there is an edge from R to T, R is called the parent node and T is called the child node. The preference for child nodes depends on their parent



CPT for Color
Rose > White > Red

CPT for Region	
White	Chianti > California > Bordeaux
Rose	California > Bordeaux > Chianti
Red	Bordeaux > California > Chianti

CPT for Sugar Level	
Red	Dry > OffDry > Sweet
Rose	OffDry > Sweet > Dry
White	Sweet > OffDry > Dry

Figure 2.1. A sample CP-net for our simplified wine domain.

nodes' values. To express (conditional) preference orderings, each node is associated with a conditional preference table (CPT), which represents a total order on possible values of that node with respect to its parents' values.

Consider our wine domain explained before and the CP-net depicted in Figure 2.1. According to this CP-net, the user's preferences on wine region depend on wine color and wine color also affects the user's preferences on sugar level. CPT for `Region` shows that the user prefers `Bordeaux` to `California` to `Chianti` when wine is red. When it is rose, `California` is preferred over `Bordeaux` and `Bordeaux` is preferred over `Chianti`. While wine is white, the user prefers Chianti to California and California is preferred over Bordeaux. Note that in CP-nets, each preference statement is interpreted under the "everything else being equal" assumption. The statement, "`White` is preferred over `Red` for wine color", means that if all other attributes such as wine region and sugar level are the same, a white wine is preferred over a red wine.

In acyclic CP-nets, there is only one best (optimal) outcome so it is straightforward to determine the best outcome (answering the outcome optimization query). From ancestors to descendants, the most desired value for each attribute is chosen in order to get the best outcome. However, we need to check whether there exists an *improving flip* or *worsening flip* sequence from one outcome to another to answer dominance queries (whether an outcome would be preferred over another). An improving flip is changing the value of a single attribute with a more desired value by using the CPT for that attribute.

A worsening flip can be defined analogously.

To illustrate this, consider the CP-net in Figure 2.1. According to this CP-net, <`Chianti`, `Red`, `Sweet`> is the least desired outcome since it involves the least desired values for the attributes. That is, according to the CPT for `Color`, the least desired value is `Red` while `Sweet` and `Chianti` are the least desired values for `Sugar Level` and `Region` respectively when the color is `Red`. The CPT for `Color` shows that `White` is preferred over `Red`. Thus, we can apply an improving flip from <`Chianti`, `Red`, `Sweet`> to <`Chianti`, `White`, `Sweet`> by changing the value of wine color. If we cannot reach one outcome from another via improving flip sequences and vice versa, we cannot compare these two outcomes. For instance, we cannot compare <`Chianti`, `Red`, `OffDry`> and <`California`, `Red`, `Sweet`> according to the CP-net in Figure 2.1 since there is not any possible improving flip sequence between them. The fact that we may not be able to compare some outcomes is one of the challenges of using CP-nets in negotiation.

Concluding, it is intuitive and natural to express the user's preferences as a CP-net. It enables to represent the preferences qualitatively and handle conditional preferences. On the other hand, it provides a partial ordering of outcomes [31, 32]. Thus, we are unable to compare some outcomes. The following quantitative preference representation, linear additive utility function provides a total ordering so we can compare each outcome pair. Linear additive utility functions are widely used in negotiation [33–36].

2.3. Linear Additive Utility Functions

A utility function maps outcomes to real values (their utilities), for every $o_i \in \Omega$ $U(o_i) = o_i \rightarrow \mathbb{R}$. In preference relation if we have $o_1 \succeq o_2$, the utility function reveals that $U(o_1) \geq U(o_2)$. A linear additive utility function is a special form of utility functions that is a weighted sum of individual utility functions over single attributes [11]. It provides a compact representation by means of additive preference independence assumption [28].

If and only if the attributes are mutually preferentially independent, an additive utility function represented in Equation 2.1 exists where we have n attributes $X_1, X_2, ..., X_n$ and $U(X_i)$ denotes the utility function over i^{th} attribute [37].

$$U(X_1, X_2, ..., X_n) = \sum_{i=1}^{n} U_i(X_i) \tag{2.1}$$

Mostly the utility functions are scaled between zero and one. For each individual utility function on single attributes, we define a weight indicating the importance of that attribute. The sum of the weights are equal to one and each weight is greater than zero. Thus, the utility function takes the form in Equation 2.2 where w_i denotes the weight for the i^{th} attribute, X_i.

$$U(X_1, X_2, ..., X_n) = \sum_{i=1}^{n} w_i * U_i(X_i) \tag{2.2}$$

In our wine example, if we assume that wine color, region and sugar level are mutually preferentially independent, we can define an additive utility function for wine such as $U(wine) = w_c * U_c(\texttt{Color}) + w_r * U_r(\texttt{Region}) + w_s * U_s(\texttt{Sugar Level})$. Unlike CP-nets, we can compare each outcome pairs in the space of all possible outcomes. However, we cannot handle conditionals. That is, we cannot represent if the wine is red, we prefer a dry wine to a sweet wine but if it is rose, a sweet wine is preferred over a dry wine. The next representation, UCP-Network, allows us to represent conditional preferences quantitatively.

2.4. UCP-Networks (UCP-nets)

A more general form of additive independence namely generalized additive independence (GAI) allows us to express conditionals and meanwhile takes the benefits of using the additivity [28]. We define $Z_1, Z_2, ..., Z_k$ as the subsets of X (a set of attributes) such that they do not have to be disjoint but the union of them should reveal X. These subsets are generalized additive independent if and only if we have a function formed as Equation 2.3 where k denotes the number of additive factors and $f_i()$ is the i^{th} fac-

tor [38, 39].

$$U(X_1, X_2, ..., X_n) = \sum_{i=1}^{k} f_i(Z_i) \tag{2.3}$$

Boutilier *et al.* propose a graphical preference model namely UCP-net [27] by combining CP-nets with generalized additive independence models (GAI-models). Hence, UCP-nets are able to represent preferences in a quantitative way rather than representing simply preference ordering. GAI-models [39] perform dominance queries (whether an outcome would be preferred over another) straightforwardly whereas CP-nets perform outcome optimization queries (maximal outcome) straightforwardly. Therefore, it is claimed that by the combination of both models, UCP-nets become more powerful [27].

Figure 2.2 shows an example UCP-net that is consistent with CP-net seen in Figure 2.1. Similar to CP-nets, we specify preferential dependency among attributes. Instead of specifying a total preference ordering over the values of each attribute according to their parents' values (conditions), we assign a real value (utility) for all values of each attribute by taking conditions into account. For instance, when wine is `Red`, the utility of `Bordeaux` as a wine region is specified as 0.9. Notice that the real values are not restricted to be between zero and one in this model. For example, the utility of `Rose` is equal to 4.0.

The utility function $u(X_1, X_2, ...X_n)$ is represented in Equation 2.4 where X_i is the i^{th} attribute of the outcome, U_i denotes a set of parents of X_i and $f_i(X_i, U_i)$ represents a factor. Note that each different factor is treated as generalized additive independent of other factors. For example, our sample UCP-Net involves three factors f_1(`Color`), f_2(`Region`, `Color`), f_3(`Sugar Level`, `Color`). The utility of an outcome is estimated as the sum of these factors. For example, the utility of <`Rose`, `California`, `OffDry`> is equal to 5.85 ($= 4.0 + 0.95 + 0.90$).

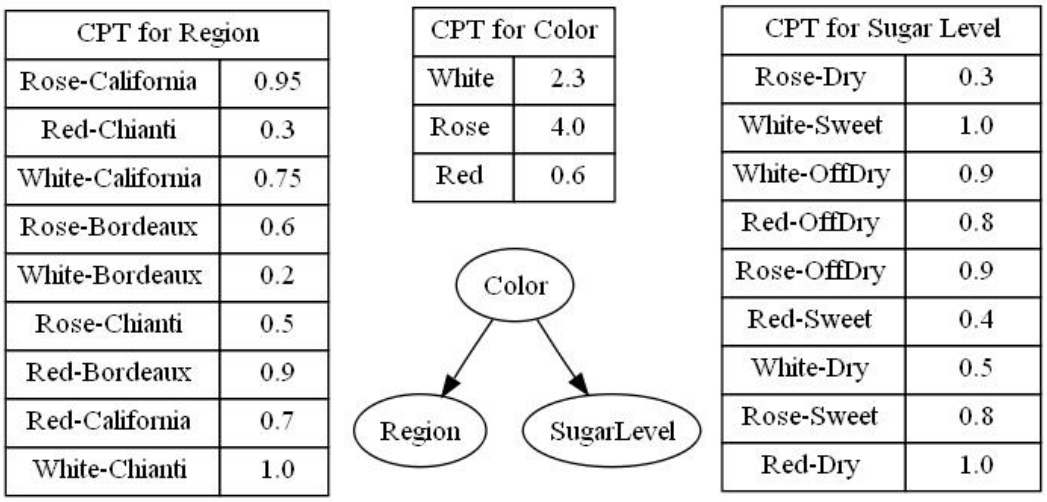


Figure 2.2. Sample UCP-net.

$$u(X_1, X_2, ...X_n) = \sum_i f_i(X_i, U_i) \tag{2.4}$$

In CP-Nets, it is implicitly induced that an ancestor has higher priority over its descendants. Note that this property constitutes a key role in UCP-nets in which each attribute should dominate its children. When we assign utilities, we need to ensure that each node dominates its children. There are several ways of verifying whether the given network is a valid UCP-net (whether it satisfies the CP-relations among attributes).

One of the methods for verifying UCP-nets is to compute the values of `Maxspan` and `Minspan` for attributes and check whether Equation 2.5 is satisfied or not. In this equation R and T are attributes and T_i are the children of R. Maxspan(T) is the maximum difference between the utilities of each possible values of T for the given parent value. We can define Minspan(R) analogously. That is, MinSpan(R) is the minimum difference between the utilities of each possible values of R.

$$Minspan(R) >= \sum_i Maxspan(T_i). \tag{2.5}$$

If the condition in Equation 2.5 is satisfied for each attribute in the given network, we can say that it is a valid UCP-net. For example, in our sample UCP-net `Color` is a parent of both `Region` and `Sugar Level`. For estimating the value of Maxspan(`Region`), we evaluate the maximum difference between the utilities of the values of `Region` for each possible value of its parent [`Red`: 0.6 $(0.9 - 0.3)$, `Rose`: 0.45 $(0.95 - 0.5)$ and `White`: 0.8 $(1.0 - 0.2)$] and choose the maximum among them (Maxspan=0.8). Similarly we get the Maxspan(`Sugar Level`) as 0.6. To be able to get a valid UCP-net satisfying the CP-relationships, the difference between the utilities of each possible value of `Color` should be at least 1.4 $(= 0.6 + 0.8)$. In our example, it is equal to 1.7 $(2.3 - 0.6)$, which is higher than 1.4. We check this condition for each attribute. If the given UCP-net satisfies this condition for each attribute (CP-relationships among attributes), we say that the given UCP-net is a valid UCP-net.

Table 2.1 shows the comparison of the preference representations with respect to a number of criteria:

- category: whether the representation is quantitative or qualitative,
- structure: which category the representation belongs to in terms of its structure,
- ordering: whether the representation is able to represent a total ordering or a partial ordering,
- capturing conditions: whether the representation is able to handle conditional preferences or not.

Table 2.1. Comparison of preference representations.

Representation	**Category**	**Structure**	**Ordering**	**Capturing conditions**
C&D constraints	Qualitative	Binary	No	Yes
CP-nets	Qualitative	Ordinal	Partial	Yes
Additive functions	Quantitative	Cardinal	Total	No
UCP-nets	Quantitative	Cardinal	Total	Yes

In negotiation, we need a preference representation that is expressive enough while its elicitation is simple for the user. As far as the human users are concerned, qualitative

preferences are more intuitive and natural. In general, users are unwilling to express their preferences quantitatively. Thus, we prefer negotiating with qualitative preferences. Since there exists some dependencies between issues in real life situations, it is desirable to be able to represent conditional preferences. When a negotiating agent generates its offers or decides whether a counter-offer is acceptable, it needs to compare some outcomes. This comparison can be performed by using the ordering captured by the preference representations as well as by checking whether the outcome is acceptable or not. In more complex negotiations, the ordering of outcomes plays a key role. Therefore, a total ordering of outcomes is required for the negotiating agents to compare each outcome in order to negotiate effectively.

3. NEGOTIATION ARCHITECTURE

In this section, we give a brief information related to negotiation basics and explain our negotiation architecture in which two agents negotiate on the content of the service.

3.1. Negotiation Basics

There are several issues making up the negotiation process such as the negotiation protocol, participants, strategy and negotiation domain. Brief explanations of these negotiation terms are:

- Negotiation domain (object): A negotiation domain represents a set of issues that the participants try to agree on [10, 33]. A negotiation issue can be defined as "a particular interest in a negotiation" such as price, neighborhood, and so on [33] and a negotiation domain may consist of one issue or multiple issues. In other words, the participants can negotiate on a single issue or multi-issue [11]. For instance, a consumer and a producer may negotiate only on "price" issue whereas they may negotiate on several issues related to the service such as the content of the service, the delivery time and so on.
- Negotiating parties (participants): A negotiating party can be a human or an autonomous agent that takes a part in a negotiation [33]. Since in this thesis we only study automated negotiations, our negotiating parties are the intelligent software agents that negotiate on behalf of their users. We can classify the negotiations with respect to the number of participants that are involved. If we have two agents negotiating, it is a *bilateral* (one-to-one) negotiation [40]. If a negotiation involves more than two participants, it is called a multilateral negotiation [21, 41]. During this thesis, we study bilateral automated negotiations.
- Bid: A bid is an offer or a counter-offer made by one of the negotiating parties during the negotiation.
- Negotiation outcome: A negotiation outcome represents the mutually accepted agreement at the end of the negotiation when the negotiation is terminated with an agreement.

- Negotiation session: A negotiation session is a single process of negotiation between parties on a particular matter that ends up with a consensus or failure.
- Negotiation strategy: A negotiation strategy determines how a negotiating party acts during the negotiation. It dictates how offers are generated and which counter-offers are acceptable. When there are huge number of all possible outcomes, an important challenge is to find ways to generate offers or counter-offers. This is determined by the negotiating party's strategy. In literature, there are a variety of strategies such as concession-based strategies, trade-off strategies and so on. In a concession-based strategy, the negotiating party concedes over time. That is, the utility of the party's next offers is less than the utility of its previous offers. The amount of concession changes with respect to the concession-based strategy employed by the negotiating party. In real life, there may be other factors affecting the behavior of the agents such as a deadline, the opponent's preferences and so on. For instance,the agent may have a tendency to concede more quickly, when its deadline for the negotiation becomes closer [21, 41]. In a trade-off strategy, a negotiating party demands more on some issues while concedes on other issues without changing its overall utility if it is possible [34]. For instance, the producer may increase the price of the product for early delivery. Since the importance of the issues may be different for the negotiating parties, a number of trade-offs increasing the social welfare can be generated.
- Negotiation protocol: A negotiation protocol manages the interaction between negotiating parties by determining how the parties interact, how the bids exchanges are done, what the valid bids are, when the negotiation closes and so on [10]. The negotiating parties should agree on the negotiation protocol before starting negotiation.

3.2. Proposed Negotiation Architecture

We are interested in bilateral automated service negotiations in which a consumer and a producer negotiate on a particular service in a distributed environment. In such a dynamic and open environment, the participants may meet for the first time and may not know each other well. Besides, the participants may not prefer to reveal their preferences to each other because of the possibility of the abuse of their own preferences by the other

participant. That is, the other participant may exploit the shared preferences towards the participant sharing its preferences and generate offers accordingly. Thus, in our negotiation settings, it is assumed that neither the producer nor the consumer knows each other's preferences.

The consumer and the producer can represent their own preferences in a variety of ways. For instance, the consumer may represent her preferences with conjunctive and disjunctive constraints or may express her preferences as a CP-net while the producer may assign utilities to the service outcomes by using a utility function. The choice of the preference representations may vary based on service or the participant's role in the negotiation.

In a distributed environment, the range of services is broad. A service can be selling a wine, arranging a holiday, giving a private lecture, and so on. For some particular services, it may be enough to express which service is acceptable or unacceptable while for some services, the participants may need to specify an ordering such as a service outcome may be preferred over another one although both service outcomes are acceptable. The consumer may prefer to express her preferences qualitatively while the producer may have a tendency to represent his preferences quantitatively via utilities.

Moreover, the producer may have a service inventory involving the available services or all possible services that the producer can provide. The service inventory can be physical as well as it can be conceptual. In the case of selling a wine or a book, the producer may have a physical inventory and negotiate with the consumer by taking its inventory into account. Under the circumstances, the producer can provide only a service if he has in his inventory. On the other hand, if the service is giving a private lecture or arranging a possible holiday, the service inventory serves as a concept of all possible services. In such a case, the producer can offer any possible services.

Figure 3.1 depicts our negotiation architecture for the automated bilateral service negotiation. Since we deal with the automated negotiation in which intelligent software agents negotiate on behalf of their users, our main components are consumer and producer agents, which communicate with each other to perform a negotiation over the

service itself (the content of the service).

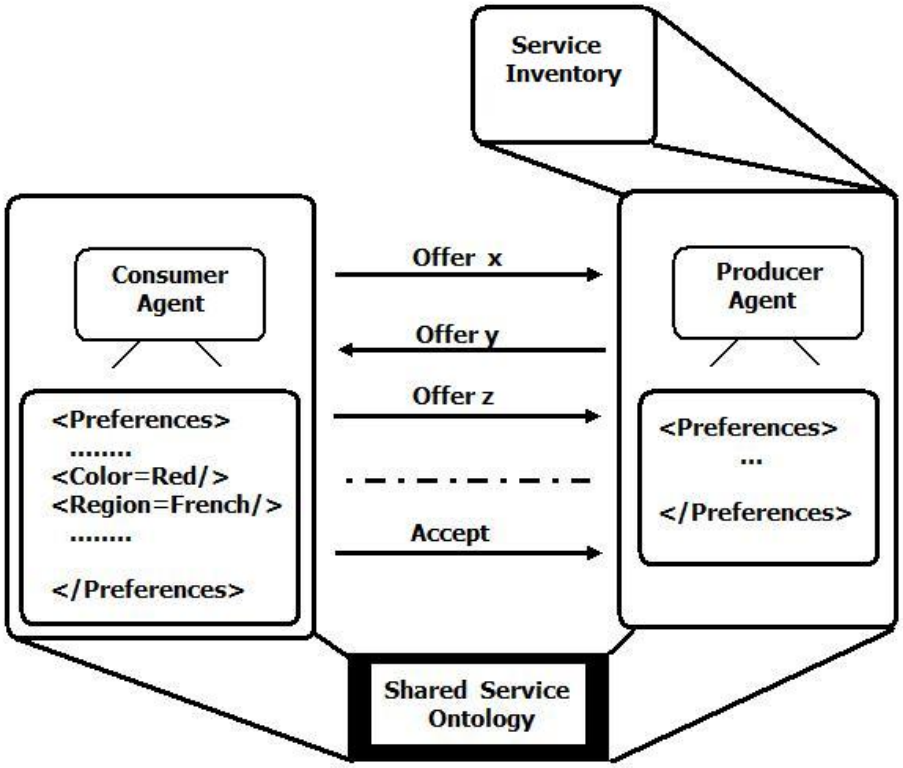


Figure 3.1. Bilateral service negotiation architecture.

An alternating offers protocol [42, 43] is adapted to govern the interaction between negotiating parties. According to this protocol, a negotiating party initiates the negotiation with an offer and the other party either accepts or rejects. If that negotiating party accepts the offer, the negotiation ends with an agreement. Otherwise, it makes a counter-offer. In our case, the consumer agent initiates the negotiation with a particular offer consistent with its preferences and the producer agent responses by providing the requested service or making an alternative offer. When it is the consumer agent's turn, it may make a new offer or stick to its previous offer. The negotiation continues in a turn-taking fashion until having a consensus or reaching a termination condition such as a deadline or no available counter-offers.

A shared ontology provides the necessary vocabulary and hence enables a common language for the agents. If the agents do not have a shared ontology, their negotiation offers and counter-offers will not make sense to each other or will be partially understood by the other agent. Since our focus is on the negotiation, we assume that a shared

ontology exists. This ontology describes the content of the service. Further, since an ontology can represent concepts, their properties and their relationships semantically, the agents can reason on the details of the service that is being negotiated. The negotiation setting is independent of the ontology used so that the service can be anything described by the ontology. A service can be represented as a vector of attribute (issue) values. For instance, a wine service can be represented as <`Chianti`, `White`, `Off-Dry`> representing a white and off-dry wine produced in Chianti region.

Figure 3.2 shows an abstract view of a negotiating agent. The agent negotiates with respect to its user's preferences. The agent only has an access to its user's preferences and does not know other agent's preferences. Its negotiation strategy determines how the agent generates its offers (bid generator module) and how the agent decide which counter-offer is acceptable (bid decider module). The agents' negotiation strategy is embedded inside the agents and kept as private as seen in Figure 3.2. According to our negotiation architecture, the consumer and the producer agents may employ any negotiation strategies. For instance, the agents may employ a concession-based negotiation strategy in which the agent starts offering the most desired service and concedes over time.

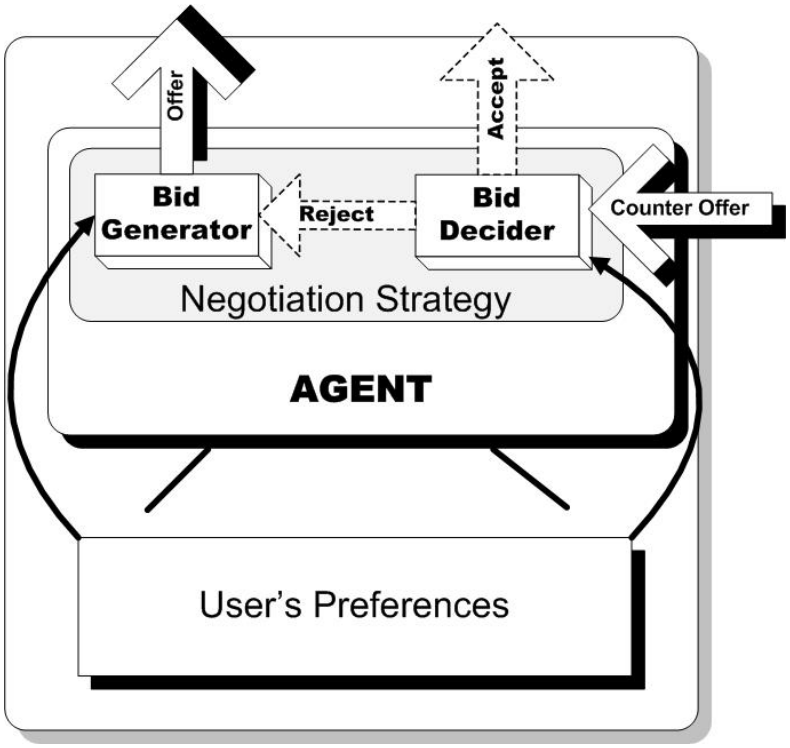


Figure 3.2. A negotiating agent.

In this negotiation setting, the participants can be self interested or collaborative. A self interested agent considers only its own preferences and negotiates regardless of the other agent's preferences. The consumer agent is mostly self interested in our settings. That is, it needs a service based on its user's preferences and tries to get the most preferred service that it is possible. The producer agent may also be self interested and generate offers with respect to its own preferences. However, the producer agent is mostly collaborative in order to provide its services to the consumer. It tries to meet the consumer's needs as well as satisfying its own preferences.

To make the best possible offer that is available in its service inventory, the producer agent does not only consider its own preferences but also takes into account the consumer's preferences. Since the consumer's preferences are private and cannot be reachable by the producer, the producer agent can try to predict the consumer's preferences from bid exchanged during the negotiation. Consequently, the producer agent can generate well-targeted counter-offers.

Our negotiation architecture allows us to investigate a variety of negotiation settings. We specifically study the following:

- Case 1: The consumer agent represents its preferences as a CP-net and generates its offers with respect to the ordering induced from its CP-net while the producer agent generates its counter-offers by only considering its own preferences and all possible services defined in its conceptual service inventory (Chapter 4 and Chapter 5).
- Case 2: In this case, the consumer agent has a UCP-net instead of a CP-net and negotiates with respect to its UCP-net. Similar to the second case, the producer agent has a conceptual service inventory and generates its counter-offers with respect to its own preferences (Chapter 5).
- Case 3: The producer agent predicts the consumer's preferences and accordingly generates its offer from its physical service inventory by considering both its own preferences and the consumer's predicted preferences while the consumer agent negotiates with its preferences as conjunctive and disjunctive constraints (Chapter 6 and Chapter 7).

4. NEGOTIATION WITH QUALITATIVE PREFERENCES

Representing and reasoning about the user's preferences play a key role in automated negotiation. Most of the negotiation strategies are developed with the assumption of having the user's quantitative preferences as utility functions. The reason is that it is straightforward to compare outcomes and find mathematical models in order to generate the offers and counter-offers with quantitative preferences. On the other hand, the users are reluctant to assess their preferences as numerical values and mostly prefer to express their preferences qualitatively, which is more natural and intuitive for them [19]. Therefore, it is crucial to be able to negotiate with qualitative preferences.

In this section, we study how the consumer agent uses qualitative preferences in negotiation. For this purpose, we consider two qualitative representations and present negotiation approaches with these preferences as follows:

- Conjunctive and disjunctive constraints (C &D constraints): The consumer agent has its user's preferences as conjunctive and disjunctive constraints and generates service offers consistent with these constraints. It is worth noticing that the consumer agent employs a purely qualitative negotiation strategy in this approach.
- Conditional preference networks (CP-nets): The consumer agent has its user's preferences as a CP-net and applies a heuristic to produce estimated utilities, which will be used by any negotiation strategy based on utilities. Although the consumer has qualitative preferences, it employs a negotiation strategy using utilities. To do this, it transforms qualitative preferences to quantitative preferences by applying a heuristic.

4.1. Negotiation with C &D Constraints

In this approach, the consumer's preferences are represented via conjunctive and disjunctive constraints. Thus, the consumer agent tries to generate service offers satisfying these constraints. Recall that a disjunctive constraint consists of a number of conjunctive constraints and in order to satisfy a disjunctive constraint, it is enough to

satisfy at least one conjunctive constraint. For instance, if the consumer prefers a red and dry wine or a white wine ($<?, \texttt{Red}, \texttt{Dry}> \cup <?, \texttt{White}, ?>$), any red and dry wine as well as any white wine would be acceptable for the consumer.

In order to generate its service offers, the consumer agent randomly selects a conjunctive constraint from its disjunctive constraint. Then, it assigns consistent values for the issues that are specified in this conjunctive constraint. For the unspecified issues, the consumer randomly chooses possible values defined in their domain. For instance, consider that the consumer has the following preference: {<?, Red, Dry> ∪ <?, White, Sweet>}. First, the consumer chooses one of the constraints randomly. If the consumer selects the first constraint, it will initialize the particular values specified in the constraint as Red and Dry. For the region issue, it will randomly pick up one of the domain values, {Bordeaux, Chianti, California} . Assume it picks Bordeaux, then the consumer's offer will be <Bordeaux, Red, Dry>.

When the consumer receives a counter-offer from the producer, it evaluates this offer according to its preferences. If the offer satisfies the consumer's preferences, the consumer will accept the offer and the negotiation will end up with a success. Otherwise, the customer generates a new offer with respect to its preferences or stick to its previous offer. This process will continue until a service offer is accepted by the consumer agent or all possible offers are put forward to the consumer by the producer.

Concluding, it is straightforward to negotiate with conjunctive and disjunctive constraints. Although it is reasonable to represent the users' preferences in this way for some negotiation domains and negotiate accordingly, it may be insufficient to represent the consumer's preferences with constraints for other domains. That is, the user may need to specify an ordering among alternative services. For instance, there can be two services that are acceptable for the consumer but if the consumer receives the first service, she may be more satisfied — the first service is preferred over the second service. Conditional preference networks (CP-nets) enables the consumer to express such kind of preferences and the next section explains how the consumer agent can negotiate when it has its user's CP-net.

4.2. Negotiation with CP-nets

As an alternative to conjunctive and disjunctive constraints, CP-nets can be used for the consumer's preferences. Although CP-nets are compact representations for qualitative preferences and are able to handle conditional preferences, they almost always represent a partial ordering [31, 32]. Since the consumer agent is mostly able to induce a partial ordering from its CP-net, there will be some service outcomes that the consumer cannot compare. However, there are two properties of CP-nets that make it difficult to be used for negotiation. The consumer agent needs to have a total ordering in order to compare each outcome pair and negotiate effectively. Most of the negotiation strategies [20–24, 41] work with quantitative preferences such as utility functions but the CP-nets do not have utilities.

To deal with these challenges, we propose to use heuristics with CP-nets to approximate a total ordering by means of estimated utilities and enable the consumer to negotiate with respect to the estimated total ordering. Consequently, the consumer agent uses a qualitative preference representation, an acyclic CP-net, while it still negotiates with its strategies using quantitative information, utility.

