GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS

The author of the PhD dissertation: Jerzy Kaczorek
Scientific discipline: Computer Science

DOCTORAL DISSERTATION

Title of PhD dissertation: Automated Negotiations Over Collaboration Protocol Agreements

Title of PhD dissertation (in Polish): Metody automatycznego negocjowania uzgodnień w realizacji usług internetowych

Supervisor *signature*	Second supervisor *signature*
prof. Bogdan Wiszniewski	
Auxiliary supervisor *signature*	Cosupervisor *signature*

Gdańsk, year 2015

Contents

List of symbols

$\oplus$	bitwise XOR
$\succeq$	preference operator
$\langle\rangle$	vector
$\{\}$	set
$\|\|$	cardinality of a set or a vector size
α_i^k	player's i k-th move
β	bit vector
δ_i	player's i discount factor
ζ	a pair of offers
$\eta(\beta)$	a number of 1-s in β
$\iota(\beta)$	length of β
κ	function obtaining results from a neural network
π_i	player's i payoff or player's i payoff function
ρ	function which calculates an average value
$\phi(\beta)$	function mean of vector β showing which bit occurs more often in a bit vector
φ	neural network transfer function
ς_i^k	sequence of offers submitted by player P_i
τ_{P_1}	player's P_1 option tree
ω	a bit word
Γ	utility priority
Δ	normalized distance
Λ	a relation of the move number to the bargaining set size
$\Xi(x11)$	implementation of policy $x11$
Π	payoff vector
Υ	function which returns the index of attribute value
Φ	centroid
Ψ	cluster
a	an attribute
b	bit field
d	distance between occurrence vectors
o	an offer

o_c	the best offer
s	strategy
$u(v)$	utility of the attribute value v
v_k^j	k-th attribute value of j-th offer
w	neural network weight
A_k	set of values of the k-th attribute
A	space of offers (a set of m-vectors)
$\mathbf{A}$	matrix $\mathbf{A}$
$A_{i\bullet}$	i-th row of matrix $\mathbf{A}$
$A_{\bullet j}$	j-th column of matrix $\mathbf{A}$
C	bargaining set
D	set of players' discount factors
E	decision tree edge
E_F	partial entropy
G	game
H	neural network hidden layer
I	entropy
L	decision tree leaf
N	decision tree node
$\mathbb{N}$	set of natural numbers
O	neural network output layer
P_i	player i
P_{-i}	player's i opponent
Q_{A_k}	ordered set of attribute A_k value labels
R	input data matrix
S	set of strategies
$\mathbb{T}$	decision tree
T	a condition of concluding a game
$U(o)$	utility of the offer o
W	vector of neural network weights
X	execution context
$\mathbb{Z}$	set of integer numbers

Chapter 1

Game theoretic negotiation models

Individuals, who collaborate in a network organization, work together for a common purpose using some computer-mediated communication facility to link across boundaries, interact and exchange information. They are often called *knowledge workers* and exchange documents constituting units of *information*, and more recently, also units of *interaction* [17]. This dichotomy has become more apparent with the advent of *active documents*.

The first architectural model of the active document has been proposed by Phelps [51], who defined a *multivalent document (MVD)* architecture. MVD incorporated layers of functionality that could be added dynamically by users during the entire document lifecycle from various physically distributed locations in order to define document *behaviors*. This concept paved the way for *reactive* documents, capable of supporting complex interaction with users.

The idea of behaviors embedded in a document evolved further into *active properties* of *placeless documents* introduced by Dourish [10]. Active properties provided users with specific services attached to the document, which could be invoked regardless of its actual location – changed each time the document was emailed or copied.

The concept of a mobile *document-agent*, as a truly *proactive* entity, capable of traveling from computer to computer under its *own control*, has been demonstrated by Satoh [60] with his MobiDoc compound document framework. The agent's body was a program defining document's behavior, while its services defined various APIs for document components.

With advances in wireless communication and mobile devices the concept of a document agent has been firmly established at the beginning of this century. Clear distinction between *reactive* and *proactive* documents has been acknowledged and the focus has been put on the middleware enabling

proactive documents to coordinate activities with other active documents [6]. The underlying model has been coordination of agent activities by exchanging structured tuples in a logic data space, e.g., MARS-X [4], building on the concept introduced by the Linda coordination language [15].

Explicit representation of workflow activities as *steps* applied to documents in a really distributed manner appeared in the WorkSpaces system developed by Tolksdorf [65]. Steps were described there with the use of a mark-up language and retrieved (along with documents providing content to work on) by special WorkSpace engines distributed over the network from a tuple space mentioned before. Owing to that, workflows could be *designed* in an orderly fashion *prior* actual process execution as graphs comprising both: basic steps, representing the "work" on document content, and coordination steps, representing the "flow" of work between collaborators. WorkSpace engines introduced also a distinction between *automatic* steps, performed within the system (most likely by the document itself), and *external* steps, performed by a local application or user with the document as input.

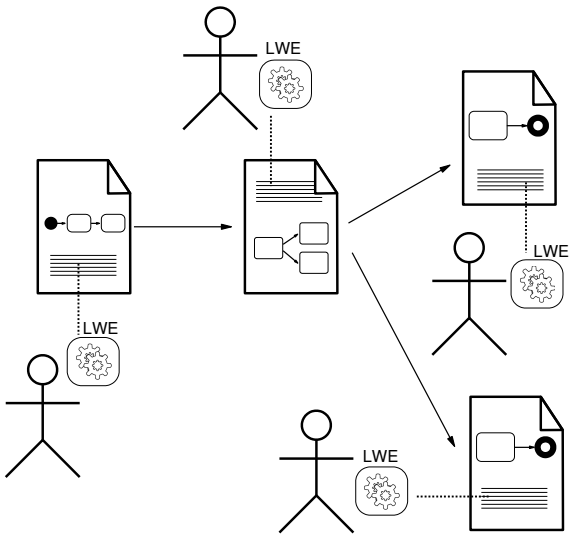


Figure 1.1: Mobile documents with embedded workflows

Mobile interactive documents MIND [19] used a coordination model based on autonomous mobile agents, which could carry both, the *content* to be worked on and *description* of steps to be interpreted by local user devices. Each document has an embedded description of its migration path, which describes activities and transitions that must be followed during its lifetime. An *activity* represents a piece of work to be performed by the worker on

the incoming document content, whereas a *transition* indicates where the outgoing document (or documents), constituting a result of the just completed activity, should migrate next. This idea is outlined in Figure 1.1, where activities are represented by rectangles, transitions by arrows, and two distinguished start and termination activities by filled and outlined circles respectively. Workers may interact with the content of MIND documents using any device currently at their disposal, from workstations to laptops, to tablets, smartphones or even cellphones. Each device has a *Local Workflow Engine (LWE)* client installed, capable of unpacking and packing the documents and sending them to other workers of the organization.

1.1 Motivation

A single activity performed in the execution context provided by the device on which the mobile document is currently located, depends on the policy of the document originator, operational characteristics of the currently used worker's device, and preferences of the knowledge worker responsible for the current processing step. Proactive MIND documents may handle that in several ways: activity may be performed automatically by the *embedded* document code, if only allowed by the worker or his/her device, it may be performed manually by the worker, using *local* services or tools installed on his/her device, also the document or the worker may call some *external* (third party) service indicated by the document – if only Internet connection is available or practical at the time of executing the activity.

The proactive document MIND architecture developed in the first phase of the MeNaID project [43] opened a new interesting research perspective for this Thesis, motivated by the fact that beliefs, desires and intentions of MIND document-agents expressed in their embedded migration paths and services [18] may often generate conflicts with execution devices. In the Thesis we postulate that these conflicts can be resolved with negotiations [32]. This problem is not trivial, as:

- execution contexts may vary, since the same worker may use different devices when performing activities of the same business process, e.g., using a workstation when in office, a smartphone during travel between office and home, and a laptop at home.

- user preferences for the same device may depend on its current location, e.g., when out of the office and accessing an untrusted network.