Here, the consumer agent is free to employ any negotiation strategy based on utilities unless the strategy use the structure of the utility functions explicitly. In our negotiation setting, the consumer agent employs a simple concession-based strategy. That is, the consumer starts offering the service outcome having the highest utility and concedes over time. To do this, it considers the estimated utilities generated by the applied heuristic. It also evaluates the producer's counter-offer with respect to the estimated utilities. If the utility of the producer's counter-offer is higher than or equal to the consumer's previous offer, the consumer agent accepts the producer's counter-offer. During the negotiation, the consumer agent also keeps track of the best counter-offer that is made by the producer. While proposing an offer, the consumer agent also considers the producer's best counter-offer. If the utility of the current offer is lower than the utility of the producer's best counter-offer, the consumer requests the producer's best counter-offer.

Without a doubt, the estimated utilities have a significant impact on the performance of the consumer's negotiation outcome. If the estimated utilities are far from the real utilities, the consumer agent will propose inadmissible offers that are not really desired by the consumer. This may result in unsuccessful negotiation results from the consumer's point of view. For instance, an unacceptable offer may be accepted by the consumer agent since the estimated utility of that outcome is not accurate. Thus, the heuristic used for generating the estimated utilities plays a key role when the consumer has a CP-net. In the next section, we present several heuristics to estimate the utilities from the consumer's CP-net.

4.3. Proposed Heuristics for Acyclic CP-nets

We present several heuristics to estimate utilities from a given CP-net. To do this, we induce a preference graph after eliciting the consumer's preferences as a CP-net. Since the proposed heuristics use the induced preference graph whose nodes denote possible service outcomes and edges represents *improving flips*, it is crucial to figure out how the preferences graph is induced from a CP-net.

To induce the preference graph, we first start with the least desired (worst) outcome, which will be placed at the top of the preference graph (root node). Then, we apply improving flips and draw a directed edge for each improving flip. A directed edge from one node to a second node shows that the second node is preferred over the first node. After applying all possible improving flips, we reach the optimal (best) service outcome that will be placed at the leaf node and complete generating the preference graph. Recall that there is only one best outcome in acyclic CP-nets. For intermediate nodes, we can only compare the nodes having a path from others. The nodes having no path to each other cannot be compared under the "everything else being equal" assumption.

To illustrate this, consider our simplified holiday domain that has three attributes with a limited a set of possible values per attribute. These are `Hotel Location`, `Season`, and `Duration`. `Hotel Location` can be near to `Sea` or `Historical Place` or `Mountain`. For `Season` there are two values: `Summer` and `Winter`. There are three possible values for `Duration`: `One week`, `Two weeks` and `Three weeks`. Imagine that the consumer

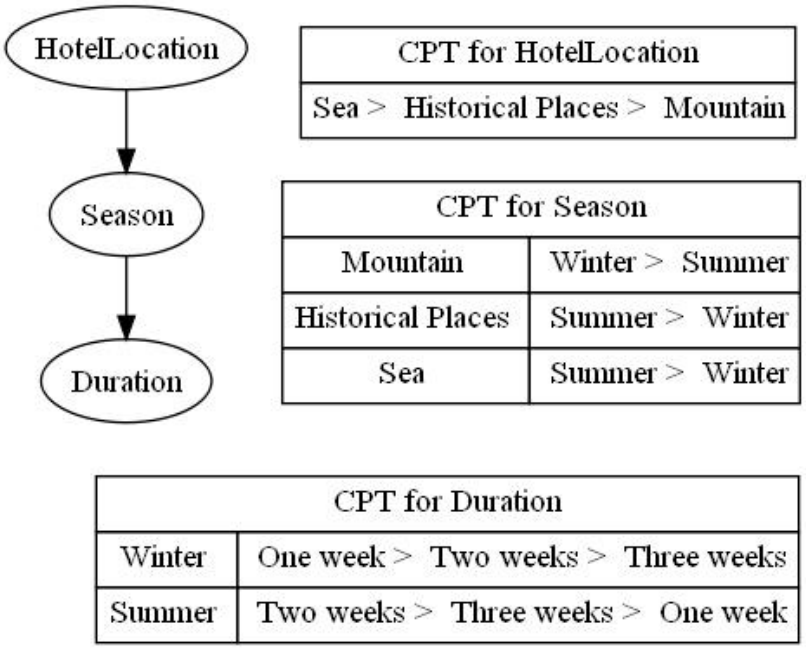


Figure 4.1. A sample CP-net for our simplified holiday domain.

expresses her preferences as the CP-net drawn in Figure 4.1.

As seen from the CP-net, the consumer's preferences on `duration` depend on `season` and her preferences on `season` depend on `hotel location`. For hotel location, the consumer prefers a hotel near a sea over a hotel near historical places and a hotel near historical places is preferred over a hotel near a mountain (`Sea` > `Historical Places` > `Mountain`). If the hotel is near a mountain, she prefers a holiday in winter rather than in summer (`Mountain`: `Winter` > `Summer`). Otherwise, a holiday in summer is preferred over a holiday in winter ([`Sea` ∨ `Historical Places`]: `Summer` > `Winter`). When it is winter, a one-week holiday is preferred over a two-week holiday and a two-week holiday is preferred over a three-week holiday (`Winter`: `One week` > `Two weeks` > `Three weeks`). When it is summer, a two-week holiday is preferred over a three-week holiday and a three-week holiday is preferred over a one-week holiday (`Summer`: `Two weeks` > `Three weeks` > `One week`).

We induce the preference graph in Figure 4.2 from the CP-net in Figure 4.1. In this graph, the node <`Mountain`, `Summer`, `One week`> represents a one-week holiday in a hotel near a mountain during summer. This is the least desired outcome according

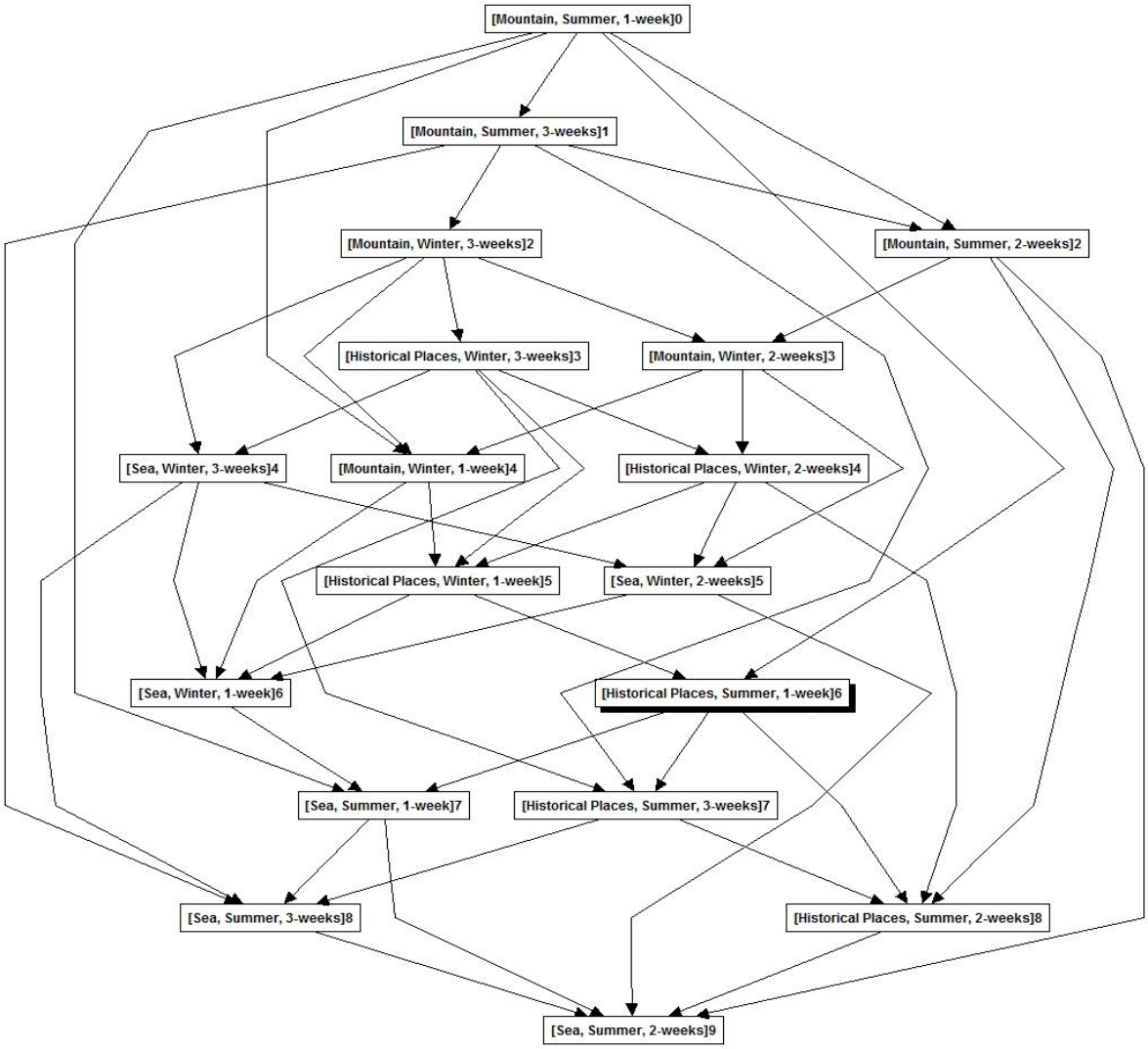


Figure 4.2. Induced preference graph from the CP-Net in Figure 4.1.

to the given preferences. It is easy to find the least desired outcome. We only need to choose the least desired values for each attribute by starting from parent attributes in the CP-net. While constructing the preference graph, we draw an edge from a node to another, if there is an improving flip from the former to the latter. It is seen that there is an edge from <`Mountain`, `Summer`, `One week`> to <`Sea`, `Summer`, `One week`>. It can be inferred that a one-week holiday in a hotel near a sea in the summer is preferred over a one-week holiday in a hotel near a mountain in the summer. Recall that we cannot compare two outcomes if there are no paths (any sequences of improving flips) between them. For instance, <`Mountain`, `Summer`, `Two weeks`> and <`Mountain`, `Winter`, `Three weeks`> are not comparable according to the preference graph. In the induced preference graph, the leaf node represents the most desired outcome where the root node denotes the least desired outcome.

It is worth to discuss the complexity of inducing a preference graph from an acyclic CP-net. The number of nodes grows exponentially with the number of attributes. If $\mid Dom(x_i) \mid$ denotes the number of possible values for the i^{th} attribute, then the total number of nodes is equal to $\mid Dom(x_1) \mid * \mid Dom(x_2) \mid * ... * \mid Dom(x_n) \mid$ where we have n attributes. The total number of edges in the graph is equal to the number of all possible improving flips. For example, there can be at most $(\mid Dom(x_1) \mid -1) * (\mid Dom(x_2) \mid -1) * ... * (\mid Dom(x_n) \mid -1)$ outgoing edges of the root node. Note that the number of outgoing edges of the nodes are decreasing as the nodes become close to the leaf node and the number of outgoing edges of the leaf node becomes zero. Since we start from the least desired outcome and continue with applying improving flips, it is straightforward to generate the preference graph.

Although inducing a preference graph from an acyclic CP-net seems to be costly and time consuming, during the negotiation the agents will not be affected by this process because we construct the preference graph once while we elicit the user's preferences as a CP-net. Then the proposed heuristics will be applied to the induced preference graph in order to get the estimated utilities that will be used by the agent's strategy during the negotiation. The heuristics that we develop are:

- Depth Heuristic (Section 4.3.1)
- Depth Level Heuristic (Section 4.3.2)
- Taxonomic Heuristic (Section 4.3.3)
- PageRank Heuristic (Section 4.3.4)
- Preferred Outcomes Heuristic (Section 4.3.5)
- Average All Heuristic (Section 4.3.6)

4.3.1. Depth Heuristic (DH)

In this heuristic, we use the concept of *depth* to produce estimated utilities of the outcomes. The depth of an outcome node in a preference graph indicates how far it is from the least desired outcome; in other words the longest distance from the worst outcome. It is intuitive to say that the better an outcome is, the further it is from the worst outcome. The depth of an outcome node is estimated as the length of the longest

path between the root node and that node.

The intuition here is that if there is an edge from x to y, we know that y is preferred over x and the depth of y is higher than that of x. According to this approach, the higher the depth of an outcome, the more likely it is preferred by the user. Further, if two outcomes are at the same depth, it is assumed that these outcomes are equally preferred by the user. We apply Equation 4.1 to estimate the utility values between zero and one. In short, the depth of a given outcome is divided by the depth of the preference graph (the highest depth) to obtain the estimated utility of that outcome. For example, consider the preference graph in Figure 4.2 that we can see the depth of each node explicitly. If we have this preference graph with a depth of 9, an outcome whose depth is equal to 4 will have utility of 0.44 (= 4/9) according to this approach. Since we update the depth of each node while inducing the preference graph, it is straightforward to estimate the utility of an outcome with respect to this heuristic.

$$U(x) = \frac{Depth(x, PG)}{Depth_{PG}} \tag{4.1}$$

It is worth noticing that the estimated utilities by the Depth Heuristic satisfy the existing CP-net relations. That is, for two outcomes o_i, o_k where o_k is preferred over o_i ($o_k \succ o_i$) with respect to a given acyclic CP-net, the utility function $U(x)$ defined in Equation 4.1 holds that $U(o_k) > U(o_i)$. With respect to CP-net semantics, an outcome o_k is preferred over o_i if there is an improving flip sequence from o_i to o_k [19]. According to our heuristic, the depth of o_k will be higher than the depth of o_i if there is an improving sequence from o_i to o_k. Therefore, if an outcome o_k is preferred over another o_i with respect to a given acyclic CP-net, the estimated utilities by the Depth Heuristic hold that $U(o_k) > U(o_i)$.

4.3.2. Depth Level Heuristic (DLH)

As an alternative to Depth Heuristic (DH), we can take the number of outcomes at each depth into account and distribute the utility functions according to the density of the outcomes at each depth. This method takes the depth of an outcome and outputs a utility value between zero and one. A candidate utility function is formalized in Equation 4.2 where $U(i)$ denotes the utility of outcomes at depth i, S_i is the number of outcomes at depth i and N is the number of all possible outcomes. We define the utility of the outcome at depth zero ($U(0)$) as $1/N$.

$$U(i) = U(i-1) + \frac{1}{N} * S_i \tag{4.2}$$

It is worth noting that the Depth Level Heuristic gives the same ordering as the Depth Heuristic because $U(i) > U(i-1)$ but the magnitude of the estimated utilities are different. This may lead to different negotiation results even if the agent applies the same negotiation strategy. For example, if the negotiation strategy only generates offers whose utility is higher than a predefined threshold value (reservation point), the agent applying DLH may generate different offers than the agent applying DH. However, this is not true for some negotiation strategies, which consider the ordering of outcomes rather than to their extent. Consider a simple concession-based strategy in which the agent starts with the outcome having the highest utility and concedes over time (the second highest utility, the third highest utility and so on) up to reaching a consensus. Under these circumstances, the negotiation result would be the same for both DH and DLH.

4.3.3. Taxonomic Heuristic (TH)

The previous heuristics (DH and DLH) consider that an outcome at a higher depth gets higher utility. That is, if there are two outcomes and the depth of the first outcome is higher than the depth of the second outcome, the utility of first outcome will be higher than the utility of the second outcome with respect to those heuristics. They also assume

that outcomes at the same depth are equally preferred by the user. In contrast, TH does not make these assumptions. According to this heuristic, the utility of an outcome is determined randomly but the estimated utilities should be *consistent* with CP-net semantics. That is, if there is a sequence of improving flips from the first outcome to the second outcome, the utility of the second outcome should be higher than the utility of the first outcome. Since there exist a sequence of improving flips from any node to its descendants, the descendants are preferred over the ancestors in the preference graph. In short, we generate the utility of an outcome randomly in accordance with the CP-net semantics.

Equation 4.3 shows how the utility of an outcome (o_i) is estimated randomly. The utility of the worst outcome is set to zero directly. For other outcomes, we consider the highest utility of their ancestors and increase this a random amount. Note that we calculate the utility of outcomes at lower depth earlier because we need to check ancestors' utility while estimating descendants and the depth of ancestors is lower than the depth of descendant.

$$U(o_i) = Max(U(Ancestors(o_i)) + randomUtility(0.0, 1.0) \tag{4.3}$$

As formulated in Equation 4.4, before using these estimated utilities in negotiation we normalize them by dividing them by the utility of the optimal outcome whose utility is the highest .

$$U_{final}(o_i) = \frac{U(o_i)}{U(o_{optimal})} \tag{4.4}$$

4.3.4. PageRank Heuristic (PRH)

PageRank is a heuristic to rank Web Pages. It determines the ranks of the nodes in a graph whose nodes denote Web pages. The underlying intuition is that a Web page is considered important if other important pages link to this page [44]. We apply the same intuition to our problem: An outcome is more preferred if it is pointed by other preferred outcomes in the preference graph — if there are improving flip sequences from other preferred outcomes to that node.

According to this approach, we first apply PageRank algorithm and get PageRank value for each outcome. To estimate the PageRank of the i^{th} service node, we use the formula in Equation 4.5 where $K(i)$ denotes the nodes pointing to the node i, $P(j)$ is the rank of the j^{th} node, N_j is the neighbors of the j and d is the damping factor, which we set to 0.85 as in the original paper [44].

$$P(i) = d \sum_{j \in K_i} \frac{P(j)}{|N_j|} + (1 - d) \tag{4.5}$$

Initially, each service node's PageRank is equal to 1.0. Then, we start the calculation of ranks according to recursive formula in Equation 4.5. When the difference between two calculations is small enough (when the algorithm converges), we reach the final ranks. Since the estimated values may exceed one, we divide each value by the maximum value in order to get normalized values between zero and one. Table 4.1 shows the estimated normalized PageRank values for the preference graph depicted in Figure 4.2.

Because of the nature of PageRank algorithm, the distance between the best outcome and others is so high that this difference may affect the performance of several negotiation strategies negatively. Thus, we do not prefer to use those normalized values directly as utilities.

Table 4.1. Outcomes and their estimated normalized ranks.

Outcome [Hotel Location, Season, Duration]	**Estimated PageRank**	**Normalized PageRank**	**Final Utility**
[Sea, Summer, Two weeks]	1.786	1.000	1.000
[Sea, Summer, One week]	0.827	0.463	0.750
[Sea, Summer, Three weeks]	0.742	0.415	0.738
[Sea, Winter, One week]	0.654	0.366	0.726
[Historical Places, Summer, Two weeks]	0.518	0.290	0.707
[Historical Places, Winter, One week]	0.386	0.216	0.500
[Sea, Winter, Two weeks]	0.360	0.202	0.497
[Historical Places, Summer, One week]	0.340	0.190	0.494
[Historical Places, Summer, Three weeks]	0.324	0.181	0.491
[Mountain, Winter, One week]	0.286	0.160	0.486
[Historical Places, Winter, Two weeks]	0.261	0.146	0.250
[Mountain, Winter, Two weeks]	0.250	0.140	0.249
[Sea, Winter, Three weeks]	0.230	0.129	0.246
[Mountain, Summer, Two weeks]	0.213	0.119	0.243
[Historical Places, Winter, Three weeks]	0.190	0.106	0.240
[Mountain, Winter, Three weeks]	0.187	0.105	0.2398
[Mountain, Summer, Three weeks]	0.175	0.100	0.239
[Mountain, Summer, One week]	0.150	0.084	0.235

To deal with this problem, we apply a clustering algorithm over the estimated normalized PageRank values and make each outcome inside the same cluster have at most a certain distance with others in that cluster. For this purpose, we use a clustering algorithm called X-means [45], which takes a range of number of clusters (minimum and maximum number of clusters) and outputs the best number of clusters and the clustering information (which data belongs to which cluster). Then, we sort the normalized PageRank values and distribute the values according to their clusters.

Equation 4.9 represents how we transform the sorted normalized PageRank values into utilities between zero and one where $U(i)$ denotes the utility of i^{th} outcome, $NPR(i)$ denotes the normalized PageRank value of i^{th} outcome and $c(i)$ represents which cluster the PageRank of i^{th} outcome belongs to. Firstly, we define k equal distance intervals for each cluster where k is the number of clusters that we have. To illustrate this, following our running example, consider that we have 18 outcomes whose normalized PageRank values are shown in Table 4.1. When we apply the clustering algorithm, we obtain four different clusters ($k = 4$). As a result, the interval for each cluster is equal to 0.25. We estimate the upper bound for each cluster by applying the formula in Equation 4.6 where c_i denotes the cluster number. According to this formula, the upper bound for each cluster (1, 2, 3 and 4) is equal to 0.25, 0.50, 0.75 and 1.0, respectively. When we estimate the utility of each outcome whose normalized PageRank value is the highest in their cluster, we use the upper bounds as the utilities of those outcomes (Case -1 Equation 4.9). For example, the outcome whose PageRank value is the highest in the second cluster will get a utility of 0.50.

$$upperBound(c_i) = \frac{1}{k} * c_i \tag{4.6}$$

$$interval(i) = \frac{1}{k * n_i} \tag{4.7}$$

$$dist(i, i+1) = \frac{NPR(i+1) - NPR(i)}{k} \tag{4.8}$$

Since our aim is to assign utilities in a way that the distance between utilities of consecutive outcomes in the same cluster can be at most a certain distance, we define another interval within each cluster by taking the number of outcomes inside that cluster into account. This interval is defined in Equation 4.7 where n_i denotes the number of outcomes in the i^{th} cluster. According to our example, since we have four outcomes in the third cluster, this interval is equal to 0.0625 (= 0.25/4). When we assign utilities to outcomes, we consider both the distance between their normalized PageRank values and the estimated interval value for the current cluster. We calculate the distance between their normalized PageRank values by applying the formula in Equation 4.8 where we divide the distance between PageRank values by the number of clusters. If the estimated distance is greater than the interval value for that cluster, we take the interval value as the distance as specified in Equation 4.9 ($min(interval(c(i), dist(i+1, i))$. As a result, the utility of current outcome is estimated by subtracting this distance from the utility of previous outcome. For example, in the third cluster, we have four outcomes with normalized PageRank values of 0.463, 0.415, 0.366 and 0.290. Since 0.463 is the maximum value in this cluster, it gets a utility of 0.75, which is the upper bound of the third cluster. The distance between the normalized PageRank values, 0.463 and 0.415 is equal to 0.012 according to Equation 4.8. Since it is less than the interval value for this cluster (0.0625), we take 0.012 as the distance. As a result, the utility of that outcome (having a normalized PageRank value of 0.415) will be equal to 0.738 (= 0.75 − 0.012).

$$U(i) = \begin{cases} upperBound(c(i)) & \text{Case-1} \\ U(i+1) - min(interval(c(i)), dist(i+1, i)) & \text{Case-2} \end{cases} \tag{4.9}$$

4.3.5. Preferred Outcomes Heuristic (POH)

In this approach, while we estimate the utility of an outcome, we take the outcomes that are preferred over that outcome into account. For this purpose, we use a scoring function that not only considers the number of outcomes preferred over the current outcome but also takes into account how much they are preferred. This scoring function

is the weighted sum where each weight denotes the importance of an outcome that is preferred over the current outcome. However, we do not know how much each outcome is preferred (importance of an outcome) since we only have the qualitative preferences of the user. Thus, in this approach the depth of an outcome is considered as the importance of that outcome: the higher the depth of an outcome, the more likely it is preferred by the user.

The formula in Equation 4.10 calculates this score for each outcome where $SR(x)$ denotes the score of the outcome x. In this formula, we sum the depth of each outcome y that is preferred over x.

$$SR(x) = \sum_{y>x} depth(y) \tag{4.10}$$

If we divide the estimated scores by the score of the worst outcome (whose score is the maximum), we obtain normalized scores. Since lower scores are more desired, we obtain the utility of an outcome by subtracting the normalized score from one. This utility function is formalized in Equation 4.11 where x is the outcome whose utility is calculated and z is the worst outcome (root node in the preference graph).

$$U(x) = 1 - \frac{SR(x)}{SR(z)} \tag{4.11}$$

4.3.6. Average All Heuristic (AAH)

This heuristic takes an average of the estimated utilities by *Taxonomic Heuristic*, *Depth Level Heuristic*, *PageRank Heuristic* and *Preferred Outcome Heuristic*. It assumes that each heuristic is equally important. Therefore, the final utilities are defined by taking the average of the estimated utilities by these heuristics as drawn in Figure 4.3.

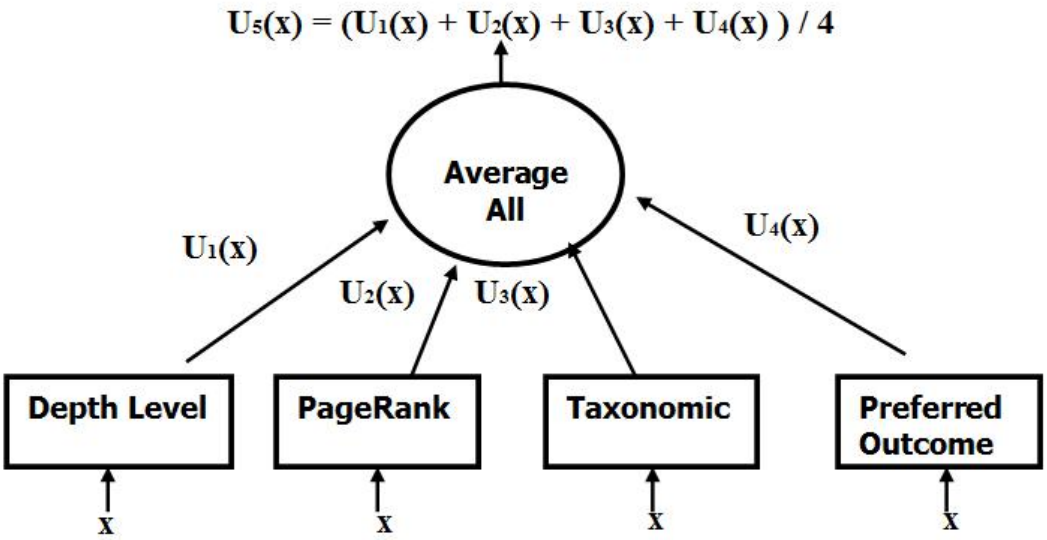


Figure 4.3. How the Average All Heuristic works.

5. EVALUATION OF CP-NETS IN NEGOTIATION

To evaluate the proposed heuristics for negotiation with CP-nets, we extend General Environment for Negotiation with Intelligent multi-purpose Usage Simulation GENIUS [46], which is a platform for bilateral negotiation . The aim of the environment is to facilitate the design and the evaluation of negotiation strategies. GENIUS provides a negotiation simulation environment in which a researcher can setup a single negotiation session or a tournament using various negotiation domains and preference profiles from a repository and choose strategies for the negotiating parties. Furthermore, it allows to create a negotiation domain and preference profiles of the negotiating parties.

Our extension enables a consumer agent to elicit its user's preferences as a CP-net and to use utilities estimated by a chosen heuristic while negotiating with a producer agent. In this setting, the platform also requests the consumer's total ordering of outcomes as a UCP-net and evaluates each negotiation outcome for that agent based on the given UCP-net. The aim of this is to evaluate how well we can generate a total order from a CP-net using heuristics. Ideally, the best we can do is to generate the total order already given by the consumer through a UCP-net. In this regard, the UCP-net serves as ground truth in our comparisons. After a consumer agent negotiates using its CP-net, we evaluate its performance as if we knew the correct total ordering (UCP-net). Note that the given UCP-net is consistent with the given CP-net since both preferences belong to the same consumer.

5.1. Experimental Settings

In order to compare the performance of the heuristics, we investigate three test cases depicted in Figure 5.1. In each test case, the consumer and the producer agents negotiate with each other. We fix both agents' negotiation strategies so that the consumer agent negotiates with the same producer agent (having the same preference profile and strategy).

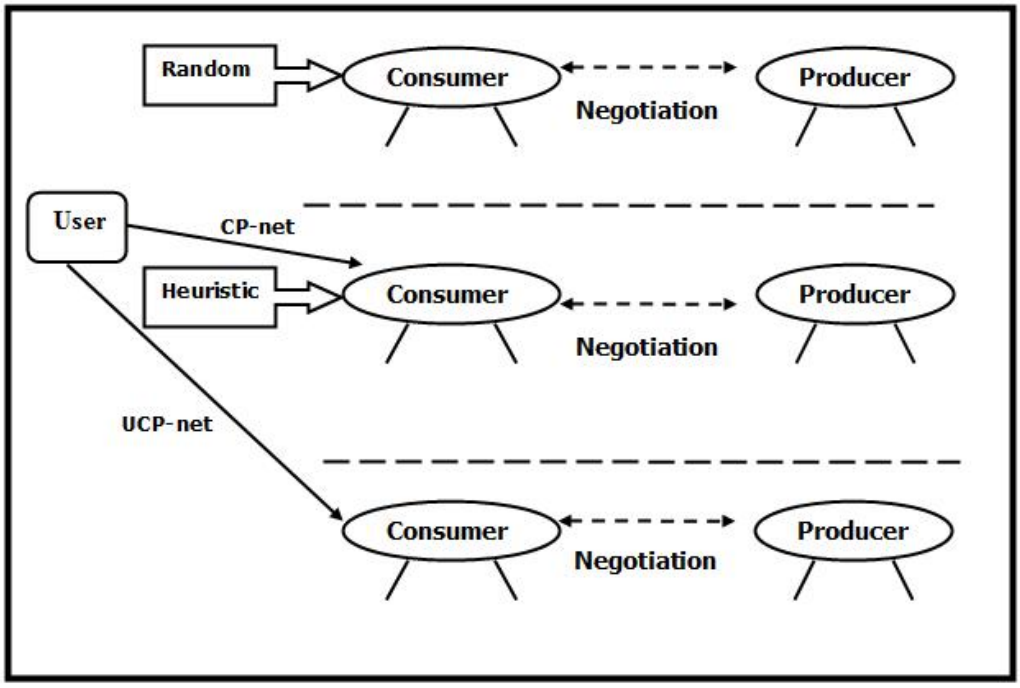


Figure 5.1. Experiment set-up for comparison of heuristics.