Document agents arriving to the particular device may have incomplete

information on the specific execution context for several reasons, which may be classified along the lines proposed in [57]:

- *laziness*, when too much about the current context must be known by the agent, i.e. it has not enough resources to acquire all the information on the context whereabouts;
- *declarative ignorance*, when the agent has been programmed by its originator in the way that it does not attempt to collect all available information on the context;
- *procedural ignorance*, when both the document-agent and the execution device do not attempt to predict consequences of their current activities because of the lack of complete information on the execution context.

Importance of the problem of incomplete information indicated above may be contrasted to the problem of providing an optimal viewing experience for users of a wide range of personal devices (from cellphones to workstation monitors), willing to read and navigate arbitrarily complex Web documents with a minimum of resizing, panning, and scrolling, which has been addressed by the Responsive Web Design approach advocating the *media queries* [21]. Media queries allow documents to use different CSS style rules based on characteristics of the device the document is being displayed on. However, the problem investigated in the Thesis is far more complex than the one addressed by media queries because incomplete information about the execution context may prevent originators of document agents from making media queries realistically implementable, i.e. agents must remain lazy or ignorant, as classified before.

Throughout the rest of the Thesis we will argue that *negotiations* can provide a solution to the problem of reaching an agreement between a document-agent and execution device not only when simple querying may not be applied because of incomplete information, but also when the two parties may have conflicting requirements on how the MIND document should perform its current activity.

1.2 Formal statement of the problem

Phases of the local lifetime of a document have been specified by a finite state machine in Figure 1.2. State transitions specify a generic functionality of LWE, which is just an email client to the knowledge worker, driven by embedded functionality of incoming MIND documents [18].