In the first case, the consumer agent is a dummy agent that does not know its user's preferences. Thus, it generates estimated utilities purely randomly and negotiates with these estimated utilities. In principle, this corresponds to the Zero Intelligence agent that randomly generates bids [47, 48]. This agent serves as a baseline for our comparisons. In the second case, the consumer agent has a CP-net and applies one of the proposed heuristics (DH, DLH, TH, PRH, POH, AAH) to derive the estimated utilities. During the negotiation, the consumer agent will act according to these estimated utilities. In the third case, the consumer agent has its user's real total preference orderings as a UCP-net. Thus, it uses the real utilities during the negotiation.

Consequently, we are able to observe what the consumer agent gets at the end of a negotiation when it applies heuristics on partial preference information (CP-net) versus when it has total preference information (UCP-net). In our experiments, each negotiation session ends in an agreement that is acceptable for both the consumer and the producer agent. Here, we pay our attention to the consumer agent's negotiation performance. To do this, all negotiation outcomes are evaluated according to the consumer's UCP-net since it serves as a ground truth. We use the UCP-net to assign utilities and thus are able to compare performance quantitatively.

In our experiments, we use the holiday domain defined in Section 5.1.1. Section 5.1.2 gives a brief information about how we elicit the performance profiles for the consumer and producer agents. Lastly, we explain which negotiation strategies that the consumer and the producer agents employ in the negotiation in Section 5.1.3.

5.1.1. Negotiation Domain

For our experiments, we define a holiday domain, which has 216 possible service outcomes. For simplicity our holiday domain has six attributes with a limited set of possible values per attribute. The attributes are `Location`, `Hotel Location`, `Room Type`, `Season`, `Duration` and `Transportation`. `Location` denotes the countries that Alice and Bob are considering for their holiday: `France`, `the Netherlands` and `Turkey`. `Hotel Location` can be near to `Sea` or `Historical Place` or `Mountain`. `Room Type` is categorized as `Non-smoking` and `Smoking`. For `Season` there are two values: `Summer` and `Winter`. The values for `Transportation` are `Car` and `Plane`. There are three possible values for `Duration`: `One week`, `Two weeks` and `Three weeks`.

5.1.2. Preference Elicitation

To construct the consumer agent's preference profiles as CP-nets and UCP-nets, we elicited preferences of 10 people that are students and faculty from the Department of Computer Engineering at Bogazici University and Delft University of Technology. We observe that people had difficulty in expressing their preferences as UCP-nets while they were more comfortable giving their preferences as CP-nets. That supports the fact that it is intuitive to use qualitative preference models from the users' point of view.

These 10 CP-nets are completely different from each other. A selection is available in Appendix 8.3. It is important to use different CP-nets in our experiments for the robustness of the results since the structure of the CP-nets (i.e. the number of dependencies and hierarchical levels) may affect the performance of the heuristics.

In order to elicit the user's preferences as an acyclic CP-net, we need to enable the user to specify any preferential dependencies among attributes as well as to rank all

possible attribute values with respect to their parents. To do this, we have developed an add-on tool to GENIUS that first asks the user to select the parents of an attribute from a given list. For instance, a user can express that her preference on hotel location depends on the season. Here, season is the parent of the hotel location. Next, since we do not allow cyclic CP-nets, our tool checks whether the given dependencies are cyclic or not. If so, it enables the user to revise her choices. Finally, after eliciting acyclic preferential dependencies, the user selects the order of attribute values from a given list. After completing the elicitation of the CP-net, this tool generates a preference graph induced from the given CP-net, which is used by the proposed heuristic in order to generate estimated utilities.

Elicitation of the user's preferences as a UCP-net is performed in a similar way. In addition to specifying preferential dependencies, utility values are assigned to all possible attribute values with respect to their parents. This time, the tool verifies whether the given UCP-net is a valid UCP-net satisfying CP-relationships. To do this, it checks whether the condition for being a valid UCP-net (Chapter 2) is satisfied. If so, the elicitation of the user's preferences as a UCP-net is completed successfully. Thus, we ensure that all CP-net and UCP-net pairs used in our experiments are consistent.

Since the producer agent's preferences also affect the negotiation outcome significantly, we generate 10 different preference profiles randomly for the producer agent. For simplicity, these preference profiles are in the form of linear additive utility functions. As a result, 10 different the consumer agents negotiate with 10 different the producer agents. Mentioned before, each agent only knows its own preferences.

5.1.3. Negotiation Settings

In principle, there are many different application domains and many different strategies for negotiation. Here we have chosen a holiday domain, since what is important is not the domain description itself but accommodating various intricate relations among attributes. Even though we have chosen a single domain, we provided variations on the preference structures by creating 10 different preference profiles that correspond to 10 different CP-nets.

We have selected a simple concession-based strategy for proposing bids during a negotiation session. Of course, in a heuristic study such as ours it is not possible to search through the infinite negotiation strategy space. Moreover, negotiation strategies are not the focus of this study and we believe that a concession-based strategy is a useful strategy in many settings [48] as well as ours. Thus, in our experiments the consumer agent uses a simple concession-based strategy explained in the previous chapter (See Section 4.2).

In our negotiation setting, the producer agent uses a simple random-based strategy. In this strategy, the agent randomly generates an outcome whose utility is higher than a predefined threshold value. This is an important reason why we have 10 preference profiles for the producer agent. By repeating the experiments using different preference profiles, we ensure that we report realistic performance results for the consumer agent. In our experiments, we choose the threshold value as 0.6. The threshold is updated when the consumer agent offers an outcome whose utility is higher than the current threshold. The producer agent accepts the consumer agent's counter offer when the utility of the counter offer is higher than or equal to the utility of the producer agent's previous offer. Note that the producer agent may produce the same offer several times because of randomness. This may lead to an increase in the number of rounds to reach a consensus. Thus, we determine a deadline for the producer agent. If the number of rounds reaches 500, the agent accepts the consumer agent's counter offer. In our experiments, very few negotiations (exactly 0.36 %) end due to this time out.

Furthermore, because of generation of offers randomly, we repeat each negotiation session 10 times and report the averages. Note that by using a random strategy we do not introduce a bias towards a particular strategy and therefore explore more of the strategy space. In addition, negotiating with the same producer agent 10 times gives the effect of negotiating with 10 producer agents having the same preferences but adapting different negotiation strategies. Consequently, we observe a variety of possible negotiations in our experiments. Table 5.1 summarizes our experimental setting.

Table 5.1. Experimental setting.

Number of CP-net and UCP-net pairs (number of consumers)	10
Number of linear additive utility functions (number of producers)	10
Number of times that the same consumer negotiates with the same producer	10
Number of negotiations for each consumer's CP-net	100
Number of all negotiations between consumers and producers	1000

5.2. Evaluation

We define four evaluation criteria for comparison: performance, reliability, speed, and relative speed.

5.2.1. Performance

We measure the performance as the average utility of negotiation outcomes for the consumer agent. In our experiments, each consumer agent negotiates 10 times with 10 producer agents that have different preferences. As a result, for each consumer agent there are 100 negotiations with the producer agent. During our experiments each negotiation session is completed successfully; that is, both agents have a consensus in all negotiations. According to this criterion, we evaluate each negotiation outcome with respect to the consumer agent's current preferences represented as a UCP-net so as to observe the performance of heuristics. Thus, the utility of each negotiation outcome is estimated over 100 negotiations and the average of these utilities is reported.

Table 5.2 shows the average utilities of negotiation outcomes for 10 different consumer agents. The first column of the table shows which CP-net is used by the consumer agent and the remaining five columns indicate the average utility of the negotiation outcomes for the consumer agent over 100 negotiations with the producer agent while this agent is applying a particular heuristic (DL, DLH, PRH, POH, TH, AAH). The seventh column shows the average utility for the case when the consumer agent uses random utilities, whereas the last column shows it for the case when the consumer agent has its user's real total preference ordering as a UCP-net. Note that since the performance of

DH and DLH is the same for all negotiations, while we talk about the performance of DH in the remainder of this section, we mean the performance of both heuristics.

Table 5.2. Average utility of negotiation outcomes for the consumer agent.

CP-net	**DH-DLH**	**PRH**	**POH**	**TH**	**AAH**	**Random**	**UCP-net**
1	0.85	**0.88**	0.87	0.82	0.85	0.71	0.91
2	0.97	0.95	0.97	**0.98**	0.97	0.80	0.98
3	**0.97**	0.94	**0.97**	0.96	0.96	0.82	0.97
4	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	0.78	0.96
5	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	0.94	0.99
6	**0.96**	0.95	0.93	**0.96**	0.95	0.81	0.98
7	**0.94**	0.92	**0.94**	0.93	**0.94**	0.81	0.96
8	**0.99**	0.98	**0.99**	**0.99**	**0.99**	0.91	0.99
9	0.98	0.96	0.98	**0.99**	0.98	0.81	0.98
10	**0.98**	0.96	0.96	0.97	0.97	0.82	0.99
Avg:	**0.96**	0.95	0.95	0.95	0.95	0.82	0.97
∗ **Bold** represents the highest average utility for the heuristics							

As expected the consumer agent using UCP-net gets the highest score since it negotiates with its user's real preference orderings. Notice that the performance of all the proposed heuristics is better than the Random approach. When we investigate the performance of the proposed heuristics elaborately, it can be seen that the consumer agent using DH gets the highest average utility in 7 out of 10 CP-nets (See the results for CP-nets: 3, 4, 5, 6, 7, 8, and 10 in Table 5.2) whereas the consumer agent using TH gets the highest average utility in 6 out of 10 CP-nets (See the results for CP-nets: 2, 4, 5, 6, 8, and 9 in Table 5.2). Furthermore, the consumer agent using POH gains the highest average utility in 5 out of 10 CP-nets (See the results for CP-nets: 3, 4, 5, 7 and 8 in Table 5.2). Thus, these three heuristics seem to be good candidates in case we only have a user's CP-net instead of a UCP-net.

Furthermore, the last row of Table 5.2 shows the average utility of negotiation outcomes with respect to the consumer agent's preferences over all negotiations between

the consumer agents and the producer agents. Among all heuristics, the performance of DH is slightly higher than the performance of the other heuristics ($0.96 > 0.95$). We have analyzed these 1000 negotiation results using ANOVA (Analysis of Variance). Since we have two factors: CP-net and Heuristic, we apply ANOVA two-factor analysis. When we analyze the results for DH with those for POH and TH, we see that the mean of the utilities of negotiation outcomes for the consumer agent using DH is statistically different from the means of the utilities for the consumer agent using POH and TH under 95 per cent confidence level where $F > F_{crit}$ and $p_{value} < 0.05$ for both factors. That is, it can be said that the average utility of the consumer agent applying DH is higher than that of the consumer agent applying either POH or TH. Notice that in general the utilities gained are rather high. From our perspective, the absolute values of utilities are not significant, since our aim is to find heuristics that perform close to UCP-nets' utilities. Hence, our aim is to closely estimate the utilities with respect to UCP-net's utilities.

We also perform the worst-case analysis of the proposed heuristics' performance with respect to the consumer agent's utility of the negotiation outcomes. To do this, we evaluate the lowest utility that the consumer agent gets over 100 negotiations for each CP-net and UCP-net pair. Table 5.3 shows the lowest utility that the consumer agent gains out of 100 negotiations.

According to the results in Table 5.3, it is obvious that the highest utility belongs to the consumer agent negotiating with UCP-nets. The second highest utility belongs to the consumer agent applying DL heuristic in 7 out of 10 cases (See the results for CP-nets: 1, 2, 3, 4, 7, 9 and 10 in Table 5.3) where the consumer agent applying TH gets the second highest utility in 6 out of 10 cases (See the results for CP-nets: 2, 3, 5, 6, 7 and 9 in Table 5.3). If we consider all of the negotiations (1000 negotiations), the second highest utility belongs to the consumer agent applying DL. These results also support that the performance of DH is slightly better than other heuristics.

While the average performance of a particular heuristic is important, it is also crucial to find a heuristic that exhibits the same performance consistently. Thus, our second evaluation criterion is about the consistency of the heuristics; which we name reliability.

Table 5.3. The lowest utility that the consumer agent gets over 100 negotiations.

CP-net	DH-DLH	PRH	POH	TH	AAH	Random	UCP-net
1	**0.74**	0.73	0.73	0.68	0.68	0.62	0.84
2	**0.92**	0.85	0.86	**0.92**	0.88	0.77	0.95
3	**0.94**	0.84	**0.94**	**0.94**	0.91	0.77	0.94
4	**0.90**	0.86	0.88	0.84	0.86	0.65	0.90
5	0.94	0.82	0.94	**0.96**	**0.96**	0.73	0.96
6	0.84	0.84	0.84	**0.89**	0.84	0.79	0.93
7	**0.86**	0.78	**0.86**	**0.86**	**0.86**	0.71	0.89
8	0.95	0.90	**0.96**	0.95	**0.96**	0.72	0.98
9	**0.92**	0.85	**0.92**	**0.92**	0.87	0.69	0.92
10	**0.87**	0.83	0.86	0.85	**0.87**	0.74	0.97
Min Lowest:	**0.74**	0.73	0.73	0.68	0.68	0.62	0.84
∗ **Bold** shows the best worst case performance for the heuristics							

5.2.2. Reliability

We measure the reliability as the number of times that the consumer agent using a heuristic negotiates at least as well as the consumer agent having a UCP-net. For each CP-net and UCP-net pair, we compare the utility gained by the consumer agent using a heuristic with the utility gained by the consumer agent having a UCP-net. If the utility gained by the agent using a heuristic is higher than or equal to the utility gained by the agent having a UCP-net, that agent receives one point. Since 10 different producer agents negotiate with the same consumer agent 10 times, we count the number of times that the consumer agent using a heuristic is at least as successful as the consumer agent having a UCP-net over 100 negotiations. Recall that we have 10 different CP-net and UCP-net pairs for the consumer agent in our experiments. When we consider all negotiations, we have 1000 negotiations between the consumer and the producer agents.

Table 5.4 shows the evaluation of this criterion for each CP-net over 100 negotiations. The last row of the table shows the number of times that the agent applying a heuristic on 10 CP-nets and negotiates as well as the agent having total preference or-

dering (UCP-net) over 1000 negotiations. When the consumer agent uses a CP-net and applies DH, it negotiates at least as well as the agent using a UCP-Net in 57.8 per cent of negotiations and *Taxonomic Heuristic* (TH) is successful at least as UCP-Net in 57.5 per cent of negotiations. Also, the performance of *Preferred Outcomes Heuristic* (POH) is close to DH and TH (57.2). Although the number of times DH performs as well as UCP-net seems to be the highest for only the tenth CP-net, it is seen that DH has the the highest score (578) while considering all negotiation results (1000 negotiations). This stems from the fact that the variance of DH is less than the other heuristics. In this respect, we can identify DH as the most reliable heuristic.

Table 5.4. Number of times heuristics performs as well as UCP-nets (out of 100).

CP-net	**DH-DLH**	**PRH**	**POH**	**TH**	**AAH**	**Random**
1	34	44	**47**	25	36	0
2	44	46	50	**68**	61	0
3	53	38	**59**	41	45	0
4	53	50	44	**60**	58	1
5	64	**86**	67	53	52	22
6	48	**49**	32	48	45	0
7	57	37	**59**	43	45	0
8	89	78	92	**93**	89	4
9	74	63	75	**91**	85	2
10	**62**	48	47	53	49	0
Sum	**578**	539	**572**	**575**	565	29
Percentage	**57.8 %**	53.9 %	**57.2 %**	**57.5 %**	56.5 %	2.9 %
∗ **Bold** shows the highest scores for the heuristics						

As far as the performance of the heuristics with respect to the first two evaluation criteria is concerned, DH, TH and POH might be considered as alternative heuristics that the user may use in negotiation when it has a CP-net. The performance of DH is slightly better than others so it may be preferred over TH and POH. In addition to negotiating effectively, it is also important to reach an agreement early. Therefore, our next evaluation criterion is how fast the consumer agent negotiates.

5.2.3. Duration

We measure the duration as the average number of rounds for reaching a consensus. We evaluate each approach with respect to their duration for a successful negotiation. For this purpose, we report the average number of rounds that is needed to reach a consensus over 100 negotiations. We consider a round as the interaction of an agent with another agent by means of sending an offer — the number of offers that are made by agents. Thus, it is desired to negotiate with fewer rounds.

Table 5.5 shows the average number of rounds to reach a consensus over 100 negotiations. Note that we do not investigate the average number of rounds for Random since its performance for our first two criteria is insufficient. On average, the agent using DH negotiates at 24 rounds while the agent using POH negotiates at 33 rounds. When we apply ANOVA two-factor analysis, it can be said that the mean of DH is statistically different from the mean of POH under 95 per cent confidence level where $F > F_{crit}$ and $p_{value} < 0.05$ for both factors. This means that DH negotiates faster than POH. The average number of rounds that is needed to reach a consensus for PRH, TH, AAH and UCP-net is relatively higher (respectively 39, 40, 38, 43 on average). Under 95 per cent confidence level, the mean of DH is also statistically different from the mean of UCP-net. As a result, it can be said that the consumer agent having a CP-net and applying DH negotiates faster than the consumer agent having a UCP-net and negotiating with the user's real total preference ordering. Further, observe that the agent using DH heuristic reaches agreements faster than any other approach.

We also investigate the worst-case analysis of the consumer agent's performance with respect to the number of rounds that are required to complete a negotiation session successfully. Table 5.6 shows the highest number of rounds to reach a consensus over 100 negotiations for each CP-net and UCP-net pair. According to the results, the consumer agent applying DH negotiates faster than others in 7 out of 10 cases (See the results for CP-nets: 1, 2, 3, 5, 7, 8 and 10 in Table 5.6) where others negotiate faster individually in only 1 out of 10 cases. Overall, it can be said that DH's worst-case performance is better than others. Since we mainly concentrate on comparing the agent negotiating with a CP-net applying a heuristic with the agent negotiating with UCP-net, we also

Table 5.5. Average number of rounds to reach a consensus.

CP-net	DH-DLH	PRH	POH	TH	AAH	UCP-net
1	**30.07**	65.02	39.78	55.62	49.99	68.96
2	**33.23**	65.75	44.43	78.68	73.91	56.63
3	**24.18**	38.90	37.28	39.81	50.06	52.06
4	27.97	32.12	29.87	37.99	**27.27**	68.36
5	**8.48**	21.97	10.10	17.25	20.13	21.30
6	22.43	24.82	27.28	25.63	**21.13**	35.59
7	**22.82**	31.28	31.58	36.34	37.79	29.33
8	**11.4**	14.94	14.77	16.47	15.66	14.47
9	39.88	43.96	43.74	48.14	**37.31**	47.21
10	**21.88**	53.68	49.29	45.25	44.22	38.09
AVG:	**24.23**	39.24	**32.81**	40.12	37.75	43.20
∗ **Bold** shows the lowest average number of rounds for a given CP-net						

Table 5.6. The highest number of rounds to reach a consensus (out of 100).

CP-net	DH-DLH	PRH	POH	TH	AAH	UCP-net
1	**131**	500	215	500	500	500
2	**141**	500	500	500	500	419
3	**255**	431	363	500	485	500
4	235	**202**	373	500	285	500
5	**129**	277	**129**	277	339	339
6	419	197	419	225	**195**	500
7	**253**	387	387	413	465	**253**
8	**37**	113	139	87	81	97
9	493	313	493	500	**236**	485
10	**107**	500	487	500	500	500
∗ **Bold** shows the best worst case performance for a given CP-net						

investigate the relative speed of the heuristics to UCP-net in negotiation.

5.2.4. Relative Speed

We measure the relative speed as the number of times that the consumer agent applying a heuristic negotiates at least as fast as the consumer agent having a UCP-net. If the consumer agent applying a particular heuristic negotiates faster or just as fast as a UCP-net, the consumer agent applying that heuristic receives one point. For each CP-net and UCP-net pair, we evaluate 100 negotiations and compare the scores that the heuristics get at the end.

Table 5.7 shows the results for this criterion. According to this result, it is seen that DH and POH outperforms other heuristics in terms of this criterion. In 68.1 per cent of negotiations the agents applying DH negotiates at least as fast as the agents using UCP-nets while the agents applying POH negotiates at least as fast as the agents using UCP-nets in 67.5 per cent of negotiations. On the other hand, the consumer agents using other heuristics negotiates at least as fast as the agent using UCP-nets in about 63 per cent of negotiations. This result is a confirmation of our previous result shown in Tables 5.5 and 5.6.

As a result, the agent may prefer to apply *Depth Heuristic* or *Preferred Outcome Heuristic* in terms of success and speed when it has a CP-net. Since the performance of DH is slightly better than POH, DH may be preferred over POH. With respect to the speed, DH has a clear advantage compared to all other approaches. Moreover, when we consider the implementation details and computational complexity of the proposed heuristics, DH can be identified as the leading heuristic. In *Depth Heuristic* we consider only the depth of an outcome and the depth of preference graph while in *Preferred Outcome Heuristic* we take into consideration all of the outcomes which are preferred over the current outcome (requiring search through the graph) and depth of those outcomes. The former heuristic is simpler. Concluding, a CP-net with DH heuristic enables us to negotiate almost as well as a UCP-net while providing faster results. This is a clear indication that one can use qualitative preferences in negotiation successfully.

Table 5.7. Number of times that the heuristic negotiates at least as fast as UCP-net.

CP-net	**DH-DLH**	**PRH**	**POH**	**TH**	**AAH**
1	67	57	61	66	60
2	48	44	51	40	45
3	64	64	62	70	63
4	72	73	78	76	75
5	89	51	88	79	82
6	69	80	76	76	77
7	59	69	55	59	55
8	70	74	71	45	52
9	70	47	68	57	60
10	73	68	65	61	57
Sum:	**681**	627	**675**	629	626
Percentage:	**68.1 %**	62.7%	**67.5 %**	62.9%	62.6 %

6. PREFERENCE PREDICTION FOR NEGOTIATION

One of the crucial challenges of the automated service negotiation is the automatic generation of the service offers by the agents. In general, the negotiating agents consider their own user's preferences while generating their offers. However, finding mutually acceptable offers may require not only considering their own preferences but also understanding other negotiating parties' needs. Thus, a negotiating agent should take into account other agent's preferences as well as its own user's preferences in order to generate well-targeted offers that will fulfill both agent's needs. This may also lead the negotiating parties to have a consensus more quickly. For instance, consider a producer agent having too many services that it can offer. Proposing each service offer arbitrarily one by one will be time consuming for both negotiating parties. It would be much useful if the producer can understand the consumer's needs and propose offers that respect both its and the consumer's preferences. This is also beneficial if the producer cannot fulfill the consumer's request. Rather than proposing all possible offers and failing after the last one, if the producer considers the consumer's needs, it can decide that the consumer's needs cannot be satisfied early on.

Nevertheless, as we have described in our negotiation architecture (see Chapter 3) each negotiating agent has right to access only its own user's preferences. Thus, neither the producer agent nor the consumer agent knows each other's preferences. The immediate question is how an agent, say the producer agent generates mutually acceptable offers meeting both its and the consumer's needs without knowing the consumer's preferences. Our intuition for tackling this problem is to enable the producer to understand the consumer's preferences from the bids exchanged during the negotiation: the consumer's offers and the producer's counter-offers that are rejected by the consumer.

In this section, we study how the producer agent predicts the consumer's preferences and revises its offers by means of these predicted preferences. The producer agent needs to apply a preference prediction algorithm that is based on concept learning. There are a number of important points to be taken into consideration to find an appropriate algorithm. These are:

- The model to be learnt: The producer agent does not know how the consumer represents its preferences and the consumer is free to use any preference representation. The consumer may represent its preferences with conjunctive and disjunctive constraints or may have a CP-net or even a utility function. Should the algorithm model a linear additive utility function or an ordering induced from a CP-net? What happens if the consumer represents its preferences with constraints and the producer agent tries to model a linear additive utility function? Since the individual models are not known, it would be better if the producer tries to find a generic model independent from the consumer's preference representation meanwhile providing useful information for successful negotiations. This generic model can classify the outcomes as acceptable or unacceptable with respect to the consumer. Consequently, the producer agent can revise its offers so that it does not propose a service offer that would be possibly rejected by the consumer.
- Training instances: At the beginning of the negotiation, the producer agent does not have any information related to the consumer agent. When the producer agent starts negotiating, it can observe the consumer agent's offers and the consumer's response towards the producer's counter-offer. Thus, the bids exchanged between the consumer and producer may constitute the training set of the producer's preference prediction algorithm. When the consumer requests a service offer, the producer agent can interpret this offer as a positive training instance since we believe that the consumer's offer reflects its preferences. If the consumer's offer were not consistent with the consumer's preferences, the consumer would not have requested it in the first place. When the producer makes a counter-offer and the consumer rejects it, the producer can interpret that counter-offer as a negative training instance. If it were acceptable at that moment, then the consumer would have taken that offer. After each offer and counter-offer, the producer needs to update the learnt concept. As a result, the producer's learning algorithm should be incremental.
- Retractability: The prediction algorithm should be able to retract its findings as more bids are exchanged. This is integral to the incremental natural of negotiation. As more bids become available, previous models of the negotiating agent may no longer be accurate. Hence, the algorithm should adapt to this.
- Domain Knowledge: While predicting the consumer's preferences, the producer agent can take an advantage of using additional domain knowledge. Incorporating

ontology reasoning will provide more powerful predictions. By means of ontology, the producer can identify similar services and treat them similarly. Thus, enhancing with the use of ontology is a desired property of an appropriate prediction algorithm for negotiation.

- The objective of the learning: It is quite difficult to learn the exact preferences in a few interactions during the negotiation. In addition, the producer does not know the consumer's preference representation. Thus, the aim of the producer is not to learn the consumer's exact preferences. Instead, it tries to learn an approximate model that guides the producer to create well-targeted offers even if the consumer's preferences are specified in a complex way. Moreover, this approach can help the producer detect early whether some preferences cannot be satisfied and thus consensus cannot be reached.

As far as the above statements are concerned, inductive learning approaches such as version spaces and decision trees can be adopted for this purpose. For example, Candidate Elimination Algorithm (CEA) [49], which is a concept learning algorithm based on building and maintaining version spaces, might be used to learn the concept of acceptable offers by the consumer. Alternatively, a well-known decision tree algorithm, ID3 [50] might be used to classify the service offers as acceptable and unacceptable with respect to the consumer. CEA, by design, is incremental, whereas ID3 needs to be modified to make it incremental. Hence, we take CEA as our starting point.

While ID3 supports learning disjunctives, CEA does not. One of the important matters is supporting to model a disjunctive concept, which consists of a number of individual concepts describing acceptable outcomes. In some cases, a single concept may become insufficient to capture all acceptable offers. Both constraints and CP-nets explained in Section 2 allow representing disjunctive concepts. For instance, a consumer may have a tendency to accept a red and dry wine. Alternatively, a rose and sweet wine may be acceptable for her. Thus, we can capture these with the following disjunctive concept: [(Color=`Red` $\wedge$ Sugar level=`Dry`) $\vee$ (Color=`Rose` $\wedge$ Sugar level=`Sweet`)]. That is, we need to support to model a disjunctive concept, which provides more powerful and realistic expressiveness than a single concept.

Moreover, neither CEA nor ID3 utilizes the domain knowledge. To improve the negotiation process, we need a prediction approach that both supports to model disjunctives and uses the domain knowledge to enable the producer agent to reason on the attribute values. Consequently, the producer identifies the similar offers and treats them similarly. We combine these ideas in an extension of CEA, called Revisable Candidate Elimination Algorithm (RCEA). RCEA is incremental and inductive as CEA, but it also supports learning disjunctive concepts as ID3, can utilize an ontology, and can retract its hypothesis about what it has learned as more interactions take place.

The producer uses RCEA to predict the consumer's preferences and to generate offers that respect them. In other words, instead of blindly searching for agreements in the search space, the producer will do an informed search. The aim of using RCEA is three fold. First, if consensus is possible, we want the agents to find the mutually agreeable service. That is, we want to enable successful negotiations. Second, we want the agents to reach this consensus in as few steps as possible. If a producer offers a possible service after too many interactions, the consumer would walk away. Hence, the number of interactions to reach a consensus is important. Third, if consensus is not possible, we want to detect this and terminate the negotiation as early as possible. If the producer does not realize that it would not be able to satisfy the consumer's needs in a reasonable time, it would waste the consumer's time.

We present a technical background and then explain RCEA elaborately in the following sections.

6.1. Technical Background

In this section we give a brief introduction to several learning algorithms and classifiers such as CEA, DEA, ID3 and naive Bayes' classifier and describe our use of ontologies.

6.1.1. Preliminaries

In our negotiation setting, each service offer is represented as a vector of attribute values. For example, consider the wine domain explained in Chapter 2. The following

offer, <`French`, `Red`, `Dry`> represents a red and dry wine produced in a French region. Both the consumer and producer agent represents their offers in this way.