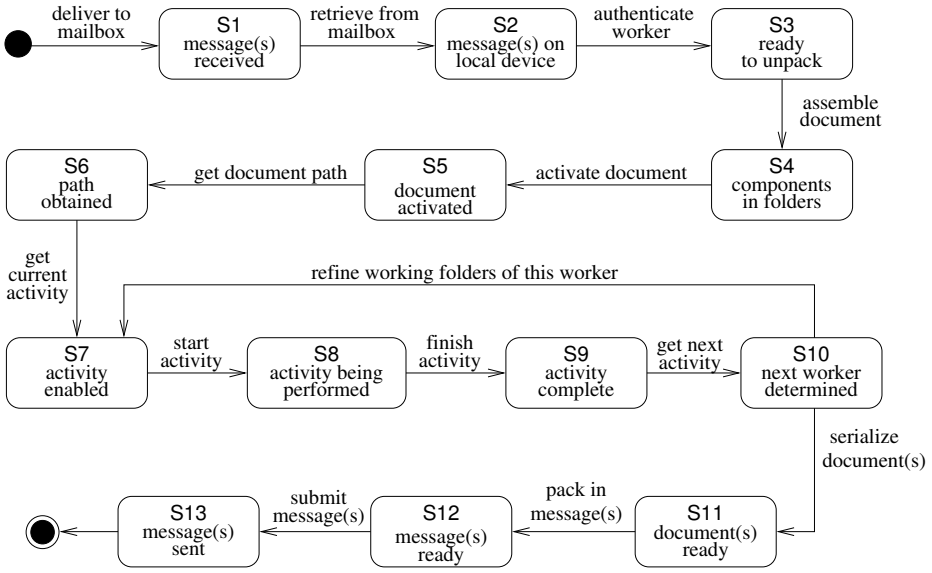


Figure 1.2: Proactive MIND document states on the worker's device

The initial state (S1) of LWE corresponds to the phase of a document lifetime when a message is received, i.e. placed in the respective worker's mailbox hosted by some email server. Depending on the particular collaboration pattern [18], transition to the next state (S2) may require delivery of one or more messages to the mailbox. LWE identifies in that effect all relevant messages in the mailbox and retrieves them to the (local) inbox at the knowledge worker's device. Making document components ready to unpack (S3) may require authentication of the worker. Unpacking of the retrieved messages enables assembling the hub document (S4), which involves creation of the local folder structure to place document components. Activation of the document enables its embedded functionality, so that it may interact with LWE, the knowledge worker and his/her local system, as well as services of external servers (S5). This interaction begins with obtaining the document `path` component (S6) and determining the activity to be started based on its internal data (S7). This state involves also resolving of conflicts (if any) between the activated document and the device on options currently available for the activity. The enabled activity is started and performed using the document services (S8): automatically, or via interaction with the knowledge worker. When the required work is done the current activity is considered complete (S9). This implies determining the next activity to be performed and the respective worker responsible for that (S10). If the worker is the same the content of local folders is refined (cleaned) to prepare document components processed so far for the next activity, which is eventually

enabled (S7) for the next cycle. Otherwise, the document components are serialized (S11), depending on the particular collaboration pattern packed in one or more messages (S12), submitted to the email server and sent out (S13).

Throughout the rest of this Thesis we will focus on state S7 in the diagram in Figure 1.2, when the activated document and the worker's device exchange offers that represent optional values of the common set of attributes describing the negotiated service. Let us consider offers exchanged by negotiating parties as m-vectors of items:

$$o = \langle item_1, item_2, \ldots, item_m \rangle \tag{1.1}$$

Each $item_k$, where $k = 1, \ldots, m$, can be assigned a value of any attribute specific type chosen from the set of values:

$$A_k = \{a_{k_1}, a_{k_2}, \ldots, a_{k_n}\}, \tag{1.2}$$

where $k_n = |A_k|$. Operator $|\ |$ denotes cardinality of its argument set, A_k represents a k-th attribute of the negotiated service. Set A of all m-vectors

$$A = A_1 \times A_2 \times \ldots \times A_m \tag{1.3}$$

is called a *space of offers*. Based on that we can define offers in a more formal fashion:

Definition 1. *Offer $o_j \in A$, where $j = 1, \ldots, |A|$ is a vector of attribute values*

$$o_j = \langle v_1^j, v_2^j, \ldots, v_m^j \rangle \tag{1.4}$$

where $v_k^j \in A_k$ and $k = 1, \ldots, m$.

Each single attribute value has an assigned numerical value, which reflects utility of the related attribute value. Utility of attribute value $a_{k_i} \in A_k$, for $k_i = 1, \ldots, |A_k|$, is calculated with function $u_k : A_k \to \mathbb{N}$. Each party has its own set of functions $\{u_1, u_2, \ldots, u_m\}$ to calculate utility of each item in the offer.

Given that, utility of each offer o_j is calculated as:

$$U_i(o_j) = \sum_{k=1}^{m} u_i(v_k^j) \tag{1.5}$$

where according to formulas 1.3 and 1.4, $j = 1, \ldots, |A|$

Throughout the rest of this thesis we will use values of the utility function, normalized against $\max_j(U_i(o_j))$, so that

$$U_i : A \to (0, 1] \tag{1.6}$$

The problem is to find the best offer $o_c \in A$ that is acceptable to both parties, P_1 and P_2, in other words maximizes their utility; we denote that as

$$o_c = \arg\max_{o \in A} U_1(o)U_2(o). \tag{1.7}$$

We will call offer o_c the *contract* between negotiating parties P_1 and P_2.

Unfortunately, the problem of finding a contract cannot be stated just as solving Equation 1.7 because neither P_1 nor P_2 knows each other's utility function.

Through the Thesis we will argue that negotiations can provide a solution to Equation 1.7 when parties do not know one another's utility function. Note that according to Definition 1, offers consist of many items. Negotiation between two parties that exchange offers of the form described by Formula 1.4 are called in the literature *multi-issue bilateral negotiations* [27]. Because of the use of utility functions in the negotiation process, it is usually called *bargaining* [45]. Since offers considered in our bargaining model specify execution contexts provided by devices to proactive documents we will call the contract $o_c \in A$ the *collaboration agreement*.

The following statement will be investigated in this thesis:

> **Selection of negotiation strategies based on the bargaining model enables effective generation of collaboration agreements between conflicted parties.**

1.3 Formulation of the problem as a game

The bargaining model considered in this Thesis derives several important game theoretic notions introduced in the following subsections.