When the producer tries to understand the consumer's preferences, consumer's each offer is accepted as a positive example where producer's each counter-offer that is rejected by the consumer is taken as a negative example. The learned service descriptions (target concept) consist of hypotheses. Similar to service offers, each hypothesis is represented as a set of attribute values but also including a special value, "?". This special value means that any value is acceptable. For instance, a possible hypothesis can be <?, ?, `Dry`> that covers any services whose sugar level is dry. For example, the wine service <`Italian`, `Rose`, `Dry`> is covered by this hypothesis.

The learned hypotheses are used to decide whether a given service offer will possibly be rejected by the consumer or it might be acceptable with respect to consumer's preferences. During the negotiation, the producer agent filters out its available services that is likely to be rejected by the consumer based on the learned hypothesis. Among the remaining services, it offers the most convenient service.

6.1.2. Candidate Elimination Algorithm (CEA)

CEA is an inductive learning algorithm that is based on version spaces in which a target concept is learned from the observed examples [49]. In a version space [51], there are two significant hypothesis sets: the most general (G) and the most specific (S). G includes the general hypotheses whose boundary is as large as possible whereas the hypotheses in S are as specific as possible so that they minimally cover the positive samples.

At the beginning, G contains the most general hypothesis covering all possible services, < ?, ?, ?> and S contains the description of the first positive sample. When a negative training sample comes, the hypotheses in G are specialized not to cover this sample any more. To illustrate this, the consumer first offers <`Chianti`, `Rose`, `OffDry`>. Thus, S includes this offer. Assume that when the producer offers <`Chianti`, `Rose`, `Sweet`>, the consumer rejects this service because it is not an acceptable service with

respect to consumer's preferences. We take this rejected service offer as a negative example. Since the current G covers this hypothesis, it should be specialized in a minimal way not to cover that sample. To do it, we use the hypothesis in S. We compare the values of the attributes in the negative example with those in the specific hypothesis. When the values are different, we use these values to specialize the current general hypothesis. In this example, only the value of the sugar level is different for them. Thus, G becomes {<?, ?, `OffDry`>}, meaning that only offdry wines are acceptable. Since S does not cover this example, it remains stable. If any hypothesis in S covers the negative sample, that hypothesis is removed from S.

When a positive example arrives, the hypotheses in S are generalized to cover this sample. For instance, if S contains {<`French`, `Red`, `Sweet`>} and the current positive example (the consumer's request) is equal to <`French`, `Rose`, `OffDry`>, S becomes {<`French`, ?, ?>} in order to cover the current positive example. And, the hypotheses in G, which do not cover this positive sample are removed from G. This rule does not allow CEA to learn disjunctive concepts because all general hypotheses cannot be enforced to cover each positive sample when the target concept is disjunctive. Table 6.1 illustrates how CEA fails in the case of this generalization process. Following this scenario, when the consumer requests <`Italian`, `White`, `OffDry`>, none of the hypotheses in G covers this positive sample. According to CEA, they should be removed from G. When we remove these hypothesis, G becomes empty.

Table 6.1. When Candidate Elimination Algorithm fails.

Type	**Sample**	**The most general set**	**The most specific set**
+	(French,Red,Dry)	{(?, ?, ?)}	{(French,Red,Dry)}
-	(French,Rose,Sweet)	{(?,Red,?), (?,?,Dry)}	{(French,Red,Dry)}
+	(Italian,White,OffDry)	{ }	{(?, ?, ?)}

In short, when a positive sample comes, the hypotheses in S are generalized if they do not cover this positive sample. When a negative sample arrives, the hypotheses in G covering this negative sample are specialized in a minimal fashion. Consequently, the hypotheses in G become more specific where those in S become more general over time.

Eventually, G and S intersect when the target concept is learned.

6.1.3. Disjunctive Candidate Elimination Algorithm (DCEA)

CEA does not support learning disjunctives. DCEA [52] improves CEA to handle disjunctives by extending the hypothesis language to include disjunctive hypothesis in addition to the conjunctives. Each attribute of the hypothesis has two parts: inclusive list, which holds the list of valid values for that attribute and exclusive list, which is the list of values which cannot be used for that attribute.

Assume that the most specific set has a single hypothesis as {<`California`, `Red`, `Sweet`>} and a positive example <`California`, `Rose`, `Sweet`> comes. The original CEA will generalize this as <`California`, ?, `Sweet`>, meaning the color can take any value. However, in fact, we only know that the color can be `Red` or `Rose`. In the DCEA, we generalize the hypothesis as {<`California`, [`Red`, `Rose`], `Sweet`>}. Only when all the values exist in the list, they will be replaced by *?*. In other words, the algorithm generalizes specific hypotheses more slowly than before.

When a positive example comes, DCEA does not eliminate the general hypotheses in G not covering the sample anymore in order to support learning disjunctive concepts. Here, the intuition is that a general hypothesis does not have to cover all positive examples. This positive sample is added as a separate hypothesis into S unless there exists any hypothesis in S that can be merged with this sample. Otherwise, the algorithm combines this sample with that specific hypothesis.

When a negative sample comes, for each hypothesis in G that covers this negative sample, new hypotheses are generated by excluding each attribute value of the negative sample from the original hypothesis. For instance, assume that the negative example is <`Chianti`, `Rose`, `Sweet`> and there exists a general hypothesis in G such as {<?, `ReddishColor`, ? >}. Note that `ReddishColor` is a parent concept of `Rose` and `Red`. DCEA is able to use the hierarchical information about attribute values in generalization process. After the negative example, this hypothesis will be eliminated and three new hypotheses will be added. These hypotheses are {<?-`Chianti`, `ReddishColor`, ?>, <?,

`ReddishColor-Rose`, ?>, <?, `ReddishColor`, ?-`Sweet`>}. Consequently, all possible general hypotheses are generated. While the process is more accurate than CEA, it is also substantially more expensive in terms of generated hypothesis.

6.1.4. ID3 Decision Tree Algorithm

ID3 is an inductive learning algorithm used in constructing decision trees in a top-down fashion from the observed examples represented in a vector with attribute-value pairs [50]. Unlike CEA, this algorithm supports learning a disjunctive concept.

A decision tree has two types of nodes: leaf node in which the class labels of the instances are kept and non-leaf nodes in which the test attributes are held. The test attribute in a non-leaf node is one of the attributes making up the concept description. The selection of test attributes is a crucial task in construction phase of the tree since it affects the size of tree. Smaller decision trees are preferable since it makes decision quicker. These test attributes are used to divide the examples into subsets by considering the (im)purity of examples in each group. ID3 uses information gain [53], which is a popular criterion for impurity test.

After selection of a test attribute, tree splits in accordance with all possible values of that attribute. For instance, if the test attribute is `Color` in our wine example, the node will be branched into three parts for the values: `Red`, `Rose`, and `White`. This operation will be performed for all new nodes recursively. The splitting will be finished when each subset is homogeneous; i.e., have the same class label or have no attribute left for testing.

The problem with this algorithm is that it is not an incremental algorithm, which means all the training examples should exist before learning. To overcome this problem, the system can keep the training instances at each time. After each new coming instances, the decision tree can be rebuilt. Without doubt, there is a drawback of reconstruction such as additional process load. Nevertheless, this process load is not significant for our purposes since ID3 algorithm runs fast.

Table 6.2 shows an example negotiation scenario between agents. After the consumer's third request, the producer agent constructs the decision tree depicted in Figure 6.1. The producer starts searching from the root to the leaf nodes in order to classify its available services. For example, according to the constructed tree in Figure 6.1, <`US`, `Red`, `Sweet`) is classified as negative where <`California`, `Red`, `OffDry`) is classified as positive. In some cases, the values of a service may not be represented via branches. For instance, consider <`French`, `Rose`, `Dry`). For the region attribute, we do not have any branch for `French`. In such a case, ID3 algorithm classifies the service as unknown. Note that during the negotiation the producer only filters the available services, which are classified as negative because they would be possibly rejected by the consumer.

Table 6.2. Example training data obtained during negotiation.

Request or offer	**Region**	**Color**	**Sugar**
First request (+):	Chianti	Rose	OffDry
First counter offer (-):	Chianti	Rose	Sweet
Second request (+):	California	White	Sweet
Second counter offer (-):	US	Rose	OffDry
Third request (+):	Italian	Red	Dry

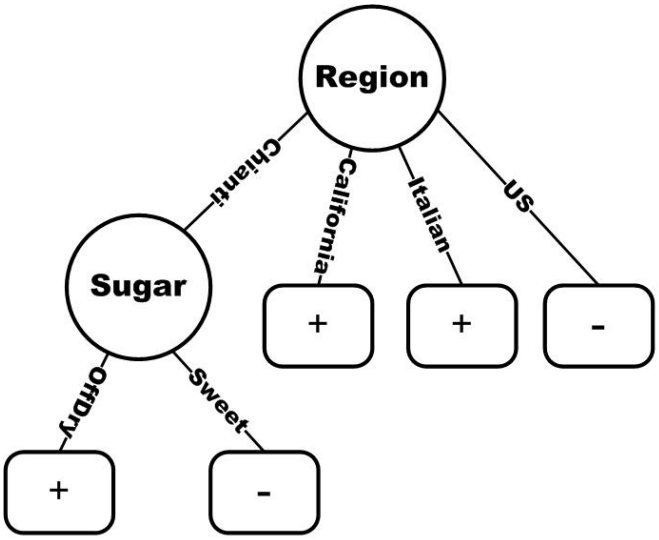


Figure 6.1. Sample decision tree.

6.1.5. Bayes' Classifier

In order to make decision under uncertainty, a classification based on probabilities can be applied [54]. In this classification, we estimate the probabilities of the classes with Bayes' rule (Equation 6.1). In this equation, $P(C)$ denotes the prior probability of an issue belonging to class C where $p(x|C)$ is the likelihood of observing x when the given issue belongs to C. Note that $p(x)$ is the evidence that is the probability of observing x where $P(C|x)$ is the posterior probability—the probability of the given issue, x belonging to C.

$$P(C|x) = \frac{P(C)p(x|C)}{p(x)} \tag{6.1}$$

In our negotiation domain, we have two classes: positive (+) and negative (−). As is customary, we assume that all issues are independent while we use Bayes' classifier [23]. Since an offer consists of several issues $(x_1, x_2, ...x_n)$, the probability of an offer belonging to a particular class (+ or −) is equal to $\prod_{i \in n} P(C|x_i)$. Instead of multiplying the posterior probabilities, we can take the logarithm of both sides. As a result, the posterior probability of an offer belonging to the given class is equal to $\sum_{i=1}^{n} \log P(C|x_i)$.

To estimate the individual posterior probabilities, the producer keeps all consumer's requests (positives) and its offers rejected by the consumer (negatives) during the negotiation. It is easy to estimate the prior probabilities because it is equal to the ratio of the number of examples belonging to that class to the number of the entire examples. For example, if we have three positive examples and two negative examples, $P(C = +)$ is equal to 3/5 and $P(C = -)$ is equal to 2/5. Since all negotiation issues are discrete, the likelihood is the ratio of how many times x belonging to C is observed to the number of examples belonging to C. To illustrate this, consider the training examples in Table 6.2. According to this table, $P(x_1|C = +)$ where x_1 has the value of `Chianti` is equal to 1/3 because we have three positive examples and only one of them has the value `Chianti`. Finally, $p(x)$ is equal to $P(C = +)p(x|C = +) + P(C = -)p(x|C = -)$.

According to this classifier, the producer estimates posterior probabilities for both negative and positive classes. If the posterior of the negative class is higher than that of the positive class, this service is classified as negative. Note that the producer may have some services whose values have not been seen yet. For example, none of the examples contains `French` as a region. In this case, posterior probabilities for both class are ignored for that issue. As a result, if our service is <`French`, `Rose`, `OffDry`>, we only consider the posteriors for color (`Rose`) and sugar level (`OffDry`).

6.1.6. Ontology and Semantic Similarity

An ontology represents knowledge of a given domain [55, 56]. Shortly, an ontology contains the specification of concepts and their meanings. We describe the concepts, specify their properties and establish some relationships among them by considering a domain of knowledge or interest. It can be thought as representation of knowledge with its semantics. For instance, consider the wine domain[1] . The ontology for this domain contains the description of a wine concept including its properties such as color, body, winery and so on. In addition to these properties, the ontology includes some relationships such as *HasA* and *isA*. For example, `Chardonnay` is a `Wine` and `Wine` has `Color` where `Color` may be one of `Red`, `Rose` or `White`.

Relationships such as *hasA* and *isA* contribute to reasoning of agents. We can represent a taxonomy by using the *isA* relation. By using these relationships, agents can discover new knowledge from the existing knowledge. For instance, using the wine ontology the following reasoning can be made: `Bordeaux` is defined as a `Wine`, `Medoc` is defined as a `Bordeaux` and `Pauillac` is defined as a `Medoc`. When an agent wants to buy wine, another agent can offer any instance of `Bordeaux`, `Medoc` and `Pauillac` since it can reason that if `Medoc` is a `Bordeaux` and `Bordeaux` is a `Wine` then `Medoc` is a `Wine` and then if `Pauillac` is a `Medoc` and `Medoc` is a `Wine` then `Pauillac` is a wine.

During negotiation, an agent will need to compute similarities between service descriptions. To do this, it will need to compare similarities between values of an attribute. We establish this through a similarity metric. Any similarity metric would be sufficient

[1] http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf

for our architecture. However, our previous comparison of similarity metrics has shown that RP Semantic Similarity Metric works well with ontologies [52] and thus is used in this work. Based on the relative distance between two concepts in a taxonomy, RP metric measures how similar two concepts are. To do this, it exploits the following intuitions. Note that we use Figure 6.2 to illustrate these intuitions.

- Parent versus grandparent: Parent of a node is more similar to the node than grandparents of that node. Generalization of a concept results in going further away from that concept. The more general concepts are, the less similar they are. For example, `AnyWineColor` is parent of `ReddishColor` and `ReddishColor` is parent of `Red`. Then, we expect the similarity between `ReddishColor` and `Red` to be higher than that of the similarity between `AnyWineColor` and `Red`.
- Parent versus sibling: A node would have higher similarity to its parent than to its sibling. For instance, `Red` and `Rose` are children of `ReddishColor`. In this case, we expect the similarity between `Red` and `ReddishColor` to be higher than that of `Red` and `Rose`.
- Sibling versus grandparent: A node is more similar to its sibling rather than to its grandparent. To illustrate, `AnyWineColor` is grandparent of `Red`, and `Red` and `Rose` are siblings. Therefore, we anticipate that `Red` and `Rose` are more similar than `AnyWineColor` and `Red`.

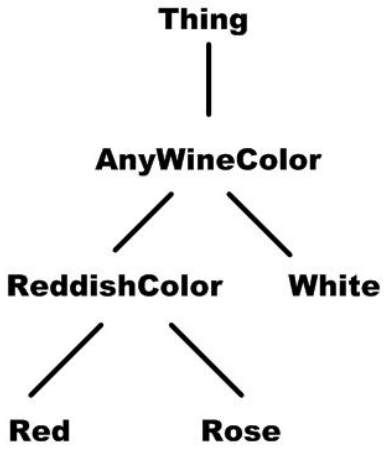


Figure 6.2. Sample taxonomy for similarity estimation.

The relative distance between nodes c_1 and c_2 is estimated in the following way. Starting from c_1, the tree is traversed to reach c_2. At each hop, the similarity decreases since the concepts are getting farther away from each other. However, based on our intuitions, not all hops decrease the similarity equally.

Let m represent the factor for hopping from a child to a parent and n represent the factor for hopping from a sibling to another sibling. Since hopping from a node to its grandparent counts as two parent hops, the discount factor of moving from a node to its grandparent is m^2. According to the above intuitions, our constants should be in the form $m > n > m^2$ where the value of m and n should be between zero and one.

Some similarity estimations related to the taxonomy in Figure 6.2 are given in Table 6.3. In this example, m is taken as $2/3$ and n is taken as $4/7$.

Table 6.3. Sample similarity estimation over sample taxonomy.

$Similarity(ReddishColor, Rose) = 1 * (2/3) = 0.6666667$
$Similarity(Red, Rose) = 1 * (4/7) = 0.5714286$
$Similarity(AnyWineColor, Rose) = 1 * (2/3)^2 = 0.44444445$
$Similarity(White, Rose) = 1 * (2/3) * (4/7) = 0.3809524$

For RP semantic similarity metric, the taxonomy for the attributes is held in the shared ontology in our architecture. In order to evaluate the similarity of the attribute vector, we firstly estimate the similarity for each attribute one by one and take the average sum of these similarities. Since we expect each attribute to make an equal contribution for estimating similarity, the average sum is used to calculate the similarity of vectors.

6.2. Revisable Candidate Elimination Algorithm (RCEA)

To model the consumer's needs, we have developed Revisable Candidate Elimination Algorithm (RCEA), which is an incremental algorithm á lá CEA in which the training samples become available only during the execution. A training sample x corresponds to a service request or a service offer and is a vector of attribute values such as $x =$

$\{x[1], x[2], ..., x[m]\}$ where m is the number of attributes and $x[i]$ is the value of i^{th} attribute. Possible domain values for each attribute are known and attributes can only be assigned values from their respective domain.

The consumer's offers are accepted as positive examples whereas the producer's counter-offers that are rejected by the consumer are taken as negative examples. The concept to be learned should cover the positive samples but not cover the negatives. Using the learning algorithm, the producer decides which of its services would be more desirable for the consumer.

The main improvement to the original algorithm is that RCEA can *retract* its hypothesis about what it has learned as more interactions take place. Further, it uses an underlying ontology of service attributes for revising hypothesis as necessary. Since CEA does not support learning disjunctives or making use of ontologies, we make the following changes to the algorithm.

First, according to CEA, all of the hypotheses in the most general set should cover the entire positive sample set. This rule prevents learning disjunctives since disjunctives are the union of more than one hypotheses and it cannot be covered by a single hypothesis with the condition of excluding negative samples. Therefore, in our learning algorithm we change this rule with that a positive sample should be covered by at least one of the hypotheses in the most general set. Furthermore, in some cases, a revision may be required when there is no more hypothesis in the most general set that is consistent with the incoming positive sample. In such a case, we need to add new hypotheses covering this new positive sample while excluding all the negative samples. Hence, we require to keep the history of negative samples.

Second, when a positive sample comes, generalization of the specific set is performed in a controlled way with a threshold value, Θ. As far as disjunctive concepts are concerned, there should be more than one specific hypothesis in the most specific set. Deciding which hypothesis will be generalized is a complicated task. To do this, we estimate similarity of the current positive sample with respect to each hypothesis in the most specific set. The process of choosing the hypothesis that will be generalized uses this sim-

ilarity information. Here, any similarity metric can be applied. During our study, we use the RP similarity (Section 6.1.6), which uses semantic information such as subsumption relations. At the end, the algorithm only generalizes the selected hypothesis.

Third, different from CEA, the generalization of a hypothesis is now controlled by another threshold value, Φ. This threshold value determines whether a generalization will be performed for each attribute. Generalization of hypothesis is different for each attribute in the hypothesis, in fact it depends on the property of the attributes. If there is an ontological information such that a hierarchy exists on the values of the attribute, the generalization depends on the ratio of the covered branches in the hierarchical tree. Otherwise, we apply a heuristic that favors generalization to *Thing* (?). We explain this further in Section 6.2.2.

6.2.1. Components of RCEA

Revised Candidate Elimination Algorithm (RCEA) manipulates four important sets:

- Most specific set (S) contains the most specific hypotheses that cover the positive examples minimally. Formally, $S = \{H_1^S, H_2^S, ..., H_n^S\}$ where n is the number of specific hypotheses in S and each specific hypothesis is in the form of $H_j^S = \{H_j^S[1], H_j^S[2], ..., H_j^S[m]\}$ where m is the number of attributes and $H_j^S[y]$ is a vector of acceptable values for the y^{th} attribute.
- Most general set (G) contains the most general hypotheses consistent with the positive samples. Formally, $G = \{H_1^G, H_2^G, ..., H_k^G\}$ where each hypothesis is of the form $H_j^G = \{H_j^G[1], H_j^G[2], ..., H_j^G[m]\}$ where m is the number of attributes and $H_j^G[y]$ is a value for the y^{th} attribute.
- Negative Sample Set (N) contains the negative examples.
- Positive Sample Set (P) contains the positive examples.

These sets are important in generating offers to the consumer. If a service is not in G, it means that the service cannot be acceptable for the consumer, since G holds the most general hypotheses about the consumer's preferences. However, many services may

fall in G, then the question is which one to offer. Among possibly acceptable services, finding the most satisfactory services for the consumer is crucial. By finding the most similar services to the most specific set (S) involving minimally acceptable services, the producer can find the most satisfactory services. Note that, here, we are not interested in the convergence of these sets. They are used independently: G for filtering unacceptable services and S for generating the best offer among the alternatives.

6.2.2. Properties of Attributes

Attributes represent the components constructing the service in negotiation. Each attribute is defined in an ontology in which the domain information for these attributes and some relations associated with these attributes such as a hierarchy are kept. Reasoning on these relations would be useful in generalization and specialization of the hypotheses.

For a hypothesis to cover the sample, all attribute values in the hypothesis should be consistent with those of the sample. Consistency can be decided by subsumption relation. For each attribute x and y, $doesCover(x, y)$ means that the concept of x is an ancestor of y or $x = y$ in the case that the attribute has a hierarchy. Otherwise, $doesCover(x, y)$ means that the $x =?$ or $x = y$. Note that ? is a special value for an attribute that means any value is acceptable and it is at the top of the hierarchy.

In some cases, the generalization of a specific hypothesis is required and performed for each attribute of the hypothesis. The generalization of attributes depends on the ontological information. Some attributes have a hierarchical classification whereas some do not. According to whether the attribute has hierarchical information or not, a specific attribute is generalized.

If an attribute has a hierarchical structure such as `WineRegion` as shown in Figure 6.3, the algorithm generalizes the attribute value to the nearest common parent of the values for that attribute under a special condition depending on a threshold value. Here, the proportion of the number of values that the hypothesis involves to the number of values that the nearest common parent concept covers is estimated. For example, if

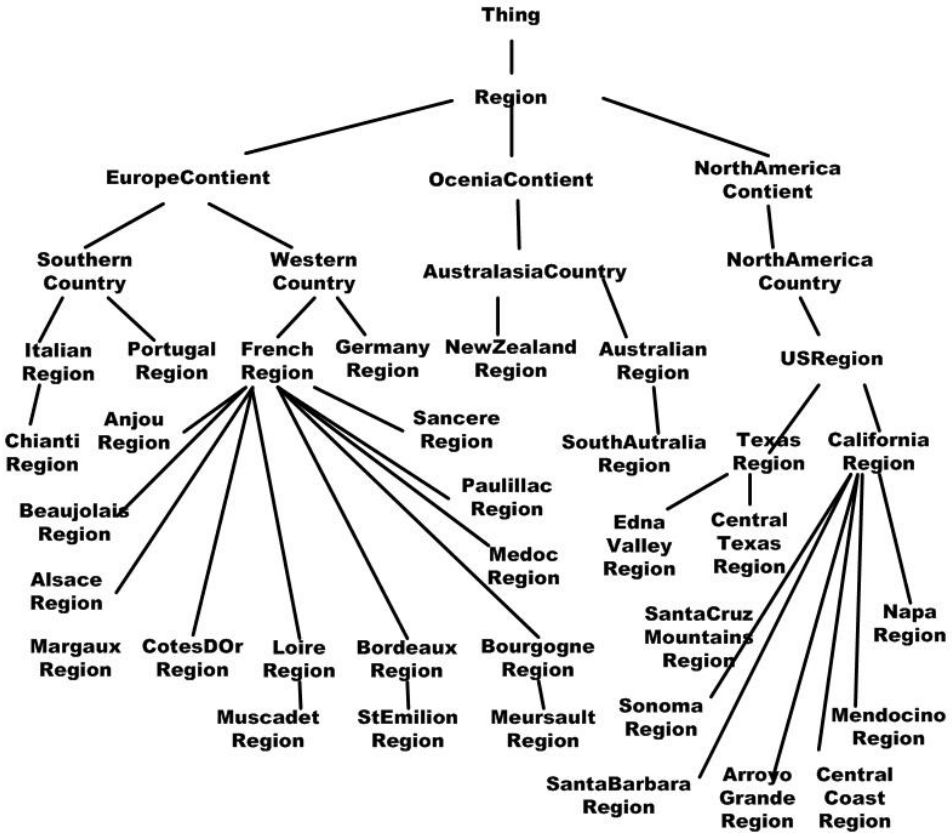


Figure 6.3. Wine region hierarchy.

we have `AnjouRegion` and `BordeuxRegion` values, this proportion for generalizing this feature to `FrenchRegion` is equal to 3/15. After estimating the proportion, it is compared with the predefined threshold value, Φ. If the proportion is greater than this threshold, the attribute will generalize to the nearest common parent. The reason for the nearest common parent is that minimal generalization of the hypothesis is desired.

If the attribute does not have any hierarchical structure, we estimate the average similarity of the values that we have. At this point, our heuristic tells us that if the attribute has more dissimilar values, we have more probability to generalize this attribute to `Thing`, ? which means every value is accepted for that attribute. For instance, for `Body` attribute we have three values: `Light`, `Medium` and `Full`. Table 6.4 shows the semantic similarities for these values. Note that these values should be assigned by a domain expert. However, in this study we assign these values in a way that there are three levels (light, medium, full) so we divide one by three ($1/3 = 0.3$). We give this ratio to the most different values (light and full) and for other values we add this to 0.3 so that the similarity medium and light becomes 0.6. According to our heuristic, if we have light and

Table 6.4. Semantic similarities for body

Body 1	Body 2	Similarity
Light	Full	0.3
Medium	Light	0.6
Full	Medium	0.6

full, we are more likely to generalize to ? than the case when we have light and medium.

6.2.3. Preference Prediction Algorithm

Each training sample is given as an input to the system. For each sample, both the most general and most specific sets are modified. Modifications depend on the type (positive or negative) of the sample. Figure 6.4 shows the general flow of the training process. If the sample is positive, it is added to the positive set, P (Line 2); otherwise it is added to the negative set, N (Line 6). Correspondingly, the most specific set (S) and the most general set (G) are updated.

1: **if** x *is positive* **then**
2: $P \leftarrow P + \{x\}$
3: $H_v^S \leftarrow$ updateSpecificSetForPositiveSample(x)
4: updateGeneralSetForPositiveSample(x,H_v^S)
5: **else**
6: $N \leftarrow N + \{x\}$
7: updateGeneralSetForNegativeSample(x)
8: removeLessGeneralFromGeneralSet()
9: updateSpecificSetForNegativeSample(x)
10: **end**

Figure 6.4. Revisable Candidate Elimination Algorithm (RCEA).

In our algorithms, the following notation is used.

- x: current training sample
- H_i^S: i^{th} hypothesis in the most specific set, S
- H_i^G: i^{th} hypothesis in the most specific set, G
- N_i: i^{th} negative sample in the negative set, N
- P_i: i^{th} positive sample in the positive set, P
- $Sim_z(x, y)$: similarity of x to the hypothesis y using the similarity metric z
- $sort_\Theta(Sim_z(x, S))$: returns the similarity values and the indices of the hypotheses in S, which are similar to the current sample by the similarity greater than Θ in descending order
- $x[i]$: the value of i^{th} attribute of the current sample
- $Attr_i$: i^{th} attribute
- ?: a special value for an attribute that means any value is acceptable
- $H_?$: the most general hypothesis consisting of "?" for each attribute

updateSpecificSetForPositiveSample(x): According to the algorithm in Figure 6.4, if the sample is positive, the algorithm updates the specific set to cover the current positive example (Line 3). The specific set should include at least one hypothesis that covers this positive sample. That is, one hypothesis in S needs to be chosen and generalized to cover this sample. The hypothesis that is chosen is the most similar hypothesis to the current sample. This is done to ensure that S is generalized as minimally as possible. The details of this process are explained in the algorithm depicted in Figure 6.5.

That is, for each specific hypothesis, a similarity is estimated and the hypotheses whose similarity with the current sample is greater than Θ are sorted according to their similarity value in a descending order (Line 3). Starting with the first hypothesis in this list, we generalize the specific hypothesis to cover the new example (Line 5). Afterward, the algorithm tests whether the extended specific hypothesis covers any negative example in the negative sample set (Line 6). If this hypothesis covers a negative example, the algorithm interprets that this extension is invalid so it passes to the next specific hypothesis in the ordered list to generalize in order to cover the new positive sample. This process continues until the extended specific hypothesis does not cover any negative

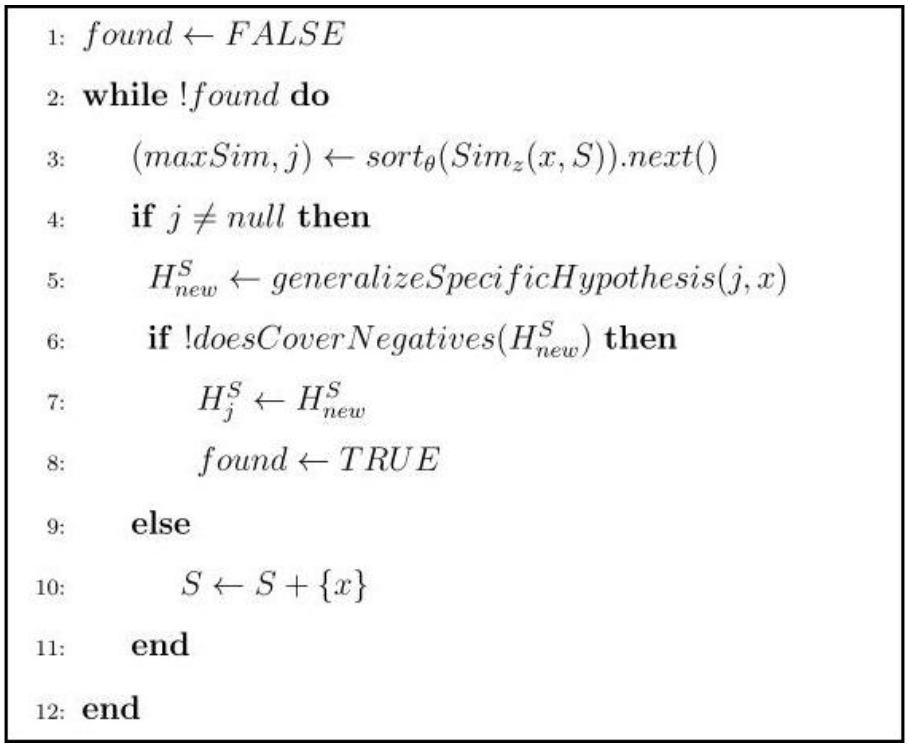


Figure 6.5. Updating the specific set when a positive sample x comes.

examples or the list is exhausted. If a hypothesis with a valid extension is found, the original hypothesis is replaced with its extended version (Line 7). On the other hand, if the list is exhausted or there is no hypothesis whose similarity is greater than Θ, the new positive example is added as a separate hypothesis into most specific set (Line 10).

We investigate the complexity analysis of the algorithm in Figure 6.5 as follows. Let m denote the number of hypotheses in S and k denote the number of attributes. The computational complexity of estimating similarity of the positive example to each specific hypothesis is $O(k)$ and sorting them in descending order is $O(mlog\ m)$, yielding $O(kmlog\ m)$. Note that it is enough to do this computation once and to use this ordered list with estimated similarity values in the while loop (Line 3).

If it is possible to generalize a particular specific hypothesis to cover the given sample, we do so and check whether the hypothesis covers any negative examples. The complexity of the generalization process is $O(kh)$ where h denotes the height of the hierarchy tree for a given issue (see the complexity analysis of the algorithm shown in Figure 6.6 for more detail). The computational complexity of checking all negative examples is $O(kn)$ where n is the number of negative examples. Overall this gives $O(k(h+$

n)). We may end up repeating this procedure m times; i.e., until we find a specific hypothesis that does not cover the negative examples or until S is exhausted. The time complexity then is $O(km(h+n))$. Overall computational complexity would then be $O(km(log\ m+h+n))$.

generalizeSpecificHypothesis(x, H_j^S): *Generalize*(H_j^S, x) results in H_i^S such that $H_i^S \supset H_j^S$ and $H_i^S \supset x$. The algorithm in Figure 6.6 indicates how the j^{th} specific hypothesis is generalized when the positive sample x is received. For each attribute, we check if the k^{th} attribute of the specific hypothesis covers the k^{th} attribute of the current sample x (Line 2). If the value of the current positive sample is not covered, the k^{th} attribute value is generalized.

```
1:  for k ← 0 to Attr.size do
2:   if !doesCover(H_j^S[k], x[k]) then
3:    if hasHierarchy(Attr_k) then
4:        H_j^S[k] ← H_j^S[k] + x[k]
5:        comParent ← findCommonParent()
6:        proportion ← |H_j^S[k]|/|comParent|
7:        if Φ ≺ proportion then
8:            H_j^S[k] ← comParent
9:    else
10:       if Φ < (1 − Sim_z(x[k], H_j^S[k])) then
11:           H_j^S[k] ←?
12:       else
13:           H_j^S[k] ← H_j^S[k] + x[k]
14:       end
15:   end
16: end
```

Figure 6.6. Generalizing a specific hypothesis H_j^S to cover x.

This generalization for an attribute having a hierarchy is performed as explained in Section 6.2.2 (Lines 3–8). If there is no hierarchy for this attribute, the dissimilarity $(1 - Sim_z(x[k], H_j^S[k]))$ is estimated between attribute values of hypothesis and current

sample (Line 10). If this dissimilarity is greater than the threshold, the attribute will be generalized to ? (Line 11). Note that the dissimilarity shows the diversity of values that the attribute can have. This means that this attribute can be generalized to ?, which involves all values for that attribute. Otherwise, the value of x is added to the hypothesis (Line 13).

This process can be exemplified as follows. Assume the service has three attributes: `Region`, `Color` and `Sugar`. The former two attributes have a hierarchy and the last attribute can take three possible values: `Dry`, `OffDry`, and `Sweet`. The color attribute can be `Red`, `Rose`, `White`, and `Reddish`, which is the parent of `Rose` and `Red`. There are 41 possible values for the region attribute and its hierarchy is more complicated.

Let us walk over the interactions depicted in Table 6.5. The consumer agent asks for <`Chianti`, `Rose`, `OffDry`> as a first request. Since there is no hypothesis in S, the first request is added as a hypothesis into S. In the second turn, the consumer requests <`California`, `White`, `Sweet`>. The similarity between this request and the hypothesis in S is estimated as 0.28 according to RP similarity metric. Assume that the threshold value is equal to 0.5 during the learning process. Because the similarity is less than the threshold value (θ), the second request is added into system as a separate hypothesis. Next time, the consumer requests <`Italian`, `Red`, `Dry`>. The similarity for the first hypothesis in S is 0.68. Therefore, the algorithm first tries to generalize the first hypothesis and the extended hypothesis becomes <(`Italian`), (`Reddish`), (`OffDry, Dry`)>. The system checks whether any negative sample is covered by this generalized hypothesis. If it does not cover any negative sample, the generalization of the specific set is completed. Otherwise, the algorithm undoes the generalization of the first hypothesis. Note that the nearest common parent for `Red` and `Rose` is `Reddish` and `Italian` subsumes `Chianti`. The dissimilarity of `OffDry` and `Dry` is 0.2. Since it is less than threshold Φ, for this attribute, generalization is not performed. We investigate the complexity analysis of the algorithm in Figure 6.6 as follows. If the issue values are structured in a hierarchy, we need to find the nearest common ancestor of two values to generalize. Let k denote the number of attributes and h the height of the hierarchy for a given issue. The computational complexity of finding the common ancestor of two nodes is equal to $O(h)$ at worst case. If there is no hierarchy for that issue, we only do a single look up to find the

dissimilarity of the issues whose complexity is $O(1)$. Since we perform the generalization for each issue, the complexity would be $O(kh)$ at worst case.

Table 6.5. Interaction between consumer and producer.

Turn	**Offer <Region, Color, Sugar>**
Consumer offers (R_1)	<Chianti, Rose, OffDry >
Learnt concept:	$S_0 = \{R_1\}$ $G_0 = \{(?, ?, ?)\}$
Producer offers	<Chianti, Rose, Sweet>
Learnt concept:	$S_1 = \{R_1\}$ G_1={(?, ?, OffDry)}
Consumer offers (R_2)	<California, White, Sweet>
Learnt concept:	$S_2 = \{R_1, R_2\}$ G_2={(?, ?, OffDry), (NorthAmerica, ?,?) , (?, White, ?) }
Producer offer	<US, Rose, OffDry>
Learnt concept:	$S_3 = \{R_1, R_2\}$ G_3={(?, White, ?),(NorthAmerica, ?, Sweet), (California, ?, ?), (Europe, ?, OffDry) }
Consumer offer (R_3)	<Italian, Red, Dry>
Learnt concept:	S_4= {(Italian, Reddish, [OffDry,Dry]), R_2} G_4= G_3∪ (?, ?, Dry)

updateGeneralSetForPositiveSample(x, H_v^S): Since our learning algorithm allows disjunctives, there may be a variety of general hypotheses consistent with different positive samples. Unlike CEA, our algorithm does not eliminate any general hypotheses not covering the current positive sample. Instead, positive sample should be covered by at least one hypothesis in the general set. If none of the hypotheses cover the new positive sample, new general hypotheses covering this positive sample but not covering any negative samples are added into the most general set (Line 4 in RCEA algorithm shown in Figure 6.4). This revision process is a new operation for Version Space and does not exist in CEA. Using this operation, our algorithm supports learning disjunctive concepts. The algorithm in Figure 6.7 shows the general picture for updating general set when a positive sample comes.

```
1: if needRevision(x) ≡ TRUE then
2:     reviseGeneralSet(H_v^S)
3:     removeLessGeneralFromGeneralSet()
```

Figure 6.7. Updating the general set by using the specific hypothesis H_v^S covering x.

reviseGeneralSet(H_v^S): As specified in algorithm depicted in Figure 6.7, the algorithm first checks whether there is a need for revision. If so, new general hypotheses that cover the positive sample but not cover the previous negative samples are generated from the most general hypothesis by the help of the most similar specific hypothesis to current sample.

The algorithm in Figure 6.8 involves how the revision process is performed. This procedure takes the specific hypothesis (H_v^S) covering the current positive sample. First, the most general hypothesis consisting of ? is added into possible hypothesis set (Line 1). For each negative sample in the negative sample set (N), all possible hypotheses in possible hypothesis set is checked for covering one of the negative samples (Line 5). For each possible hypothesis covering the values of the negative sample, the values of the specific hypothesis, H_v^S, are used for specializing the possible hypotheses (Line 10). If there is a hierarchy for that attribute of the specific hypothesis, the most general parent concept in the hierarchy not covering the negative example is found (Line 12). Then, by setting this value to the current possible hypothesis a new hypothesis is created and added as a separate hypothesis into the possible hypothesis set (Lines 13–14). If there is no hierarchy for this attribute, the value of specific hypothesis is used directly (Lines 16–18).

The process can be illustrated with the following example. In the second turn in Table 6.5, the consumer makes a request as <`California`, `White`, `Sweet`>. Since the current most general set G, <?, ?, `OffDry`> is consistent with only the wines whose sugar level is `OffDry`, the current request is not accepted by the current G. Therefore, revision is needed. According to our revision algorithm, first the most general hypothesis, <?, ?, ?> is generated as a possible hypothesis. In a loop, the set of possible hypotheses is checked to see whether the hypotheses in this set covers any negative sample. If a

```
1: HypoSet[0] ← H?
2: for i ← 0 to |N| do
3:  for j ← 0 to |HypoSet| do
4:   H_t ← HypoSet[j]
5:   if doesCover(H_t, N_i) then
6:    HypoSet ← HypoSet − {HypoSet[j]}
7:    for k ← 0 to Attr.size do
8:     if doesCover(H_t[k], N_i[k]) then
9:        for r ← 0 to |H_v^S[k]| do
10:          if !doesCover(H_v^S[k][r], N_i[k]) then
11:             if hasHierarchy(Attr_k) then
12:                g ← ParentNotCover(N_i[k])
13:                H_t[k] ← g
14:                HypoSet ← HypoSet + {H_t}
15:             else
16:                H_tnew ← H_t
17:                H_tnew[k] ← H_v^S[r]
18:                HypoSet ← HypoSet + {H_tnew}
19:             end
20:       end
21:    end
22:  end
23: end
```

Figure 6.8. Revising the general set by using the specific hypothesis H_v^S.

hypothesis covers a negative sample, the algorithm compares each attribute of the negative sample with the specific hypothesis involving the current positive sample. In detail, our negative sample set only includes <`Chianti`, `Rose`, `Sweet`>. The specific hypothesis involving the current positive sample is <`California`, `White`, `Sweet`>. The first two attributes are different. The possible hypotheses set includes only one hypothesis, <?, ?, ?>. Two new possible hypotheses may be generated as <`California`, ?, ?> and <?, `White`,?>. Since the hypotheses in G should be as general as possible, we make another revision. By using hierarchical information kept in the ontology, we can find the most general parent of this value not involving the negative value `Chianti`. Since `NorthAmerica` is the most general concept subsuming `California`, the new possible hypothesis becomes <`NorthAmerica`, ?, ?> and <?, `White`, ?)>. If newly constructed possible hypotheses cover any negative samples, they are modified not to cover negatives any more. As a result, these two newly generated hypotheses <`NorthAmerica`, ?, ?> and <?, `White`, ?> are directly added into the most general set since they do not cover any negative samples.

We investigate the complexity analysis of the algorithm in Figure 6.8 as follows. The complexity of checking whether a hypothesis covers a negative example is equal to the number of attributes, $O(k)$. In a loop, each candidate general hypothesis in $HypoSet$ is checked to see whether it covers any negative example in N. If it does, we generate possible specialization of that hypothesis in a way that it does not cover any negative example but covers the given specific hypothesis. To do this, for each candidate hypothesis in $HypoSet$, we compare the values of specific hypothesis with that of the current negative example for each attribute (Line 10). At worst case, the cost of specializing a single general hypothesis and generating new hypotheses not covering a negative example is $O(pk)$ where p denotes the maximum number of possible values for an attribute in the system. Note that $HypoSet$ includes only the most general hypothesis consisting of ? at the beginning and it grows with respect to the content of the negatives and the given specific hypotheses. At worst case, we specialize a candidate hypothesis and generate (pk) different hypotheses. The computational complexity of specializing the hypothesis in $HypoSet$ would be $O(p^2k^2)$ if we assume that we have pk hypotheses in $HypoSet$. Note that the size of $HypoSet$ is dynamic and may vary for each run. Since we traverse all of the negative examples in N to check whether any candidate general hypotheses covers any of them, the overall complexity would be $O(p^2k^2n)$, which

means that the revision process will grow polynomially with the number of attributes, the number of values that a specific hypothesis contains for each issue, and the number of negative samples.

updateGeneralSetForNegativeSample(x): The aim of this method is to specialize the hypotheses in G, which cover the negative example, in a minimal fashion. Minimal specialization is crucial since it is desired that the hypotheses remain as general as possible. Therefore, the algorithm in Figure 6.9 checks all the hypotheses in the most general set to see if they cover the negative sample (Line 2). Each hypothesis covering the negative sample is backed up as an abandoned hypothesis (Line 3) since the information kept in those hypothesis should not be lost before removing them. Then, these hypotheses are removed from the most general set (Line 4). By only taking the hypotheses in the most specific set that are covered by the abandoned hypotheses (Line 6) into account, a minimal specification of the abandoned hypotheses that do not cover the negative sample are generated. To accomplish this, the algorithm compares the value of each attribute of specific hypothesis with that of the negative sample (Line 11 & Line 16). If the values are different and there is no hierarchy for the attribute, the value of attribute of the specific hypothesis is replaced in the abandoned hypothesis (Lines 17–18). If the values are different and there is a hierarchy for that attribute, the most general common parent of the value of the specific hypothesis, which does not cover the negative sample, is found (Line 12) and this value is replaced in the abandoned hypothesis (Lines 13–14).

How this process is performed can be exemplified with the following example. As specified in Table 6.5, at first G is equal to the most general hypothesis, <?, ?, ?>. After first negative example <`Chianti`, `Rose`, `Sweet`>, there is a need for specialization of hypotheses in G to make them not cover the negative samples. To accomplish this, for each hypothesis in G covering negatives, we use the hypotheses in S. Some hypotheses in G may not cover some particular hypothesis in S. Therefore, when updating the hypothesis in G, we should only consider the related hypotheses in S. In this example, S includes only <`Chianti`, `Rose`, `OffDry`> and this is covered by <?, ?, ?>. Now, the algorithm compares the values of the attributes in specific hypothesis with those of the negative sample one by one and it selects the attributes whose values are different. For example, the sweetness degree in specific hypothesis is `OffDry` whereas that for negative

```
1:  for i ← 0 to |G| do
2:   if doesCover(H_i^G, x) then
3:    abandoned ← H_i^G
4:    G ← G − {H_i^G}
5:    for j ← 0 to |S| do
6:     if doesCover(abandoned, H_j^S) then
7:      for k ← 0 to Attr.size do
8:          for t ← 0 to |H_j^S[k]| do
9:              temp ← abandoned
10:             if hasHierarchy(Attr_k) then
11:                 if !doesCover(H_j^S[k][t], x[k]) then
12:                     gnew ← ParentNotCover(x[k])
13:                     temp[k] ← gnew
14:                     G ← G + {temp}
15:             else
16:                 if !doesCover(H_j^S[k][t], x[k]) then
17:                     temp[k] ← H_j^S[k][t]
18:                     G ← G + {temp}
19:             end
20:         end
21:     end
22:   end
23: end
```

Figure 6.9. Updating the general set when a negative sample x comes.

sample is `Sweet`. From this information, the algorithm modifies general hypothesis as <?, ?, `OffDry`>. Since sugar level does not have hierarchy, we modify the general hypothesis directly. In the case of hierarchy, we find the most general concept that does not cover the value of the negative sample.

We investigate the complexity analysis of the algorithm in Figure 6.9 as follows. We traverse each general hypothesis and check if it covers the given negative example, yielding a time complexity of $O(tk)$ where t is the number of general hypotheses in G and k is the number of attributes that a hypothesis involves. Next, for each hypothesis covering the given example (maximum is t), we compare the values of the specific hypothesis to those of the negative example for each attribute. If the values are different, we specialize the current general hypothesis. Let p denote the number of possible values for an attribute. The complexity of the comparison and specialization process is $O(pk)$. We repeat this process for each specific hypothesis, yielding $O(pkm)$ (where m is the number of specific hypotheses in S), yielding an overall complexity of $O(pkmt)$. Since the complexity of the first part is insignificant ($O(tk)$), we conclude that the overall complexity of updating general set is $O(pkmt)$ at worst case.

removeLessGeneralFromGeneralSet(): If there are some hypotheses in G less general than the other hypotheses in G, these are removed from the system. Because, the aim is to keep the most general hypotheses in G. $LessGeneral(H_i, H_j)$ means that for each k, $H_j[k] \supseteq H_i[k]$ and $H_i \neq H_j$. To achieve this, the hierarchical information can be used. In the hierarchy, a child concept is less general than its ancestors. If there is no hierarchy for that attribute, any value is less general than ?.

updateSpecificSetForNegativeSample(x): If there are hypotheses in the most specific set that cover a negative sample, they are removed. Since each positive sample should be covered by at least one of the hypotheses in S, each positive sample in P is checked whether it is covered by S. If none of the specific hypotheses in S covers a positive sample, this positive sample is added as a separate hypothesis to S. Consequently, each positive sample in P will be covered by the most specific set.

6.3. Producer's Negotiation Strategy

Figure 6.10 shows the abstract view of the producer agent's strategy involving three significant components: bid generator, bid decider and preference predictor. As seen from the figure, the producer has a service inventory involving its available services.

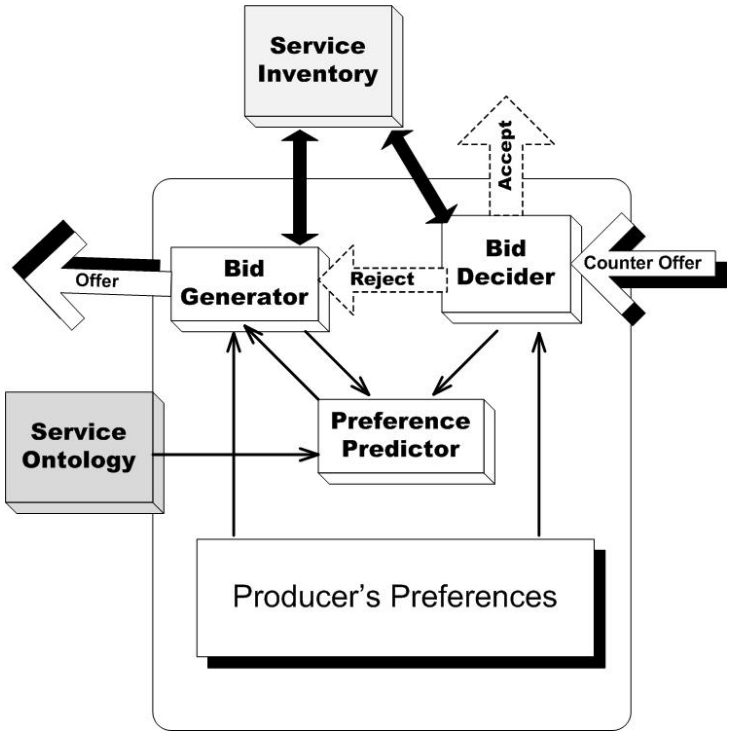


Figure 6.10. Producer agent

During the negotiation, the producer tries to predict the consumer's preferences from the bids exchanged (preference predictor module). Any concept-based learning algorithm can be adopted here. When the producer agent generates its offer (bid generator), it considers both its available services and their utilities for the producer and the consumer's predicted preferences. The producer does not offer a service, which is classified as a rejectable service by the preference prediction algorithm. Among services classified as acceptable, producer chooses the service having the highest utility value for the producer.

In detail to generate the best offer, the producer agent uses its service inventory and one of the inductive learning algorithm such as CEA, DCEA, RCEA, ID3 or Bayesian classifier. The service offering mechanism is the same for both the original CEA, DCEA

and RCEA, but their methods for updating G and S are different.

The producer uses the hypotheses in G to filter out its inventory. That is, if a service in a stock is not covered by G, then it is assumed that it will not be accepted by the consumer since it does not fit the modeled preferences. The producer assigns a utility value to each of its services and prefers to offer a service that is both an acceptable service as far as the consumer's preferences are concerned and a desired service whose utility is more than others for the producer. Among the services that are likely to be offered, an average similarity value is estimated with respect to the hypotheses in S. At the end, the most similar service is offered to the consumer.

If the producer tries to predict the consumer's preferences with ID3, a similar mechanism is applied with two differences. First, since ID3 does not maintain G, the list of unaccepted services that are classified as negative by the decision tree are removed from the candidate service list. Second, the similarities of possible services are not measured with respect to S, but instead to all previously made requests. Note that ID3 is not an incremental algorithm, which means it requires all the training samples at the beginning of the training. To overcome this problem, the system keeps consumer's requests throughout the negotiation interaction as positive samples and all counter offers rejected by the consumer as negative examples. After each coming request, the decision tree is rebuilt. Similarly, the producer using Bayesian eliminates the services that are classified as negative. After selecting the services having the highest utility for the producer, the most similar service to the positive sample set is offered by the producer.

When the producer agent evaluates the consumer's counter-offer (bid decider), it acts collaboratively. That is, if the producer has the consumer's offer, it provides this service and complete the negotiation.

7. EVALUATION OF RCEA IN NEGOTIATION

To evaluate our preference prediction algorithm, we construct an environment in which a consumer agent negotiates with a producer agent on the service of wine selling as explained before. For this purpose, we use an extension of the well-known wine ontology[2] providing a huge number of alternative wine services. In our negotiation setting, the consumer's preferences are represented with conjunctive and disjunctive constraints. The producer agent has a service inventory including its available services and ranks each service with respect to their utility for the producer. Our test environment is written in Java and Jena[3] is used as ontology reasoner.

We evaluate how well an agent negotiates when it predicts its opponent's preferences using RCEA. We could not compare our algorithm directly with the existing approaches in negotiation because of diversity in settings. First, our system requires a service ontology holding semantic similarities and hierarchical information about the issue domain whereas other approaches do not use such a service ontology. Second, some studies need additional knowledge about the domain. For instance, fuzzy similarities are needed for Faratin's trade-off strategy [34] where we use semantic similarities in our work. Third, in contrast to other approaches [23, 26], in our setting the producer agent has a service inventory and it can only generate a counter-offer from its available services in its repository. However, other approaches usually do not consider a limited service repository as we did. Therefore, we compare the performance of our algorithm with other four alternatives, namely Candidate Elimination Algorithm (CEA), Disjunctive Candidate Elimination Algorithm (DCEA), ID3 and Bayesian Classifier.

7.1. Negotiation Domain

For our experiments, we use wine domain including seven negotiation issues: `Sugar level`, `Flavor`, `Body`, `Wine Color`, `Region`, `Winery`, and `Wine Grape`. For these issues, there are three, three, three, three, 34, 26, 15 possible values respectively. Hence, there

[2]http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf
[3]http://jena.sourceforge.net/

are a very large number of possible wine services. For some issues, there is a hierarchy of possible values exists while for other issues we define semantic similarities.

7.2. Negotiation Settings

During the negotiation, the consumer agent employs a negotiation strategy explained in Section 4.1. That is, the consumer agent generates its offer with respect to its preferences represented as conjunctive and disjunctive constraints. If the producer agent offers an alternative service that is consistent with the consumer's preferences, it accepts this offer. Otherwise, it may stick on its previous offers or propose a new offer.

As for the producer agent, it first filters out its service inventory on the basis of its prediction about the consumer's preferences. That is, the producer eliminates the services that would possibly rejected by the consumer. If the producer uses one of the Version Space based learning algorithms such as CEA, DCEA and RCEA, it uses G to filter out its service repository. Similarly, the producer using ID3 and Bayesian eliminates services that are classified as negative. Among the remaining services, the producer chooses the services whose utility is the highest for the producer. If there are more than one candidate service, it offers the most similar service to the consumer's predicted preferences. To do this, an average similarity to the hypotheses in S is estimated if the producer applies one of CEA, DCEA and RCEA. The reason is that S represents the consumer's predicted preferences minimally. In the case of ID3 and Bayesian, the producer offers the service that is most similar to consumer's previous offers. Note that after the consumer's each offer and the producer's each counter-offer that is rejected by the consumer, these producers train their own learning element accordingly.

7.3. Experimental Setting

In our setting, the same consumer agent negotiates with five producers having the same service inventory but employing a different learning algorithm to predict the consumer's preferences. As far as the performance of the negotiation is concerned, the number of interactions between the consumer and the producer is the main factor after whether having a successful negotiation. Obviously, completing the negotiation in as

few interactions as possible is desirable for both parties. The number of interactions depends on the producer's negotiation strategy, the producer's inventory, the consumer's preferences and the order and content of the consumer's offers. In our experiments, in order to analyze the performance of the producer agent's learning algorithm, simulation of negotiation is executed for different combinations of these parameters.

Since the producer's available services and the consumer's order of requests affect the negotiation time, to compare different producers fairly, 100 different producer inventories are generated randomly. Each inventory contains 50 different wine services. The utility of these services varies between one and five and are equally distributed in the inventory. It is worth noting that 100 different producer inventories show 100 producers having different service inventories. Further, for each scenario the negotiation process is repeated 10 times varying the request orders.

To construct the consumer preference profiles, we asked 20 people (students and faculty from the Department of Computer Engineering at Bogazici University) to create consumer profiles. First two preference profiles only consist of conjunctive constraints but other 18 preferences contain conjunctive and disjunctive constraints. Table 7.1 shows five representative preference profiles. The preferences of the consumers are taken from the user by using a simple interface. The output of this interface is a preference file including the preference information and it is used only by the consumer agent. That is, the producer agents cannot access this file.

Table 7.1. Selected consumer preferences.

Id	**Preference — [Sugar, Flavor, Body, Color, Region, Winery, Grape]**
1	[OffDry, ?, ?, White, ?, Cheap, ?]
2	[?, Moderate, ?, ?, US, ?, WhiteGrape]
4	[?, Moderate, ?, White, US, ?, ?] or [?, Delicate, ?, Red, Italian, ?, ?]
6	[Dry, Moderate, Medium, ?, (Italian, Portugal), ?, ?] or [Dry, ?, Light, Rose, ?, ?, ?]
7	[Sweet, ?, Light, ?, ?, ?, ?] or [Dry, Strong, ?, ?, ?, ?, ?] or [OffDry, ?, ?, ?, ?, ExpensiveWinery, RedGrape]

The satisfiability of the consumer's preferences over the inventories also varies. For example, the second preference is not satisfiable in 55 of the 100 inventories and in the remaining 45 inventories, there is only one service consistent with the second preference. For the seventh preference, all inventories have some services consistent with user preferences. On average, 16.62 of the 50 services in an inventory are desirable for the customer. Compared to the second preference, it is highly possible to find a desired service.

Table 7.2 summarizes our experimental setting.

Table 7.2. Experimental setting.

Number of times that the same consumer negotiates with the same producer	10
Number of producer service inventories	100
Number of available services in a service inventory	50
Number of negotiations for each consumer profile	1000
Number of consumer preference profiles	20

We first compare the performance of the producers using Version Space learning algorithms: RCEA, CEA and DCEA. Then, we compare our proposed algorithm, RCEA with adaptations of other well-known classifiers, mainly ID3 and Bayesian.

7.4. Case 1: Comparing RCEA, CEA, and DCEA

We experiment with three separate producers: Producer with Candidate Elimination Algorithm (PCEA), Producer with Disjunctive (PDCEA) and Producer with Revisable Candidate Elimination Algorithm (PRCEA). We have ten runs for 100 different inventories, resulting in 1000 negotiations for each consumer's preference profile. In each of the ten runs, the consumer agent starts the negotiation with a random offer that is compatible with its preferences. We repeat these experiments for 20 different consumer profiles. We evaluate the behavior of the producers using the following evaluation criteria: the performance, consistency, speed of success, and speed to detect failure.