1.3.1 Simple Bargaining Game

First, we have to introduce the concept of *rationality*, which implies that each party, termed a *player*, is rational and assumes that its opponent is also rational. Given that, each player has a complete contingent plan for choosing offers from the space of offers. We will call such a plan a *strategy*. A formal definition of a strategy will be provided later in this section.

We define shortly a two person game (after [66]) as a tuple $G = \langle P, S, \Pi \rangle$ consisting of the set of players $P = \{P_1, P_2\}$, each with the associated finite set of strategies $S = \{S_1, S_2\}$ and payoffs $\pi_i \in \Pi$ corresponding to each player P_i, which is the result of a special function that maps strategies into real numbers.

Each single player's choice will be called a *move*, denoted as α_i^k where $i = 1, 2$ indicates a player and k denotes a move number $k \in \mathbb{N} \cup \{0\}$, or $\mathbb{Z}_{\geq 0}$ for brevity.

A sequence of two moves, each one for each player, starting from P_1 is called a *stage* and denoted as $(\alpha_1^k, \alpha_2^{k+1})$. Since $n \in \mathbb{Z}_{ge0}$, $k = 2n$ for player's P_1 moves and $k = 2n + 1$ for player's P_2 moves, where n denotes the stage number. A *multi-stage game* is a game for which $n \geq 0$.

In order to make a choice of offer $o \in A$ after making move α_i^k player P_i will valuate it with its payoff function $\pi_i : A \times N \to [0, 1]$. This function differs from utility function U_i in that it takes a context of choosing offer o, in particular A reflects whether o has been proposed before, i.e. in some preceding step $k' \leq k$.

Discounting is a mechanism which is used to compare values in different stages of the same game. The value of the offer from stage n equals the value of the offer from stage $n+1$ multiplied by the given player's discount factor. The value of an offer is multiplied by a discount factor each time when the given player rejects an offer. Let us consider a multistage game G and player P_1 sending its offer. If P_2 accepts that offer, it is not discounted. If P_2 rejects the offer and submits its own offer, the offer has to be discounted. If P_1 rejects the offer by submitting its own offer, the new player's P_1 offer is also discounted.

When describing various players' activities we use the indexes 1 and 2 to emphasize to whom the given activity belongs. However, sometimes we are not interested in assigning an activity to the player but rather contrasting the player's and its opponent's activities. Then we use index i for the current player and $-i$ for its opponent.

To recapitulate, offers are discounted when player P_i receives opponent's P_{-i} offer $o_{-i} \in C$ and decides to reject it. Then in order to compare utility $U_i(o_{-i})$ with utility $U_i(o_i)$ of the new offer o_i made by player P_i, utility $U_i(o_i)$ has to be discounted with δ_i.

Let us introduce the following definition:

Definition 2. *Factor $\delta_i \in (0, 1]$ used to calculate payoff $\pi_i : C \times Z_{\geq 0} \to [0, 1]$ for offer o by player P_i in the $(k + 2)$-th move as:*

$$\pi_i(o, k + 2) = \delta_i \cdot \pi_i(o, k)$$

is called a discount factor. *Each player has its own discount factor that remains constant throughout the entire game*

Recall that each player P_i, $i = 1, 2$ has its own utility function $U_i(o)$ for each $o \in A$. Realistically, players will continue choosing offers from a certain

subset of offers $C \subset A$, which we call the *bargaining set*. In each move either player attempts to maximize its own utility function by complying with the rules of a *simple bargaining game*. We will define such a game formally later, below we just specify its rules (given originally in [35]):

Rules of a simple bargaining game (SBG)

1. The game is started by player P_1.
2. Players P_i, $i = 1, 2$ keep in secret their private information, including U_i, δ_i and π_i, but share knowledge on the bargaining set C (a set of offers that players negotiate over).
3. Values of players' offers are discounted at each transition to the next stage.
4. Players exchange offers until the game is concluded, i.e., one of the players accepts an offer or quits the game.
5. The game is concluded by player P_i when:
   (a) P_i repeats its own offer o' what means quitting the game by P_i with the payoff $\pi_i(o', k) = \pi_{-i}(o', k-1) = 0$ for each player.
   (b) P_i repeats player's P_{-i} offer o'' what means accepting it as the contract and exiting.