7.4.1. Performance

We measure the performance as the number of negotiations that are finalized successfully. Table 7.3 shows the number of successful negotiations for each consumer preference profile out of 1000 negotiation trials. The last column shows the number of trials in which success was possible; i.e., the producer agent had a possible service in its inventory that could satisfy the consumer. The figures show that PDCEA always carries out a successful negotiation if there is a satisfying service. The reason for these perfect results is that DCEA does not filter the services except for the counter offers rejected by the consumer. Therefore, it never gives up offering services unless there are no remaining services in its inventory or its offer is accepted by the consumer. On the other hand, PRCEA's success is close but may miss up to 8.2% of the time. PCEA, on the other hand, has varying performance and may miss up to 85% of the possible services. This is mainly due to the fact that CEA cannot handle disjunctives properly.

We also apply the Friedman statistical test [57], which is often used for comparison of multiple classifiers in order to see whether the success of producers is statistically significantly different from each other. In Friedman test, the algorithms are ranked for each data set separately from the best performing algorithm to the worst performing algorithm [57]. According to the average ranks of the algorithms, this test compares the multiple classifiers. In our experiments, each negotiation is considered as a different data set because different training sets are obtained in each negotiation. We take alpha as 0.01, where alpha is the significance level of the statistical test. According to this statistical test, there is no statistically significant difference between the number of successful negotiations of PRCEA and that of PDCEA under 99 per cent confidence level. However, there is a statistically significant difference between those and that of the PCEA. Thus, it can be said that PRCEA is as successful as PDCEA on average.

Recall that, for each of the 100 producer's inventories, we test the performance with 10 different consumer request orders. It is also important for a producer to negotiate consistently in all ten orders. That is, a producer that negotiates well with a certain ordering of requests but not too well when the ordering changes is really not useful, since nothing can be guaranteed about its negotiation. This is what we investigate next.

Table 7.3. Case-1: Number of negotiations that end with consensus.

	# of success			Poss. Successful
Consumer's profile	PRCEA	PCEA	PDCEA	Negotiations
1	970	970	970	970
2	450	450	450	450
3	998	476	1000	1000
4	786	119	790	790
5	1000	694	1000	1000
6	894	349	920	920
7	1000	675	1000	1000
8	1000	562	1000	1000
9	968	201	970	970
10	960	193	990	990
11	1000	324	1000	1000
12	1000	331	1000	1000
13	1000	335	1000	1000
14	1000	579	1000	1000
15	999	336	1000	1000
16	1000	615	1000	1000
17	529	140	530	530
18	1000	616	1000	1000
19	989	255	990	990
20	817	306	890	890
Average:	918.0	426.3	925.0	925.0

7.4.2. Consistency

We measure the consistency as follows. We let a producer win a point if it completes all 10 negotiation trials successfully. This is estimated for all producer's inventories and summed up at the end. Therefore, it indicates the consistency of producer's overall success. Since the consistency performance of the algorithms can be at most the maximum consistency performance, we find the ratio of each producer to the maximum and take the average of these ratios to see the average consistency performance of each producer. According to Table 7.4, PDCEA and PRCEA have the most consistent performance whereas the performance of PCEA is much below. For the first two preference profiles, the consistency performance of PCEA is the same with those of PDCEA and PRCEA because these preferences include only conjunctives.

When we apply the Friedman test to these results, it can be said that there is no significant difference between the consistency of PRCEA and PDCEA but there is a statistically significant difference between these and the consistency of PCEA under 99 per cent confidence level. Predicting the consumer's preferences during the negotiation enable the producer to generate accurate offers that are more likely to be acceptable for the consumer. This increases the number of success negotiations. The above results show that PCEA fails in significant number of negotiations whereas both PDCEA and PRCEA are successful in most of the negotiations. The result is exemplified if we also consider the consistency. Between PDCEA and PRCEA, we see that PDCEA is slightly more successful in providing a service that will satisfy the consumer. Now, we study how fast the participants reach a consensus.

7.4.3. Duration

We measure the duration as the average number of interactions for successful termination in the case that all producers have a success at the same run. An interaction involves that the consumer proposes an offer and the producer responses with a counter-offer. We select a subset of the negotiations in which all three producers have been successful and study the average number of interactions needed for termination.

Table 7.4. Case-1: Consistency performance of the producers.

Consumer's profile	PRCEA	PCEA	PDCEA	Max*
1	97	97	97	97
2	45	45	45	45
3	98	4	100	100
4	75	0	79	79
5	100	8	100	100
6	74	6	92	92
7	100	6	100	100
8	100	7	100	100
9	95	0	97	97
10	74	1	99	99
11	100	0	100	100
12	100	0	100	100
13	100	1	100	100
14	100	3	100	100
15	99	2	100	100
16	100	6	100	100
17	52	0	53	53
18	100	4	100	100
19	98	1	99	99
20	59	8	89	89
Average:	95.42	12.93	100.00	100.00
*: The maximum consistency performance value				

Table 7.5 shows that when PCEA is successful in negotiation, then the number of interactions it requires is quite low. Recall that PCEA is unsuccessful in many interactions as seen in Table 7.3. This shows that PCEA either completes a negotiation quickly or fails (mostly when the preferences are disjunctive). When PRCEA and PDCEA are compared, we see that PDCEA needs more interactions to end a negotiation successfully. The reason stems from the fact that DCEA does not filter the candidate services in accordance with the consumer's preferences as well as RCEA does. Hence, it requires more time to complete the negotiation.

In 8525 out of 20000 negotiation runs all of the producers, RCEA, CEA and DCEA, are successful. We apply Friedman test to the results of the 8525 runs. Under 99 per cent confidence level, the results of PRCEA, PCEA and PDCEA are statistically significantly different from each other and according to the number of interactions when all are successful, the ordering can be represented as $PCEA \prec PRCEA \prec PDCEA$. This means that PRCEA negotiates faster than PDCEA and when all producers have a success, PCEA negotiates faster than PRCEA and PDCEA. Recall that when the preferences are both conjunctive and disjunctive, PCEA fails. As well as completing a successful negotiation as fast as possible, it is also important to realize a possible failure as early as possible. This is what we investigate next.

7.4.4. Duration to detect failure

We measure the duration to detect failure as the average number of interactions needed to decide that a negotiation will not terminate successfully because the producer inventory does not have a useful service for the consumer. This is important since ideally the preference prediction algorithm should detect that the consumer' preferences cannot be satisfied as early as possible, so as to not tire the consumer out.

Table 7.6 shows the average number of interactions needed to decide that the consumer's preferences cannot be satisfied by the available services in the producer's service inventory. For some preference profiles, all inventories have some satisfactory services so we are interested in cases for other preferences.

Table 7.5. Case-1: Average number of interactions when all are successful.

Consumer's profile	PRCEA	PCEA	PDCEA
1	5.14	3.13	11.62
2	7.42	3.98	19.63
3	2.50	1.76	2.61
4	7.36	2.40	9.21
5	1.93	1.56	2.07
6	4.25	2.35	5.56
7	1.63	1.40	1.64
8	1.76	1.41	1.66
9	4.64	2.12	5.26
10	4.22	1.98	5.66
11	2.33	1.57	2.38
12	2.84	1.74	2.95
13	2.16	1.50	2.16
14	2.26	1.64	2.37
15	2.98	1.70	3.63
16	1.49	1.36	1.50
17	5.86	2.09	9.14
18	1.79	1.49	1.76
19	3.17	1.85	3.73
20	5.42	2.56	9.23
Average:	3.18	1.97	4.94
Standard Deviation:	4.02	1.34	8.20

Table 7.6. Case-1: Average number of interactions when inventories do not have the desired service.

Consumer's profile	PRCEA	PCEA	PDCEA
1	12.70	5.40	50.00
2	12.89	5.40	50.00
4	33.27	3.15	50.00
6	33.09	4.33	50.00
9	36.60	3.27	50.00
10	29.30	2.90	50.00
17	27.24	3.46	50.00
19	40.40	3.90	50.00
20	22.83	3.85	50.00
Average:	27.59	3.96	50.00

As seen in Table 7.6, PDCEA has 50 interactions for each case, which is the worst case. This is due to the nature of DCEA. Since it only filters out the producer's counter offers rejected by the consumer, all remaining services are still plausible. On the other hand, RCEA generalizes the rejected counter-offers and eliminates others that are similar to those that were rejected. Hence, PRCEA ends the negotiation in a reasonable time, without showing the consumer all possible services in its inventory. This is an important advantage of PRCEA over PDCEA. Meanwhile, PCEA completes the negotiations even earlier than PRCEA. However, this situation does not show that PCEA performs better because the number of successful runs for PCEA is much less than that for PRCEA.

These results show that CEA is not convenient in our negotiation domain, since it does not support disjunctives. DCEA is one of the alternative learning algorithms that can be used in predicting the consumer's preferences but the number of interactions that are necessary to complete the negotiation when none of the producer's available services is high when compared to RCEA. We conclude that among the three Version space algorithms, RCEA performs better than both CEA and DCEA, when different success and performance criteria are considered.

7.5. Case 2: Comparing PRCEA, PID3, and PBayesian

In this section, we compare RCEA with other well-known classifiers such as ID3, and a Bayes' classifier when we applied in the context of negotiation for predicting preferences. We experiment with three separate producers, Producer with Revisable Candidate Elimination Algorithm (PRCEA), Producer with ID3 (PID3) and Producer with Bayes' classifier (PBayesian). We use 20 consumer preference profiles for evaluation and we have 10 runs for 100 different inventories, resulting in 1000 negotiations. We evaluate the behavior of the producers using the evaluation criteria that we defined before: the performance, consistency, speed of success, and speed to detect failure.

7.5.1. Performance

We first study the number of negotiations that are finalized successfully. Table 7.7 shows the number of successful negotiations for each consumer's preference profile out of 1000 negotiation trials. The last column shows the number of trials in which success was possible. On average the number of successful negotiations for PRCEA and PID3 are similar (918 vs. 916.8) and that for PBayesian less than that of PRCEA and PID3 (891.55).

The results show that success of the algorithms are close to each other. We see that PRCEA is slightly more successful in providing a service that will satisfy the consumer. Now, we study how fast they negotiate.

7.5.2. Duration

When we evaluate the producers with respect to their negotiation duration, we take into consideration 17665 negotiation runs out of 20000 in which all of the producers, PRCEA, PID3, and PBayesian are successful at the same run. Table 7.8 compares the three producers in terms of average number of interactions needed to complete a negotiation successfully when all producers reach a consensus with the consumer. Accordingly, PRCEA needs fewer interactions to end a negotiation successfully when its performance is compared to others. We apply Friedman test to the results of the 17665 runs. Under

Table 7.7. Case-2: Number of negotiations that end with consensus.

	# of success			Poss. Successful
Consumer's profile	PRCEA	PID3	PBayesian	Negotiations
1	970	969	963	970
2	450	437	401	450
3	998	1000	998	1000
4	786	790	739	790
5	1000	1000	1000	1000
6	894	871	813	920
7	1000	1000	1000	1000
8	1000	1000	999	1000
9	968	958	899	970
10	960	983	946	990
11	1000	1000	986	1000
12	1000	999	989	1000
13	1000	994	981	1000
14	1000	1000	999	1000
15	999	988	977	1000
16	1000	1000	1000	1000
17	529	526	395	530
18	1000	1000	999	1000
19	989	972	965	990
20	817	849	782	890
Average:	918.00	916.80	891.55	925.00

99 per cent confidence level, the results of PRCEA, PID3, and PBayesian are statistically significantly different from each other and according to the number of interactions when all are successful, the ordering can be represented as $PRCEA \prec PBayesian \prec PID3$, which means that PRCEA negotiates faster than PBayesian, which negotiates faster than PID3 when all producers complete the negotiation successfully.

Table 7.8. Case-2: Average number of interactions when all are successful.

Consumer's profile	PRCEA	PID3	PBayesian
1	5.10	9.08	8.12
2	6.97	15.21	10.68
3	4.68	4.87	5.27
4	14.21	14.52	12.99
5	2.79	2.88	3.26
6	8.64	9.98	8.79
7	2.48	2.53	2.66
8	3.27	3.27	3.39
9	11.61	13.04	11.51
10	11.24	11.92	11.66
11	6.88	7.50	7.07
12	7.00	7.47	7.21
13	6.87	7.23	6.83
14	3.56	3.68	4.06
15	7.64	9.32	8.46
16	2.38	2.48	2.49
17	11.69	13.73	11.47
18	2.88	3.06	3.46
19	8.26	9.73	8.30
20	9.07	11.68	10.65
Average:	6.48	7.50	6.97
Standard Deviation:	6.66	7.78	6.53

Moreover, we investigate how many times each producer negotiates as fast as the fastest producer out of 17665 negotiations. The first part of Figure 7.1 shows this information for each producer graphically. It is seen that PRCEA completes negotiations early in most of the cases (12030), which supports the above results. The number of times that PID3 reaches a consensus early is higher than the number of times that Bayesian reaches a consensus ($9686 > 7562$), but still less than that of PRCEA. We also calculate the proportion of the number of minimum interactions to the number of interactions that producer needs for each negotiation. This proportion shows the relative speed (i.e., in terms of number of interactions) of an algorithm compared to the best algorithm for a given negotiation instance. Higher proportion means that the number of interactions that the producer needs is closer to the fastest producer.

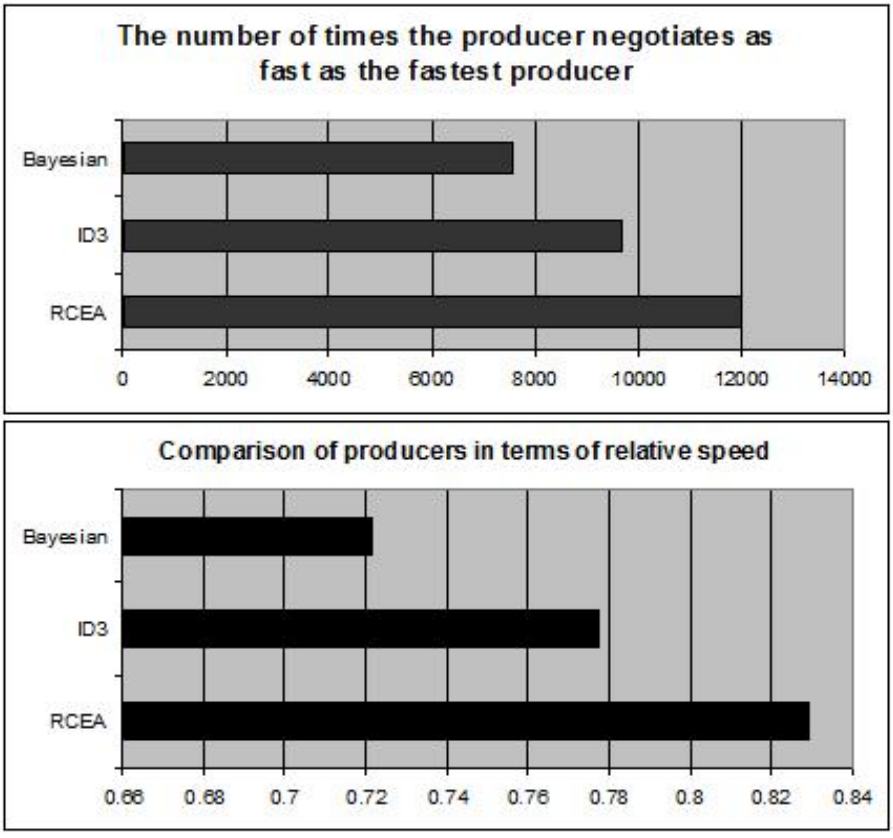


Figure 7.1. Performance of RCEA, ID3, and Bayesian.

The second part of Figure 7.1 draws the relative speed for 17665 negotiations. It is seen that the proportion for PRCEA is best, while PID3 is the second best and higher than PBayesian (nearly $0.77 > 0.72$). This is an interesting result for both PID3 and PBayesian. From Table 7.8, we know that on average the number of interactions required

to complete a negotiation successfully is less in PBayesian than that in PID3. However, in Figure 7.1, we see that the relative speed of PID3 is higher than PBayesian. We observe that this stems from the fact that sometimes PBayesian completes a negotiation much earlier than PID3—the difference between the number of interactions between PID3 and PBayesian is significantly higher for this case. Next, we investigate how fast the producer agent realizes that it cannot provide a service satisfying the consumer.

7.5.3. Duration to detect failure

Table 7.9 shows the average number of interactions needed to decide to end a negotiation in the case of the inventory does not have any desired service for the consumer. Thus, the table only shows cases when an inventory does not contain services that will satisfy the consumer's preferences. Both PRCEA and PBayesian end the negotiation in a reasonable time, without showing the user all possible services in their inventories. Compared to PRCEA and PBayesian, PID3 needs more interactions to complete the negotiation. As a result, PRCEA can facilitate faster negotiation of service descriptions. If no consensus can be found, PRCEA and PBayesian signal this much earlier than PID3.

Table 7.9. Case-2: Average number of interactions when inventories do not have the desired service.

Consumer's profile	PRCEA	PID3	PBayesian
1	12.70	26.43	24.70
2	12.89	35.43	21.81
4	33.27	30.45	26.83
6	33.09	34.68	26.21
9	36.60	42.67	27.97
10	29.30	30.20	31.10
17	27.24	36.73	26.27
19	40.40	44.10	29.00
20	22.83	29.03	22.60
Average:	27.60	34.41	26.28

As a result, our experimental results show that the producer applying RCEA negotiates faster and more successfully compared to other learning approaches such as DCEA, naive Bayes'classifiers and ID3. That is, RCEA predicts the opponent's preferences to facilitate fast and effective negotiation of services. If no consensus can be found, RCEA signals this much earlier than DCEA, Bayes' classifier, and ID3.

8. RELATED WORK AND CONCLUSION

In this section, we review related work from the literature. We first overview a variety of negotiation approaches in computer science and then review a variety of ranking approaches for alternatives when we have partial preference ordering of alternatives as CP-nets or CP-net-like models.

8.1. Negotiation Approaches

Faratin, Sierra, and Jennings study multi-issue service negotiation among many parties and present a number of tactics and strategies for agents in order to generate their offers and evaluate counter-offers [21]. Their negotiation model is based on Raiffa's model [11] for bilateral negotiation in which two agents negotiate over multi issues. According to this model, for each issue x, there is a lower and upper bound value called $min(x)$ and $max(x)$. Each agent has a scoring function assigning a score to a particular value of an issue. This value is a real number between zero and one. Since the importance degree of each issue may vary, a weight value is defined for each issue. The weighted sum of score values for each issue constitutes the additive scoring function for the agent. This value can be considered as the utility of an offer. The scoring functions are used to generate the agent's offers and evaluate the counter-offers. Hence, similar to our approach, the negotiating agents do not know other agents' preferences represented as additive scoring functions in their model. In their setting, these scoring functions are defined as either monotonically increasing or monotonically decreasing.

They develop three kinds of tactics in this study. These are:

- Time-dependent: In time-dependent tactics, each agent has a deadline and the agent's behavior changes with respect to the time. That is, when the deadline becomes closer, the agents have a tendency to concede more quickly. The score of each issue value in an offer is calculated with a time-dependent function considering the range for the values (min, max) as well as the time. They consider two families of functions deciding the amount of concession with respect to time: polynomial

and exponential. These functions are parameterized by a value β and can be categorized according to the value of β as *Boulware tactics* and *Conceder tactics*. In Boulware tactics, the value of β is less than one; thus, the value of the function increases/decreases more slowly. This leads the agent to concede more slowly. In Conceder tactics, the value of β is greater than one. In contrast to Boulware tactics, the function concedes faster. As a result the agent approximates the lowest value that can be accepted more quickly when it uses Conceder tactics compared to the agents using Boulware tactics.

- Resource-dependent: In resource-dependent tactics, the quantity of resource has a significant impact on the negotiation. The functions take into account the amount of resource instead of time. Note that time-dependent tactics can be considered as resource-dependent tactics where the resource is time.
- Imitative: In imitative tactics, the agent imitates the opponent agent's behavior. The degrees of imitation may vary. It can be in absolute terms, proportionally and average proportionally. To illustrate this, consider an agent taking into account its opponent's previous offer in order to generate its offer. Assume that the opponent agent increases the price by 50 units. The imitating agent can decrease the price by 50 units by applying absolute terms. Another alternative is to imitate the other agent proportionally. In this approach, the imitating agents take into account the changes of its opponent's behavior in a number of previous steps. The agent can apply average proportional imitation. In this case, there is a window size. The average of changes within the window is considered to generate the offer. For example, if the window size is equal to two, the average of the changes between the opponent's last offer and the offer before the last offer are taken into account.

In their study, the agent evaluates its opponent's offer with respect to its own scoring function. If the scoring value of the opponent's offer is higher than the offer that the agent will do, the agent accepts its opponent's offer. Otherwise, the agent makes its offer.

Jonker and Treur present a generic bilateral negotiation model for multi-attribute negotiation [58]. In this model, the evaluation of the bids are performed in two levels: the attribute level and the bid level. That is, the process of generating a bid involves

considering both the overall utility of the bid and the utility of each attribute separately. While estimating the target utility of the next bid, the agent takes into account the utility of its previous bids and determines the amount of concession by applying a formula on the basis of the difference between the utility of the agent's bid and the utility of the other agent's bid. Further, the amount of the concession is parameterized by a concession factor. Therefore, the concession is performed in a controlled way. After determining the target utility of the next bid, the agent distributes this utility over the attributes. Then, the agent decides the attribute values giving the target attribute utility and generates its next bid accordingly.

To evaluate the attributes, the agent uses its user's evaluation functions. For continuous attributes, a specific type of function such as a linear function can be used while for discrete attributes, a table form evaluation is used. The overall utility of a bid is estimated by the weighted sum of the attribute evaluation values. In the developed system, the agent's preferences are kept as private as possible. The system is able to communicate with the user to ask for adjustment of the negotiation parameters such as the concession factor. Also, it informs the user if the agent finds an agreement or has a failure.

In above studies, the negotiating agent considers only its own preferences while generating its offers, but a negotiating agent may improve the negotiation process by taking into account other agents' offers in order to generate well-targeted offers. To do this, a trade-off strategy might be employed by the agent. Faratin, Sierra, and Jennings present a similarity-based trade-off negotiation strategy [34]. The main idea behind this strategy is that the importance of the issues may be different for the negotiating parties so that a number of trade-offs increasing the social welfare can be made. That is, a negotiating party demands more on some issues while concedes on other issues without changing its overall utility if possible. For instance, a consumer agent may pay higher price in order to have an earlier delivery.

Similar to their previous study [21], each agent has a weighted scoring function, and each agent only knows its own preferences. The negotiating agents use their own previous counter-offers and the opponent's last offer in order to generate their new offer. The proposed heuristic model employs a hill-climbing exploration for finding possibly

acceptable offers that assumes that agents have linear utility functions. A set of random contracts are generated having the same utility with the utility of the agent's previous counter-offer. For example, if the score of the agent's previous counter offer is 0.90, the utility of these randomly generated contracts will be 0.90. If it is not possible to generate contracts with the specified utility, the agent concedes. Among these randomly generated contracts, the agent chooses the most similar counter-offer to the opponent's last offer.

For similarity estimation, they use a fuzzy similarity functions. In detail, there are several criteria functions to estimate the fuzzy similarity for an issue. For instance, there are three criteria functions such as temperature, luminosity and visibility for the color issue. The fuzzy similarity is estimated by using a weighted sum of the corresponding values for each function. The opponent's importance weight vector might be known by the agent or the agent may only know the order of importance, not the exact values. Alternatively, the weight values might be taken from a normal distribution or undifferentiated weights might be used. As specified before, the estimated similarity is used to find the most similar counter-offer to the opponent's last offer. While Faratin, Sierra, and Jennings consider only the opponent agent's last offer, our proposed approach based on preference prediction (see Chapter 6) does not only consider the opponent's last offer but also takes into account the opponent's previous offers during the negotiation. In short, the producer agent in our approach considers both its own preferences and the consumer's predicted preferences. Thus, it chooses the services whose utility is the highest for itself among the service offers that would possibly be acceptable for the consumer. Among the candidate offers, the producer agent offers the service whose similarity is the highest to the consumer's predicted preferences. To do this, we use the RP similarity metric instead of fuzzy similarity functions.

Jonker, Robu and Treur incorporate a guessing heuristic into theie previous work [58] in order to generate better concessions [24]. In order to generate its next bid, the agent determines the target utility of the next bid similarly but in this case it also considers the opponent's weights as well as its own weights in order to distribute this target utility over the attributes. In the proposed architecture, the negotiating agents are free to reveal their preferences for some attributes while keeping some private. They develop a guessing heuristic using the history of the opponent's bids for predicting the opponent's unknown

weights.

The guessing heuristic is based on attribute value distance between the opponent's bids. For each unknown weight, the distance between the values of attributes in two successive bids are estimated. For instance, assume that the ordering of attributes are `very good` $\succ$ `good` $\succ$ `standard` $\succ$ `bad`, so the distance between `very good` and `standard` is estimated as 2. After estimating the attribute value distance, a mapping from the distance to a coefficient for the unknown weight is done. They determine the coefficients experimentally. For example, if the distance is 2, the corresponding coefficient is defined as 3. With respect to the estimated coefficients, the unknown weights are assigned. While they focus on predicting only the other agent's unknown weights, we focus on predicting a generic model for the other agent's preferences that is used for avoiding unacceptable offers for the opponent and selecting offers, which are more likely to be acceptable for the opponent.

To generate well-targeted offers leading to successful negotiations, a negotiating agent needs to understand other agent's needs as well as its own needs. For understanding other agent's needs, a variety of learning and modeling techniques are used in the literature. We review some leading approaches.

Hindriks and Tykhonov study how a negotiating agent learns the model of the opponent's preferences by using the bids exchanges in a multi-issue negotiation [23]. The presented learning algorithm is based on Bayesian learning and requires some assumptions about the opponent's preferences and even the opponent's strategy. They assume that the opponent's preferences are represented as the weighted sum of the evaluation functions, linearly additive functions. Since the aim is to learn both evaluation functions and weights for each issue, they define a set of hypotheses about the opponent's weights and define a hypothesis space that consists of predefined evaluation functions. They assume that the evaluation functions can be one of these shapes: downhill, uphill, and triangular. Each hypothesis is associated with a probability distribution. If prior knowledge does not exist, a uniform distribution is used for the prior probabilities.

The aim is to estimate the probability that the particular hypothesis would be the evaluation function/issue weight given the opponent's bid. To do this, the agent needs to know the utility of the bid for the opponent. Therefore, the opponent's strategy is assumed to be a concession-based strategy in which the utility of the first bid is the highest and the utility of the next bids is decreasing monotonically. Consequently, the predicted utility of the opponent's bid might be used. The probabilities are updated after opponent's each bid by using the Bayes' rule.

While negotiating with the opponent agent, the agent estimates the utility of its candidate counter-offers with respect to the opponent's predicted preferences and proposes the bid whose estimated utility for the opponent is the highest. As far as the open and distributed environments are concerned, the above assumptions may become inappropriate. Therefore, our approach is to predict the opponent's preferences in a generic way without making assumptions about the opponent's preference representation and strategy. While Hindriks and Tykhonov model the opponent's preferences as a linear additive function, we predict a set of concepts that are more likely to be accepted by the opponent and the negotiating agent proposes the most similar offer to the predicted preferences.

Coehoorn and Jennings extend Faratin's trade-off strategy [34] by incorporating the prediction of the opponent's weights [35]. In the proposed negotiation model, each agent has a weighted scoring function and a range of acceptable values for each issue: a lower and an upper reservation value. Further, each agent has a deadline to complete the negotiation and employs a time-dependent negotiation strategy. For predicting the opponent's weights, they apply a non-parametric statistical method, namely kernel density estimation. Their intuition is that the difference between the opponent's last offers may have a relation with the opponent's weights. For example, if the distance is relatively small for a particular issue at the beginning of the negotiation and is relatively large at the end of the negotiation, it might be inferred that this issue may be important for the opponent so that it has a tendency to concede slowly as possible on this issue .

For kernel density estimation, they use the offers and counter-offers proposed in the past negotiations. The kernel for each issue weight has three dimensions: the predicted

weight, the difference between two consecutive offers and the probability density of this weight given the difference. The estimated weights are used to generate well-targeted trade-offs. That is, the agent finds a set of candidate offers whose utilities are the same as its previous offer. Among these candidate offers, the most similar offer to the opponent's offer is chosen as the current offer. For similarity estimation, a weighted fuzzy similarity is used where the weights are estimated by the kernel density estimation. While Coehoorn and Jennings use the offers and counter-offers in the past negotiations, in our framework the agent only considers the offers and counter-offers in the ongoing negotiation. That is, we do not keep the history of the past negotiations. It would be interesting to use the data gained from the past negotiations in order to improve the negotiation process. Moreover, our aim is to predict a generic model to generate offers that are more likely to be acceptable by the opponent rather than predicting the opponent's weights. Unlike Coehoorn and Jennnings, we do not assume that the opponent employs a time-dependent strategy.

Lau *et al.* present an evolutionary learning approach for generating offers in multi-issue negotiation [26]. They develop a genetic algorithm in which the candidate offers are represented by the chromosomes. Similar to the trade-off approach [34], their approach also considers both the negotiating agent's own utility and the opponent's last counter-offer. The aim is to maximize the utility of the negotiating agent while minimizing the distance between the candidate offer and the opponent's last counter-offer. Accordingly, the fitness value of each chromosome is estimated by a fitness function. The agent's attitude in terms of the extent of being selfishness and benevolence is controlled by a parameter. Further, the effect of time pressure is reflected by another parameter. The fittest chromosome representing the candidate offer is put forward to the opponent. Contrary to their approach, we do not only consider the opponent's counter-offer but also take into account the offers that are rejected by the opponent in order to predict the opponent's preferences. Accordingly, the agent predicting its opponent's preferences avoids to propose an offer that is possibly rejectable by the opponent.