A simple bargaining game can provide a solution to Equation 1.7, known as the *Nash equilibrium* [46], which players may find even when not knowing one another utility functions. The rules presented above are implemented by Algorithm 1.1 in procedure `playSBG()`. The algorithm will be used later in the experiments. One of the parameters required by the procedure is another procedure – `submitOffer()`. This parameter will indicate values implementation of the `submitOffer()` procedure, namely `submitOffer(SAlg)` implementing Algorithm 1.2 and `submitOffer(EAlg)` implementing Algorithm 1.4. They are optional negotiation algorithms investigated in this Thesis. We will specify formally several such algorithms later in the current Chapter. Algorithm 1.1 will run as follows:

1. In the first step the bargaining set is created by intersecting both players' option trees,
2. Next a loop is executed. In the body of the loop procedure `submitOffer()` is called. First offer o_1 is created by calling the procedure for arguments

belonging to player P_1 and offer o_2. Next offer o_2 is created by calling the procedure for arguments belonging to player P_2 and offer o_1.

3. The loop ends when offer o_1 equals o_2, i.e. when players agree the offer. Additionally the loop may be broken if a player submits an offer which was sent already.

Algorithm 1.1: playSBG()

```
1  Data: structures representing players P_i, with sets of sent P_i.C_S and
         received offers P_i.C_R (i indicates the player's number),
         bargaining set C shared by players, player's P_i offer o_i, move
         number k.
2  procedure submitOffer()
3  Result: chosen offer o_i and move number mNo
4  while o_1 ≠ o_2 do
5      o_1 ← submitOffer(C, P_1.C_R, P_1.C_S, o_2, k);
6      if o_1 = o_2 then
7          return [o_1,k];
8      k ← k + 1;
9      o_2 ← submitOffer(C, P_2.C_R, P_2.C_S, o_1, k);
10     if o_1 = o_2 then
11         return [o_2,k];
12     k ← k + 1;
```

Bargaining set in Algorithm 1.1 is agreed by both players before starting the bargaining game. We will explain the problem in more detail in Section 3.3.

Definition 3. Strategy *is a function:*

$$s_i : \mathbb{Z}_{\geq 0} \to C$$

which associates all possible moves of player P_i *that follow move* α_i^k, $0 \leq k \leq |C|$, *with the relevant offers.*

We will define the Nash equilibrium as the *strategy profile* $\langle s_i'(k), s_{-i}'(k+1)\rangle$ representing the *best responses* to the opponents' strategies. We define the best response as strategy s_i' which is better than any other strategy s in the sense of payoff they provide to the player. Namely, strategy s' is better than any other strategy s if the payoff of the offer returned by strategy s' is greater than the payoff of the offer returned by strategy s.

Definition 4. *Strategy profile* $\langle s'_i(k), s'_{-i}(k+1)\rangle$ *is the* Nash equilibrium *of* SBG *if*

$$(s'_i, s'_{-i}) \succeq (s, s'_{-i}) \tag{1.8}$$

for any $s \in S$*; by '*$\succeq$*' we denote a pairwise comparison operator of* 2*-element vectors.*

Strategies that satisfy Equation 1.8 will be called *equilibrium strategies.*

The advantage of using a Nash equilibrium is that the knowledge of the opponents' strategies is not needed to find it: the player needs only to express beliefs on its opponent's strategies. On the other hand, a Nash equilibrium deficiency with regard to Equation 1.7 is that players may be interested only in maximizing their own payoffs. We may illustrate that after defining function *map*:

Definition 5. *Function map* : $f, X \to Y$ *where* f *is a function and* $X = \langle x_1, x_2, \ldots, x_n\rangle$*, calculates* $Y = \langle f(x_1), f(x_2), \ldots, f(x_n)\rangle$