The mentioned studies above assume that the agent has an evaluation function for evaluating the outcomes in terms of utility. Unlike these studies, Luo *et al.* propose a prioritized fuzzy constraint based negotiation model in which the buyer agent specifies

its offers by using constraints [59]. According to the proposed model, the agent reveals its partial preference information (some parts of its constraints) in a minimal fashion. That is, the buyer agent sends its constraints to the seller. The seller agent checks the constraints and if it cannot satisfy this constraints, it requests the buyer to relax them. In our model, the consumer does not reveal its partial preferences. Further, the producer agent tries to predict what the consumer agent would like if its initial request cannot be satisfied. Hence, the challenge in our study is to find out a model for the consumer's private preferences.

The following approach deals with aspects that are complementary to our work. In order to shorten the negotiation time for better agreements, Ramchurn *et al.* present a negotiation algorithm incorporating the use of rewards and extend Rubinstein's alternative offer protocol [42] by allowing the exchange of the arguments [60]. In addition to proposing offers and counter-offers, the negotiating agent is able to promise a reward or ask for a reward in their work. They specifically consider the repetitive negotiation games in which the negotiating parties meet each other in a new negotiation after completing the current negotiation. In this setting, if an agent promises a reward to its opponent, it means that in the next negotiation it concedes on a particular issue more. Consider that a buyer and a seller. The seller may convince the buyer to buy his product with a particular price promising a reward such as a discount in the next time. Or the buyer may ask for a reward to accept the seller's offer.

Similar to the previous studies [21, 34], the negotiating agents have a utility function for each issue. The utility of an offer is estimated by a weighted sum of these utilities. Similarly, an upper and a lower bound for acceptable values are defined for each issue. Each agent has a deadline and the utility of a negotiation outcome is decreased over time by a discounting factor. If the agents do not have a consensus before the deadline, they cannot start the next negotiation. They present a reasoning mechanism for a reward generation that consists of three steps: computing concession degrees, determining rewards and sending offer and rewards. When the agent concedes nearly equal to its opponent, there is no need for a reward. When the agent concedes more than its opponent, the agent may ask for a reward. The agent promises a reward if it concedes less than its opponent. They also propose a reward-based tactic based on Faratin's trade-off strategy [34]. To

sum up, Ramchurn *et al.* show that their negotiation tactic based on the generation of rewards improves the utility of the deals. In our work, we do not consider rewards. We concentrate on constructing an approximation to the consumer's preferences. By eliminating the offers possibly to be rejected by the consumer, we increase the chance of generating a mutually acceptable offer.

Rahwan *et al.* study argumentation-based negotiation approaches on the basis of exchanging meta-information about negotiation [61]. By means of arguments such as promise, threat, justification of a proposal and so on, an agent may have an influence on the other agent's states. With the help of arguments, the probability of the other agent's acceptance of offer is expected to increase. For instance, the agent can generate an argument explaining its aim. The other agent may offer an alternative proposal helping the agent reach its goal. Our negotiation model does not involve arguments but it would be interesting to extend our model by adding arguments.

8.2. Ranking Approaches

Rossi *et al.* extend CP-nets to capture multiple agents' qualitative preferences and present mCP-nets [62]. In mCP-nets, there exists a partial CP-net for each individual agent. In contrast to CP-nets, partial CP-nets allow users not to specify the ordering of the values of some particular attributes. This is necessary to be able to represent preferences in mCP-nets such as "My preference on x depends on other agent's preference on x". Thus, we have additional flips such as indifferent and incomparable flips in addition to improving/worsening flips. When there is a chain of worsening and indifferent flips and at least one worsening flip from α and β, it is said that α is preferred over β. Unlike CP-nets, there may be more than one optimal outcome in an acyclic partial CP-net because of the unranked attributes.

In that study, several voting methods such as Pareto, Majority, Max, Lex and Rank [62] are proposed to aggregate agents' preferences and to determine whether an outcome is preferred over another for those agents. Among these semantics, only *Rank* removes incomparability. That is, there may be some incomparable outcomes with respect to other semantics except *Rank*. When we apply *Rank* method, each agent ranks each

outcome by estimating the length of the shortest sequence of worsening flips between that outcome and one of the optimal outcomes. When we compare two outcomes, we sum up each agent's rank for these outcomes. To be able to say that an outcome α is preferred over β, the sum of ranks assigned to α should be smaller than the sum of ranks for β.

Unlike that study, we do not address the problem of capturing and handling multiple agents' qualitative preferences. Instead, we focus on how an agent reasons on its user's qualitative preferences represented as an acyclic CP-net and negotiates accordingly with an opponent agent. In our depth based heuristic, we rank an outcome by comparing it to the worst outcome generated with the longest sequence of improving flips. Rossi *et al.*, on the other hand, consider the shortest sequence of worsening flips to find the optimal outcome.

We briefly examine the differences in our and their rankings by considering the CP-net explained in Figure 4.1 and the preference graph induced from this CP-net depicted in Figure 4.2 (See Chapter 4). According to their *Rank* method, <`Sea`, `Winter`, `Two weeks`> is preferred over <`Historical Places`, `Winter`, `One week`> since the length of the shortest sequence of worsening flips between the optimal outcome, <`Sea`, `Summer`, `Two weeks`), and <`Sea`, `Winter`, `Two weeks`> is equal to one while that between the optimal outcome and <`Historical Places`, `Winter`, `One week`> is equal to three. Recall that the smaller rank is accepted as more preferable with respect to *Rank* method. However, the length of the longest sequence of improving flips from the worst outcome to the current outcome is the same for both outcomes. In other words, they are at the same depth (five). Therefore, these two outcomes are equally preferred with respect to our depth heuristic.

Li *et al.* also study the problem of collective decision making with CP-nets [63]. The aim of that study is to find a Pareto-optimal outcome when agents' preferences represented by CP-nets. They firstly generate candidate outcomes to increase the computational efficiency instead of using the entire outcome space. Then, each agent in the system ranks these candidate outcomes according to their own CP-nets. For ranking an outcome, they use the longest path between the optimal outcome and that outcome in the induced preference graph. Thus, the minimum rank is desired for the agents. They

choose the final outcome for the agents by minimizing the maximum rank of the agents.

While they propose a procedure for collective decision making with the aim of choosing one outcome for multiple agents, we focus on estimating utility values of each outcome that will be used during the negotiation for an individual agent. Both their ranking approach and our depth heuristic give the same ordering of outcomes.

Rossi and Venable propose two new ways of reasoning with CP-nets: applying new semantics for CP-nets and approximating CP-nets with soft constraint satisfaction problem [64]. Similar to our approach, the new semantics provide a total order while the original CP-net semantics provides a partial one. The proposed approaches in their study takes the hierarchy level of each attribute in CP-nets into account while comparing outcomes. The attributes are ordered with respect to their dependencies where the independent attributes take the first places. According to the first semantics, α is preferred over β if α dominates β first lexicographically. To illustrate this, consider we have two attributes R and S and our CP-net says $r_1 \succ r_0$, $r_1 : s_1 \succ s_0$ and $r_0 : s_0 \succ s_1$. Note that R is in the first level where S is the second level. When we compare two outcomes: $< r_1$, $s_0 >$ and $< r_0$, $s_0 >$, we say that the former outcome is preferred over the latter outcome because it has a better value (r_1) for the first attribute (implicitly more important than the second attribute).

In the second semantics a so-called *majority lexicographical (ml) ordering*, the outcome dominating on the majority of the important attributes (that are in the higher hierarchical level) is more preferred. To illustrate this, assume that we have five attributes R, S, X, Y and Z and a CP-Net saying that $r_1 > r_0$, $s_1 > s_0$, $s_1 : x_1 > x_0$, $s_0 : x_0 > x_1$, $s_1 : y_1 > y_0$, $s_0 : y_0 > y_1$, $x_1 : z_1 > z_0$ and $x_0 : z_0 > z_1$. In this CP-net, R and S are in the first level, X and Y are in the second level and Z is the third level. According to ML ordering, the outcome $< r_1, s_0, x_0, y_0, z_1 >$ dominates $< r_0, s_1, x_1, y_0, z_1 >$. Since the former outcome wins on R and the latter outcome wins on S, they tie on the first level. However, in the second level the former outcome wins on both X and Y while the latter outcome only wins on X. Since $< r_1, s_0, x_0, y_0, z_1 >$ wins on the majority in the second level, it dominates $< r_0, s_1, x_1, y_0, z_1 >$. Note that we do not need to check for the attribute in the third level. In our study, we use the induced preference graph to

find the estimated utilities of the outcomes and compare outcomes with respect to these estimated utilities while they use the hierarchical level of CP-net directly to compare two outcomes.

Chalamish and Kraus presents an automated mediator for bilateral negotiations in which agents share their qualitative preferences only with the mediator [65]. The aim of the mediator is to suggest acceptable agreements for both participants when it is necessary to speed up negotiations. In the proposed system, each agent represents its user's preferences as a Weighted CP-net, an extension of CP-net with weighted importance table indicating the relative importance of attributes. After both agents send their WCP-nets to the mediator, the mediator sorts all possible outcomes with respect to agents' preferences separately — resulting in two sorted list of outcomes. While sorting outcomes, the mediator uses an enhanced version of majority lexicographical (ML) ordering [64] with weights. The weighted version of ML presented by Chalamish and Kraus is based on the comparison of weighted sum that the outcome gets for its assignment of each attribute. For instance, assume that for X attribute it is specified that $x_1 > x_0$, which means x_1 is preferred over x_0 and the relative importance of X is expressed with $riw(X)$. If the outcome involves x_1, it gets $riw(X)$ while it gets zero if it includes x_0. When we compare two outcomes, α and β, if the summation for α is higher than that for β, it is said that α is preferred over β. To sum up, the mediator sorts outcomes with this metric and recommends Pareto-optimal outcomes.

While the mediator uses other agents' WCP-Nets to suggest Pareto-optimal outcomes to the negotiating agents, in our study the agent tries to derive estimated utility values from its own CP-Net by applying one of the proposed heuristics. These estimated utility values will be employed by the agent's negotiation strategy while negotiating with the opponent agent. Since we do not have relative importance weights, it is hard to compare our approaches with theirs directly. However, it can be said that it is straightforward to apply their approach in binary WCP-nets but it is not clear how the rank of each outcome will be estimated when the attributes have more than two values.

Gérard *et al.* present a probabilistic ranking approach based on minimal specificity principle [9]. The aim of this study is to rank alternatives that are represented as a vector

of qualitative criteria evaluations rather than issue values. Note that they use the same linearly ordered scale of satisfaction levels for each criterion. For instance, consider that we would like to compare students with respect to their grades for two courses, namely *Math* and *Literature*. These two criteria can be evaluated on a scale like $a \succ b \succ c$ (See Example 2 in [9]). For this example, the alternatives that are needed to be ranked are vectors of evaluations such as $< a, b >$, $< a, a >$, $< c, b >$ and so on. In that study, the authors propose a qualitative approach to find a complete preorder satisfying all constraints that are partially specified. That is, they have a partial preorder and their aim is to find a complete preorder as little arbitrary as possible. Note that a constraint can be a preference statement like "$< a, b >$ is preferred over $< b, a >$" or a generic preference such as Pareto ordering or a generic rule like "Math is more important than Literature in that school". The proposed approach produces a possibility distribution with respect to the given constraints. This distribution provides a complete preorder between alternatives. According to minimal specificity principle, an alternative that is dominated by less alternatives would take the higher possible degree. According to their algorithm, the most preferred alternatives are computed first and the set of alternatives that are only dominated by those are selected and so on. It is worth noticing that their approach does not aggregate the vectors of criteria evaluations unlike some numerical aggregation approaches such as Choquet integrals [66].

In our study we propose several heuristics that take a partial ordering of outcomes represented via a preference graph induced from a given CP-net and estimate utility values resulting in a total ordering of outcomes. Unlike their study, in our study each outcome represents a vector of issue values rather than criteria evaluations. We briefly compare some of our heuristics with their ranking approach based on minimal specificity. For this purpose, we can use Example 2 in their study. Note that although our heuristics are developed for the preference graphs induced from CP-nets, they can be applied to any preference graphs showing preference ordering between outcomes. Figure 8.1 depicts the preference graph showing partial ordering for this example. Note that if there exists an edge from x to y in the graph, that means y is preferred over x. When we apply their approach, we get $\{< a, a >\} \succ \{< a, b >\} \succ \{< b, a >, < a, c >\} \succ \{< c, a >, < b, b >\} \succ \{< b, c >\} \succ \{< c, b >\} \succ \{< c, c >\}$ where $< a, a >$ is the best outcome and cc is the worst outcome. On the other hand, when we apply our depth heuristic,

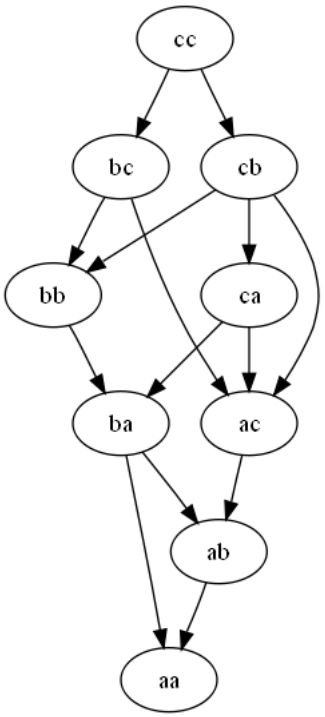


Figure 8.1. Partial ordering for Example 2 [9].

the order will be like $\{< a, a >\} > \{< a, b >\} \succ \{< b, a >, < a, c >\} \succ \{< c, a >, < b, b >\} \succ \{< b, c >, < c, b >\} \succ \{< c, c >\}$. Since $< b, c >$ and $< c, b >$ are at the same depth, we say that they are equally preferred according to depth heuristic. In contrast to our estimated ordering, $< b, c >$ is preferred over $< c, b >$ with respect to their ranking approach. It would be interesting to examine how their rankings would apply in the context of negotiation.

When we compare their approach with our *Preferred Outcomes Heuristic*(POH), it can be said that unlike their approach POH does not consider only the number of outcomes which dominates the current node but also take their importance into account. According to this approach, the order will be like $\{< a, a >\} \succ \{< a, b >\} \succ \{< b, a >, < a, c >\} \succ \{< b, b >\} \succ \{< c, a >\} \succ \{< b, c >\} \succ \{< c, b >\} \succ \{< c, c >\}$. It can be seen that $< b, b >$ is preferred over $< c, a >$ with respect to our heuristic while according to their approach they are equally preferred.

8.3. Discussion of Contributions and Future Directions

This thesis mainly pursues the following research challenges:

- How does a negotiating agent reason about its user's qualitative preferences represented as CP-nets and negotiates effectively on behalf of its users? (Chapter 4)
- How does a negotiating agent use the existing negotiation strategies based on utilities while its user's preferences are represented qualitatively as CP-nets? (Chapter 4)
- How does a negotiating agent revise its offer and generate well-targeted offers that are more likely to be acceptable by the other agent? (Chapter 6)
- To accomplish the above objective, how does an agent predict the other agent's preferences from the bids exchanged during the negotiation? (Chapter 6)

Even though CP-nets are natural and intuitive for representing human users' preferences, they can mostly provide a partial ordering of the outcomes, so we are unable to compare some outcomes with respect to a given CP-net. This might be a reason why CP-nets are not ordinarily used in negotiation. This thesis develops a number of heuristics for CP-nets to estimate the utilities of the outcomes. In other words, we estimate a total ordering of the outcomes from a partial ordering by applying a heuristic on the preference graph induced from a given CP-net. Consequently, the estimated utilities enable us to compare all outcome pairs.

The developed negotiation approaches can be applied in a variety of real life problems requiring group decision. Consider the following scenario. Melisa lives in Istanbul and needs to arrange a phone meeting with Jack living in New York. However, there are some conflicts in their preferences on the meeting date. For example, Melisa prefers a meeting on Wednesday over a meeting on Monday while Jack prefers to meet on Monday rather than to meet on Wednesday. Thus, they need to negotiate and find a mutually acceptable date. Since reaching a consensus on the meeting date might be time consuming, they may not be willing to negotiate via emails. A possible solution might be to adopt an agent associated with each of their calendar applications. To do this, each agent needs to elicit its user's preferences on the meeting date. Here, Melisa and Jack specify their preferences qualitatively as CP-nets.

For common understanding of the offers, the agents use a shared ontology describing the negotiating issues and their possible values. For simplicity, we have two issues:

day and time slot. Possible meeting days are `Monday`, `Wednesday`, and `Friday`. There are three possible time slots shown in Table 8.1. Since there is a seven-hour time difference between Istanbul and New York, the agents use a predefined generic slots such as `Slot A`, `Slot B`, and `Slot C`. Both users specify their preferences on time slot with respect to these slots while considering the corresponding time slots in their own cities.

Table 8.1. Time slots for the meeting date.

Time Slot	**Time slot for Istanbul**	**Time slot for New York**
Slot A	10:00 – 11:00 a.m.	03.00 – 04:00 a.m.
Slot B	16:00 – 17:00 p.m.	09:00 – 10:00 a.m.
Slot C	20:00 – 21:00 p.m.	13:00 – 14:00 p.m

Figure 8.2 shows the preferences on the meeting for both Melisa and Jack. Melisa's preferences on time slot depends on the day. For example, when the meeting is on Monday, she prefers a meeting in `slot B` over a meeting in `slot A` and prefers a meeting in `slot A` over a meeting in `slot C` (`Slot B` > `Slot A` > `Slot C`). On the other hand, Jack's preferences on day and time slot are independent from each other. Irrespective of on which day the meeting is, Jack prefers `Slot C` over `Slot B` and over `Slot A`.

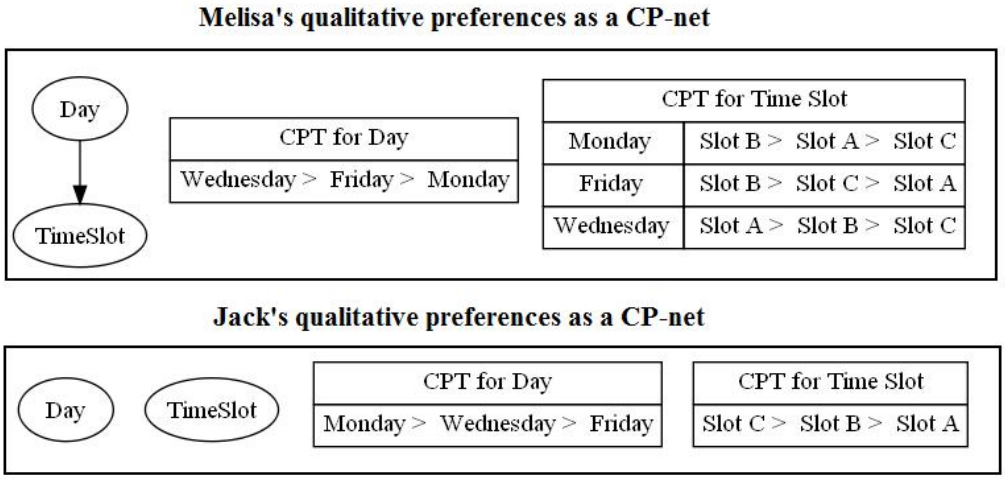


Figure 8.2. Melisa's and Jack's CP-nets.

After elicitation of the preferences, each agent induces a preferences graph from its user's CP-net and applies one of the proposed heuristics to the induced preference graph in order to generate estimated utilities of the outcomes. Figure 8.3 and Figure 8.4 draw

the preference graphs induced from Melisa's CP-net and Jack's CP-net, respectively. In our scenario, both agents apply the Depth Heuristic and employ simple concession-based negotiation strategies. Table 8.2 shows the bid exchanges between the agents. Each agent starts with the offer whose estimated utility is the highest for itself and concedes over time. The outcome <`Wednesday, Slot A`> is the most desired date whose depth is the highest in the preference graph induced from Melisa's CP-net. Thus, Melisa's agent first asks for <`Wednesday, Slot A`>. Jack's agent makes a counter-offer <`Monday, Slot C`>, which is the most desired date for Jack. Then, Melisa's agent concedes and generates an offer from the second highest depth (5), <`Wednesday, Slot B`>. Similarly, Jack's agent makes its counter-offer as <`Monday, Slot B`>. When Melisa's agent offers <`Wednesday, Slot C`>, Jack's agent accepts this offer since its depth is equal to that of Jack's previous counter-offer and the negotiation is over. Then, the agents inform Melisa and Jack about the agreed meeting time, <`Wednesday, Slot C`>.

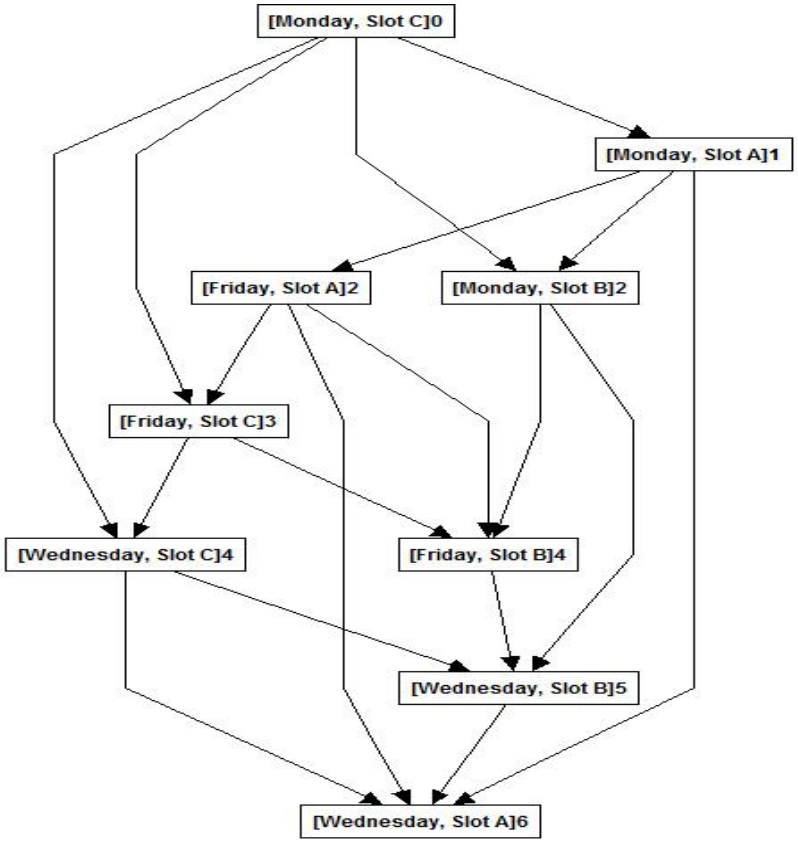


Figure 8.3. The preference graph induced from Melisa's CP-net.

As seen from the above negotiation scenario, the proposed automated tools facilitate the daily-life activities for human users. Furthermore, we evaluate the performance of

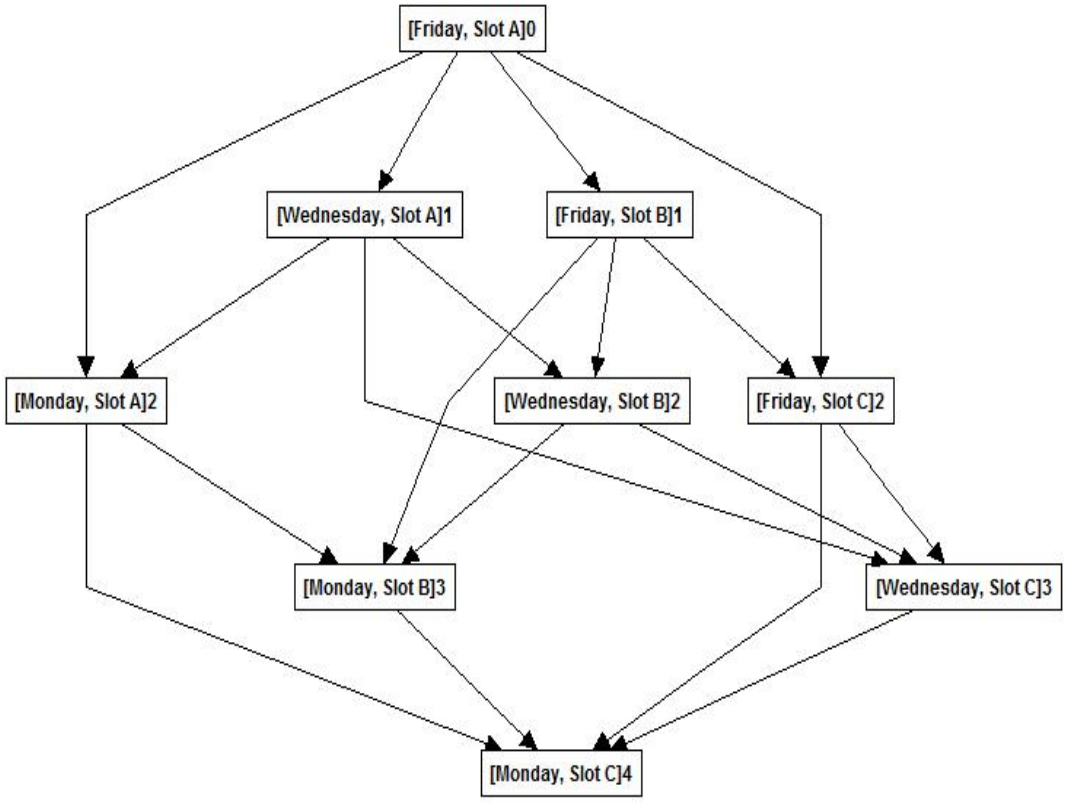


Figure 8.4. The preference graph induced from Jack's CP-net.

Table 8.2. Negotiation between Alice's agent and Jack's agent.

Offer	**<Day, Time slot>**	**Utility for Melisa**	**Utility for Jack**
Melisa's Agent:	<Wednesday, Slot A>	1.0	0.25
Jack's Agent:	<Monday, Slot C>	0.0	1.0
Melisa's Agent:	<Wednesday, Slot B>	0.83	0.5
Jack's Agent:	<Monday, Slot B>	0.33	0.75
Melisa's Agent:	<Wednesday, Slot C>	0.67	0.75

the proposed heuristics in a realistic negotiation setting. For this purpose, a negotiation platform, GENIUS is extended. Our extension allows us to elicit both the user's real total ordering represented as a UCP-net and the user's partial ordering represented as a CP-net. The negotiating agent having a CP-net applies one of the proposed heuristics in order to estimate the utilities. Thus, the negotiating agent is able to employ a negotiation strategy based on utilities. While the agent negotiates with the estimated utilities, the negotiation outcome is evaluated with respect to the user's real total ordering, specifically with respect to the utilities gained from its UCP-net. The developed system also allows comparing an agent negotiating with its user's qualitative preferences represented as a CP-net versus an agent negotiating with its user's quantitative preferences represented as a UCP-net.

Our evaluation criteria for the performance of the proposed heuristics in negotiation are the average utility of the negotiation outcomes and the number of rounds that are required to reach a consensus. As expected, the agent using its user's real total ordering, having a UCP-net, performs best with respect to overall utility gained. However, our results show that it is still admissible to elicit the user's preferences as a CP-net and to negotiate with these preferences by applying a heuristic instead of eliciting the user's preferences quantitatively. Among the proposed heuristics, Depth heuristic can be identified as the leading heuristic. Our results show that applying Depth Heuristic in negotiation yields almost identical success to using UCP-nets and achieves a superior speed in reaching consensus.

A direction for future research is studying the effect of negotiation strategies on the performance of the heuristics. In this thesis, the agent applies a simple concession-based strategy that is explained in Chapter 5. It would be interesting to apply other negotiation strategies such as a Trade-off strategy and compare their effects. It would be ideal to identify pairings of heuristics and strategies that perform well together.

Since in many negotiation settings the participants' preferences are private, the negotiating agents need to predict its opponent's preferences in order to generate well-targeted offers leading to successful negotiations. Accordingly, this thesis develops a novel preference prediction algorithm based on concept learning, namely Revisable Candidate

Elimination Algorithm for predicting a generic model about the opponent's preferences from the bid exchanges. The proposed algorithm incorporates the idea of revision. Thus, as the negotiation proceeds, the negotiating agents can revise its idea about its opponent's preferences. As humans use their common knowledge about the domain when they predict someone's private information, the developed preference prediction algorithm is able to use domain knowledge. In other words, it is enhanced with ontology reasoning.

The negotiating agent uses the predicted generic model about its opponent's preferences in order to avoid the service offers that are possibly rejected by the opponent in the negotiation and generate well-targeted offers that are more likely to be acceptable for the opponent. To do this, the opponent's offers are considered as positive examples and the agent's counter-offers that are rejected by the opponent are taken as negative examples. During the negotiation, the predicted generic model about the opponent's preferences is updated after each bid exchange. The experimental results show that the participants reach early agreement when the developed preference prediction algorithm is used. Moreover, if consensus is not possible, the negotiating agent can realize this earlier than other approaches.

In this thesis, we assume that the agents' preferences are fixed during negotiation. However, in real life people's preferences may change. For instance, a particular service outcome, which is rejected by a consumer at first, may become acceptable for the same consumer after a while. Conversely, an acceptable service outcome may become undesired by the consumer over time. Although our prediction algorithm is not designed to handle the changing preferences, it can be extended to handle such preferences by adding two new rules. First, if the new positive example exists in the negative sample set before, this sample can be removed from the negative sample set and added into the positive sample set. Since each positive sample should be covered by the general set and the specific set, our revision process would perform the necessary modifications. Second, if the new negative sample exists in the positive sample set before, it is removed from the positive set so it is not covered by any of the hypotheses anymore and it is added into the negative sample set. Since none of the negative samples should be covered by any hypotheses, our algorithm would perform the necessary updates on the hypotheses sets.

Table 8.3 illustrates how it works for the first case in a sample negotiation scenario. According to the given scenario, the consumer initiates the negotiation with a service offer <`Monday, Slot A`>. The producer agent considers this service offer as a positive sample and adds it into the positive sample set. In the second turn, the consumer agent rejects the producer's counter-offer <`Wednesday, Slot C`>. Since it is a negative sample, it is added into the negative sample set and the most general set G is specialized minimally not to cover it anymore. After a few interactions, the consumer agent changes its preferences and offers <`Wednesday, Slot C`> that was offered by the producer agent before and rejected by the consumer agent. At that moment, the current negative sample set includes two samples: <`Wednesday, Slot C`> and <`Friday, Slot C`>. According to the new rule, we remove <`Wednesday, Slot C`> from the negative sample set and add it into the positive sample set. Since current G does not cover this new positive sample, we need a revision process in which new hypotheses are generated to cover the positive sample. After this process, we generate the general hypothesis (`Wednesday, ?`) and add it into G. As seen from the given negotiation scenario, our extension may lead RCEA to handle predicting the other agent's changing preferences.

Table 8.3. A negotiation scenario for the first case.

Turn	**Offer <Day, Time slot>**
Consumer offers	<Monday, Slot A>
Learnt concept	G_1={(?, ?)} S_1= {(Monday, Slot A)}
Consumer rejects	<Wednesday, Slot C>
Learnt concept:	G_2={(Monday, ?), (?, Slot A)} S_2=S_1
Consumer offers	<Monday, Slot C>
Learnt concept:	G_3={(Monday, ?), (?, Slot A)} S_3={(Monday, ?)}
Consumer rejects	<Friday, Slot C>
Learnt concept:	G_4={(Monday, ?), (?, Slot A)} S_4=S_3
Consumer offers	<Wednesday, Slot C>
Learnt concept:	G_5={(Monday, ?), (?, Slot A), (Wednesday, ?)} S_5=S_4 $\cup$ {(Wednesday, Slot C)}

Moreover, this thesis focuses only on predicting the other agent's preferences but understanding the other agent's intension may also improve the negotiation process. Here, the intension means whether the agent's preferences are arbitrary or rational. When the other agent's preferences are arbitrarily specified, convincing the other agent might be easier than the case when the other agent is well-informed about the domain. If the agent understands the other agent's intension by analyzing the bids exchanges, it can act accordingly. For instance, it may change its negotiation strategy with respect to the predicted intension.

Another direction for future research is extending the preference prediction algorithm to be able to support predicting a group of agents' preferences rather than predicting a single agent's preferences. That might be useful when an agent needs to negotiate with a group of agents on a particular matter. Furthermore, it would be interesting to associate weights with the predicted acceptable offer descriptions. That is, some acceptable offers might be preferred more than other acceptable offers for the opponent. Thus, capturing this relation may lead to more successful negotiations.

Furthermore, in our negotiation setting the consumer and producer agents negotiate on a particular service that consists of discrete issues. However, a service description may contain continuous issues such as price. In such a case, we can define generic values such as `Low`, `Medium`, and `High` for the price in a shared service ontology. The users specify their preferences on price by considering these generic values. For example, a consumer can specify her preferences such as `Low` $>$ `Medium` $>$ `High`. Further, each agent also elicits their users' ranges, minimum and maximum values for each generic value. These ranges may vary for the agents. For example, the maximum value for `Low` is possibly different for the consumer and the producer. Although the domain values for the price seems to be discrete, this issue is defined as a continuous issue, which means that the agents should convert these values to a real number with respect to their own individual ranges and negotiation strategies. To illustrate this, consider the consumer generating an offer whose price in the range of `Medium`. If the consumer's range for `Medium` is 100 and 500, the consumer can generate any real value between 100 and 500 for the price while making its offer. The agent's negotiation strategy may dictate which real number will be used in the offer. Consequently, the agents can also negotiate on continues issues in our negotiation

framework.

The last issue is the structure of the bid. In this thesis, the structure of the bid is fixed and defined in a shared service ontology. On the other hand, under some circumstances the structure of the bid might be changed during the negotiation. It would be interesting to handle varying structure of the bid. The agents might start negotiating on a small number of issues and the structure of the bids might evolve during the negotiation. These are all interesting topics that we defer to future work.

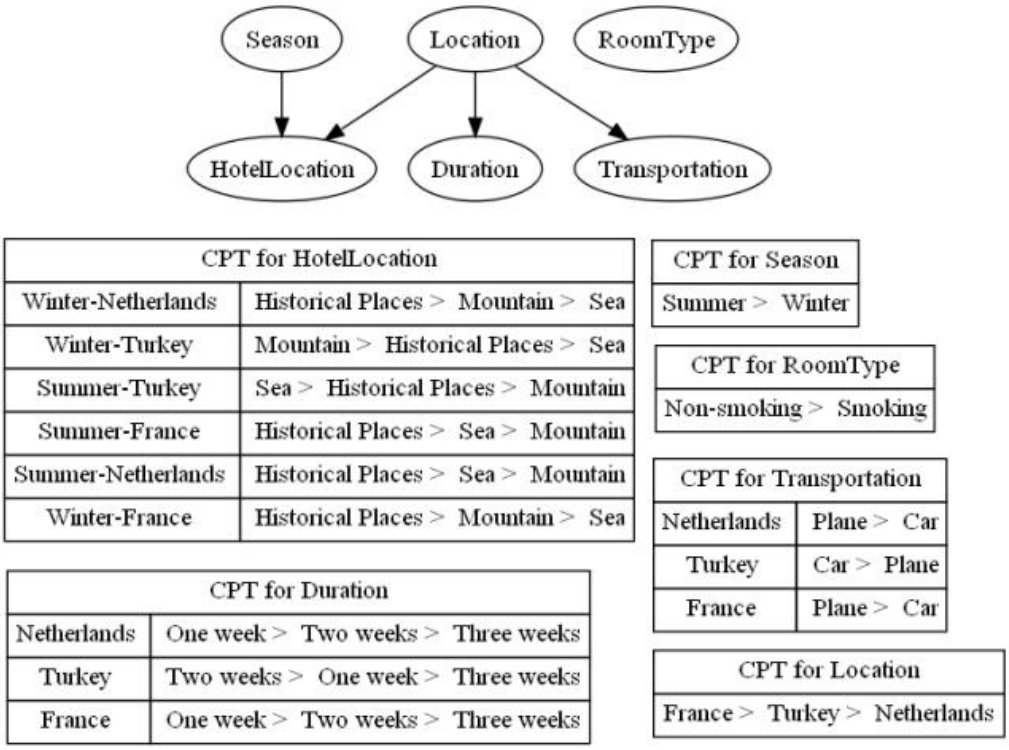
APPENDIX A: CPNETS USED IN OUR EXPERIMENTS

Table A.1 gives information about 10 users' CP-nets and induced preference graphs from these CP-nets. The second column shows how many dependencies exist in the CP-net, which is equal to the number of edges in the CP-net. Note that more dependencies express more information about user's preferences. The third column indicates the level of hierarchy — the length of the longest path between ancestor and descendant nodes. The forth column shows the number of independent nodes (not having any connections with other nodes) in the CP-net and the fifth columns shows the total number of orderings expressed in CPTs. The last column indicates the depth of induced preference graph from this CP-net.

Table A.1. Information about CP-nets and their induced preference graphs.

CP-net	**Dependency**	**Hierarchy**	**Independent Nodes**	**Orderings**	**Graph Depth**
1	1	2	4	7	10
2	5	3	1	15	16
3	4	3	2	16	16
4	4	2	1	15	18
5	4	3	1	12	18
6	3	3	2	10	13
7	3	3	1	11	14
8	5	3	1	17	22
9	4	3	1	13	21
10	5	3	1	16	19

Figure A.1 shows the forth CP-net where Figure A.2 draws the fifth CP-net used in our experiment. Figure A.3 and Figure A.4 shows the seventh and eighth CP-nets respectively.



CPT for HotelLocation	
Winter-Netherlands	Historical Places > Mountain > Sea
Winter-Turkey	Mountain > Historical Places > Sea
Summer-Turkey	Sea > Historical Places > Mountain
Summer-France	Historical Places > Sea > Mountain
Summer-Netherlands	Historical Places > Sea > Mountain
Winter-France	Historical Places > Mountain > Sea

CPT for Season
Summer > Winter

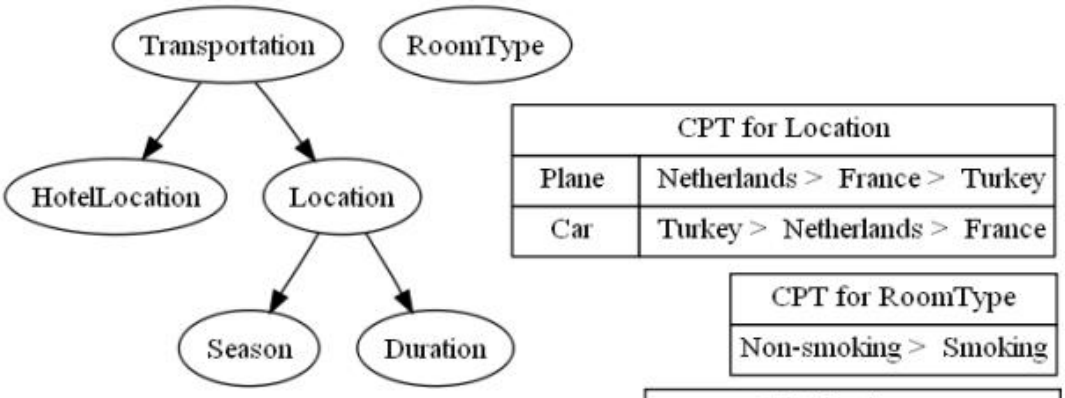
CPT for RoomType
Non-smoking > Smoking

CPT for Transportation	
Netherlands	Plane > Car
Turkey	Car > Plane
France	Plane > Car

CPT for Duration	
Netherlands	One week > Two weeks > Three weeks
Turkey	Two weeks > One week > Three weeks
France	One week > Two weeks > Three weeks

CPT for Location
France > Turkey > Netherlands

Figure A.1. The fourth CP-net used in our experiment.



CPT for Location	
Plane	Netherlands > France > Turkey
Car	Turkey > Netherlands > France

CPT for RoomType
Non-smoking > Smoking

CPT for Season	
France	Winter > Summer
Turkey	Summer > Winter
Netherlands	Summer > Winter

CPT for HotelLocation	
Car	Historical Places > Mountain > Sea
Plane	Sea > Mountain > Historical Places

CPT for Duration	
Netherlands	Two weeks > Three weeks > One week
Turkey	One week > Two weeks > Three weeks
France	One week > Two weeks > Three weeks

CPT for Transportation
Car > Plane

Figure A.2. The fifth CP-net used in our experiment.

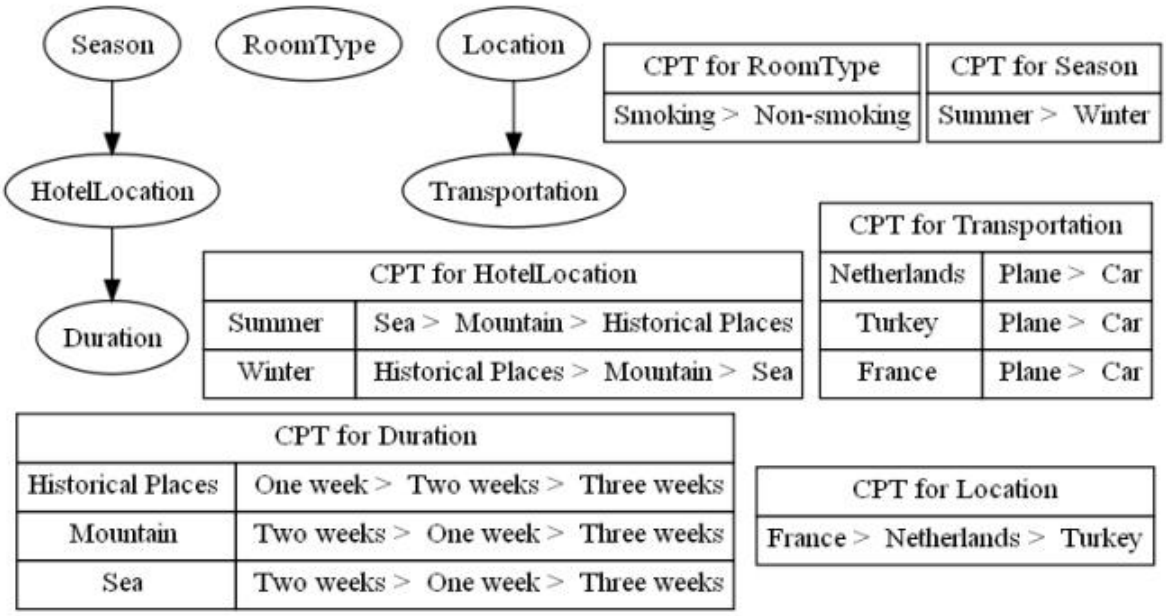


Figure A.3. The seventh CP-net used in our experiment.

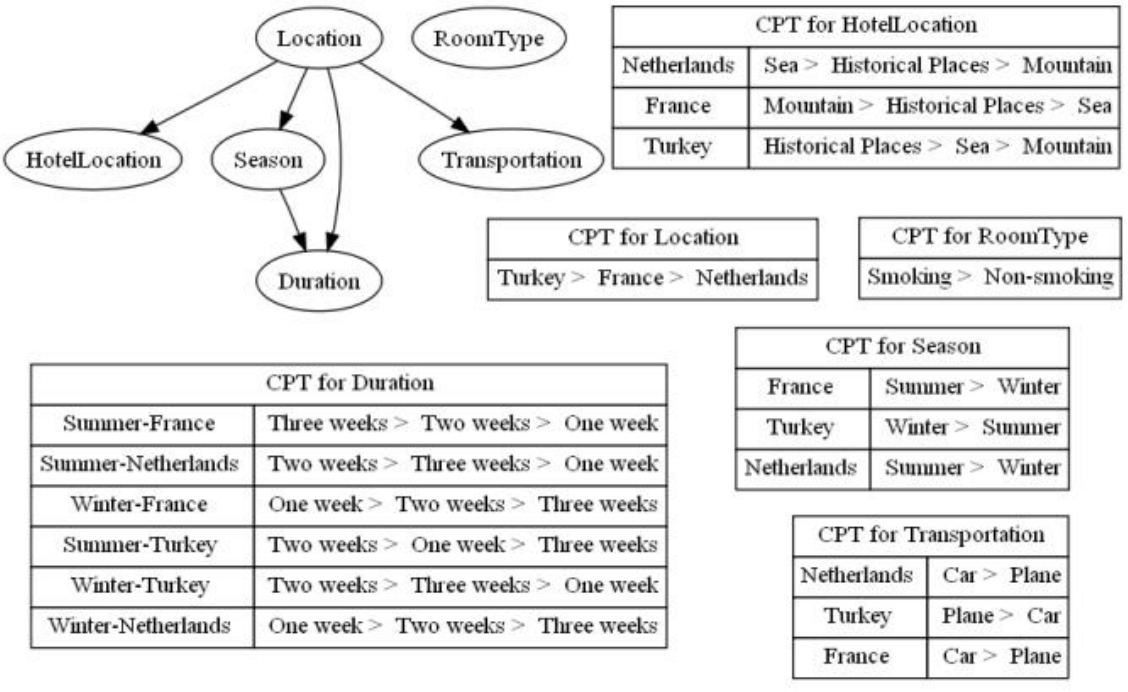


Figure A.4. The eighth CP-net used in our experiment.

REFERENCES

1. Aydoğan, R. and P. Yolum, "Learning Disjunctive Preferences for Negotiating Effectively", *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1201–1202, Budapest, Hungary, May 2009.

2. Aydoğan, R. and P. Yolum, "Ontology-Based Learning for Negotiation", *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pp. 177–184, Milan, Italy, September 2009.

3. Aydoğan, R. and P. Yolum, "Learning Opponents Preferences for Effective Negotiation: An Approach Based on Concept Learning", *Journal of Autonomous Agents and Multi-Agent Systems*, 2011.

4. Aydoğan, R., N. Taşdemir, and P. Yolum, "Reasoning and Negotiating with Complex Preferences Using CP-nets", Aalst, W., J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, W. Ketter, H. Poutr, N. Sadeh, O. Shehory, and W. Walsh (editors), *Agent-Mediated Electronic Commerce and Trading Agent Design and Analysis*, Vol. 44 of *Lecture Notes in Business Information Processing*, pp. 15–28, Springer Berlin Heidelberg, 2010.

5. Aydoğan, R. and P. Yolum, "Effective Negotiation with Partial Preference Information", *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1605–1606, Toronto, Canada, May 2010.

6. Aydoğan, R. and P. Yolum, "Heuristics for CP-Nets: Negotiating Effectively with Partial Preferences", *HUCOM Workshop in Group Decision and Negotiation*, The Netherlands, 2010.

7. Aydoğan, R., T. Baarslag, K. Hindriks, C. M. Jonker, and P. Yolum, "Heuristic-based Approaches for CP-nets in Negotiation", *Proceedings of the Forth International Workshop on Agent-based Complex Automated Negotiations*, Taipei, Taiwan, 2011.

8. Aydoğan, R., T. Baarslag, K. Hindriks, C. M. Jonker, and P. Yolum, "Utility-based Negotiation without Knowing Utilities: A CP-net Approach", *Under Review in a Journal*, 2011.

9. Gérard, R., S. Kaci, and H. Prade, "Ranking Alternatives on the Basis of Generic Constraints and Examples: A Possibilistic Approach", *Proceedings of the Twentieth International Joint Conference on Artifical intelligence*, pp. 393–398, 2007.

10. Jennings, N. R., P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated Negotiation: Prospects, Methods and Challenges", *International Journal of Group Decision and Negotiation*, Vol. 10, No. 2, pp. 199–215, 2001.

11. Raiffa, H., *The Art and Science of Negotiation*, Harvard University Press, Cambridge, 1982.

12. Abedin, F., K.-M. Chao, N. Godwin, and H. Arochena, "Preference Ordering in Agenda Based Multi-issue Negotiation for Service Level Agreement", *Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops*, pp. 19–24, IEEE Computer Society, Washington, DC, USA, 2009.

13. Singh, M. P., "Value-oriented Electronic Commerce", *IEEE Internet Computing*, Vol. 3, No. 3, pp. 6–7, 1999.

14. Lai, G., C. Li, K. Sycara, and J. A. Giampapa, "Literature Review on Multi-attribute Negotiations", Technical report, Robotics Institute, Pittsburgh, PA, December 2004.

15. Russell, S. J. and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

16. Wooldridge, M. and N. R. Jennings, "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review*, Vol. 10, No. 2, pp. 115–152, 1995.

17. Wooldridge, M., *An Introduction to Multiagent Systems*, John Wiley & Sons, Ltd,

2009.

18. Singh, M. P. and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, John Wiley & Sons, Ltd, West Sussex, England, 2005.

19. Boutilier, C., R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole, "CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements", *Journal of Artificial Intelligence Research*, Vol. 21, pp. 135–191, 2004.

20. Baarslag, T., K. Hindriks, C. M. Jonker, S. Kraus, and R. Lin, "The First Automated Negotiating Agents Competition", Ito, T., M. Zhang, V. Robu, S. Fatima, and T. Matsuo (editors), *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence (to appear)*, Springer-Verlag, 2010.

21. Faratin, P., C. Sierra, and N. R. Jennings, "Negotiation Decision Functions for Autonomous Agents", *International Journal of Robotics and Autonomous Systems*, Vol. 24, pp. 3–4, 1998.

22. Fatima, S., M. Wooldridge, and N. Jennings, "On Efficient Procedures for Multi-issue Negotiation", Fasli, M. and O. Shehory (editors), *Agent-Mediated Electronic Commerce. Automated Negotiation and Strategy Design for Electronic Markets*, Vol. 4452 of *Lecture Notes in Computer Science*, pp. 31–45, Springer Berlin, Heidelberg, 2007.

23. Hindriks, K. and D. Tykhonov, "Opponent Modelling in Automated Multi-Issue Negotiation Using Bayesian Learning", *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 331–338, Estorial, Portugal, 2008.

24. Jonker, C. M., V. Robu, and J. Treur, "An Agent Architecture for Multi-attribute Negotiation Using Incomplete Preference Information", *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 15, No. 2, pp. 221–252, 2007.

25. Boutilier, C., R. I. Brafman, H. H. Hoos, and D. Poole, "Reasoning with Condi-

tional Ceteris Paribus Preference Statements", *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 71–80, Stockholm, 1999.

26. Lau, R. Y. K., M. Tang, O. Wong, S. W. Milliner, and Y. P. Chen, "An Evolutionary Learning Approach for Adaptive Negotiation Agents", *International Journal of Intelligent Systems*, Vol. 21, No. 1, pp. 41–72, 2006.

27. Boutilier, C., F. Bacchus, and R. I. Brafman, "UCP-Networks: A Directed Graphical Representation of Conditional Utilities", *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 56–64, Seattle,USA, 2001.

28. Brafman, R. I. and C. Domshlak, "Preference Handling — An Introductory Tutorial", *AI Magazine*, Vol. 30, No. 1, pp. 58–86, 2009.

29. Chevaleyre, Y., P. E. Dunne, U. Endriss, J. Lang, M. Lemaitre, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa, "Issues in Multiagent Resource Allocation", *Informatica*, Vol. 30, No. 1, pp. 3 – 31, 2006.

30. Hansson, S. O., "What is Ceteris Paribus Preference?", *Journal of Philosophical Logic*, Vol. 25, No. 3, pp. 307–332, 1996.

31. Domshlak, C., E. Hllermeier, S. Kaci, and H. Prade, "Preferences in AI: An Overview", *Artificial Intelligence*, Vol. 175, No. 7-8, pp. 1037–1052, 2011.

32. Walsh, T., "Representing and Reasoning with Preferences", *AI Magazine*, Vol. 28, No. 4, pp. 59–69, 2007.

33. Tykhonov, D., *Designing Generic and Efficient Negotiation Strategies*, Ph.D. thesis, Delft University of Technology, Delft, The Netherlands, 2010.

34. Faratin, P., C. Sierra, and N. R. Jennings, "Using Similarity Criteria to Make Issue Trade-offs in Automated Negotiations", *Artificial Intelligence*, Vol. 142, pp. 205–237, 2002.

35. Coehoorn, R. M. and N. R. Jennings, "Learning on Opponent's Preferences to Make

Effective Multi-issue Negotiation Trade-offs", *Proceedings of the Sixth International Conference on Electronic Commerce*, pp. 59–68, 2004.

36. Jonker, C. and V. Robu, "Automated Multi-Attribute Negotiation with Efficient Use of Incomplete Preference Information", *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1054–1061, New York, USA, 2004.

37. Keeney, R. L. and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Wiley series in probability and mathematical statistics, John Wiley & Sons, Inc, New York, 1976.

38. Bacchus, F. and A. Grove, "Graphical Models for Preference and Utility", *Proceedings of the Proceedings of the Eleventh Conference Annual Conference on Uncertainty in Artificial Intelligence*, pp. 3–10, Morgan Kaufmann, San Francisco, CA, 1995.

39. Fishburn, P. C., *Utility Theory for Decision Making*, Wiley, New York, 1970.

40. Li, C., J. A. Giampapa, and K. Sycara, "A Review of Research Literature on Bilateral Negotiations", Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2003.

41. Sierra, C., P. Faratin, and N. Jennings, "A service-oriented negotiation model between autonomous agents", Boman, M. and W. van de Velde (editors), *Proceedings of the Eighth European Workshop on Modelling Autonomous Agents in Multi-Agent World, MAAMAW'97*, Vol. 1237 of *Lecture Notes in Artificial Intelligence*, pp. 17–35, Springer-Verlag, 1997.

42. Rubinstein, A., "Perfect Equilibrium in a Bargaining Model", *Econometrica*, Vol. 50, No. 1, pp. 97–109, 1982.

43. Osborne, M. J. and A. Rubinstein, *A Course in Game Theory*, MIT Press, 1994.

44. Brin, S. and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine", *Computer Networks and ISDN Systems*, Vol. 30, No. 1–7, pp. 107–117,

1998.

45. Pelleg, D. and A. Moore, "X-means: Extending K-means with Efficient Estimation of the Number of Clusters", *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 727–734, Morgan Kaufmann, San Francisco, 2000.

46. Hindriks, K. V., C. M. Jonker, and D. Tykhonov, "A Multi-Agent Environment for Negotiation", *Multi-Agent Programming: Languages, Tools and Applications*, pp. 333–363, Springer, US, 2009.

47. Gode, D. K. and S. Sunder, "Allocative Efficiency of Markets with Zero-Intelligence Traders: Market as a Partial Substitute for Individual Rationality", *The Journal of Political Economy*, Vol. 101, No. 1, pp. 119–137, 1993.

48. Hindriks, K. V. and D. Tykhonov, "Towards a Quality Assessment Method for Learning Preference Profiles in Negotiation", Aalst, W., J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, W. Ketter, H. Poutr, N. Sadeh, O. Shehory, and W. Walsh (editors), *Agent-Mediated Electronic Commerce and Trading Agent Design and Analysis*, Vol. 44 of *Lecture Notes in Business Information Processing*, pp. 46–59, Springer Berlin Heidelberg, 2010.

49. Mitchell, T. M., *Machine Learning*, McGraw Hill, New York, 1997.

50. Quinlan, J. R., "Induction of Decision Trees", *Machine Learning*, Vol. 1, No. 1, pp. 81–106, 1986.

51. Mitchell, T. M., "Generalization as Search", *Artificial Intelligence*, Vol. 18, No. 2, pp. 203–226, 1982.

52. Aydoğan, R. and P. Yolum, "Learning Consumer Preferences Using Semantic Similarity", *Proceedings of Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1293–1300, Honolulu, Hawaii, May 2007.

53. Raileanu, L. E. and K. Stoffel, "Theoretical Comparison between the Gini Index and Information Gain Criteria", *Annals of Mathematics and Artificial Intelligence*,

Vol. 41, No. 1, pp. 77–93, 2004.

54. Alpaydın, E., *Introduction to Machine Learning*, The MIT Press, October 2004.

55. Gruber, T. R., "A Translation Approach to Portable Ontology Specifications", *Knowledge Acqusition*, Vol. 5, No. 2, pp. 199–220, 1993.

56. McGuinness, D. L., "Ontologies Come of Age", *Spinning the Semantic Web*, pp. 171–194, MIT Press, Cambridge, 2003.

57. Demšar, J., "Statistical Comparisons of Classifiers over Multiple Data Sets", *Journal of Machine Learning Research*, Vol. 7, pp. 1–30, 2006.

58. Jonker, C. M. and J. Treur, "An Agent Architecture for Multi-Attribute Negotiation", *Proceedings of the Seventeenth International Joint Conference on Articifial Intelligent*, pp. 1195–1201, Morgan Kaufman, 2001.

59. Luo, X., N. R. Jennings, N. Shadbolt, H. Leung, and J. H. Lee, "A Fuzzy Constraint Based Model for Bilateral, Multi-issue Negotiations in Semi-competitive Environments", *Artifical Intelligence*, Vol. 148, No. 1-2, pp. 53–102, 2003.

60. Ramchurn, S. D., C. Sierra, L. Godo, and N. R. Jennings, "Negotiating using Rewards", *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 400–407, 2006.

61. Rahwan, I., S. D. Ramchurn, N. R. Jennings, P. Mcburney, S. Parsons, and L. Sonenberg, "Argumentation-based Negotiation", *Knowledge Engineering Review*, Vol. 18, No. 4, pp. 343–375, 2003.

62. Rossi, F., K. B. Venable, and T. Walsh, "*m*CP Nets: Representing and Reasoning with Preferences of Multiple Agents", *Proceeding of the Nineteenth National Conference on Artificial Intelligence*, pp. 729–734, 2004.

63. Li, M., Q. B. Vo, and R. Kowalczyk, "An Efficient Procedure for Collective Decision-making with CP-nets", *Proceeding of the Nineteenth European Conference on Artifi-*

cial Intelligence, pp. 375–380, Lisbon, Portugal, 2010.

64. Rossi, F., K. B. Venable, and T. Walsh, "CP-networks: Semantics, Complexity, Approximations and Extensions", *Proceedings of the Forth International Workshop on Soft Constraints (Soft-02) in CP-2002*, USA, 2002.

65. Chalamish, M. and S. Kraus, "AutoMed: An Automated Mediator for Bilateral Negotiations under Time Constraints", *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 248:1–3, 2007.

66. Grabisch, M., "Fuzzy Integral in Multicriteria Decision Making", *Fuzzy Sets and Systems*, Vol. 69, No. 3, pp. 279–298, 1995.