Let us consider the following example:

Example 1. *Players* P_1 *and* P_2 *choose offers from bargaining set* $C = \{o_1, o_2\}$*; let utilities of the respective players be calculated* $map(U_1, C) = \{0.2, 0.1\}$ *and* $map(U_2, C) = \{0.1, 0.2\}$*. Moreover, let respective discount factors be* $\delta_1 = 0.7$ *and* $\delta_2 = 0.6$*.*

The game from Example 1 is illustrated in Figure 1.3 as a tree. Its root represents the initial state of the game, where the first (starting) player's P_1 move is to be made. All possible sequences of moves start from the root. Upon starting, each sequence goes down the tree to a distinguished terminal node. In other words, each complete sequence of moves, from the starting one to the concluding one is represented in a game tree by a tree path, going from the root to one of its leafs. Non-terminal nodes are called *decision nodes.* A label of each node consists of the index of the player who makes a decision at the given node, and separated by colon, the name of the node denoted by a character. Additionally of each terminal node the respective payoff vector $\langle \pi_1, \pi_2\rangle$ is displayed. Edges are labeled with symbols representing offers.

- In move α_1^0 at node 1:a player P_1 makes its first choice.
- If it chooses o_1, then node 2:b is reached, that is where the decision is made by P_2.
- In move α_2^1 player P_2 may accept its opponent's P_1 offer by choosing it or reject it and choose offer o_2. By choosing o_1, i.e. accepting the opponent's offer, P_2 gets $\pi_2 = 0.1$ and the game will be over because of the game rules. Its opponent gets $\pi_1 = 0.2$.

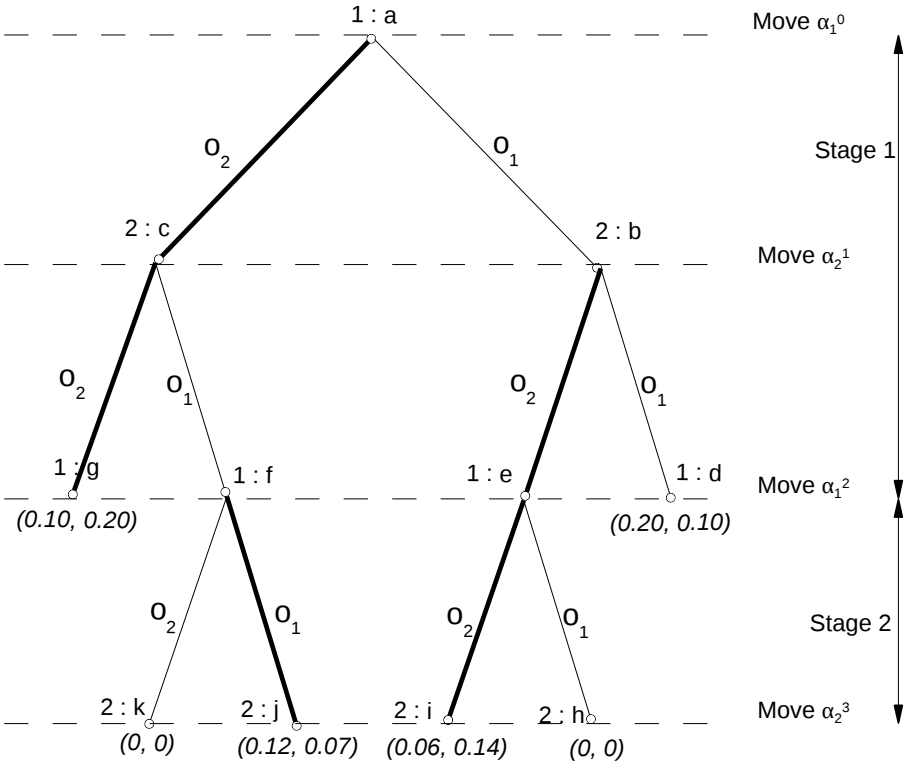


Figure 1.3: Game in the extensive form

- If in move α_2^1 player P_2 chooses o_2 node 1:e is reached; now P_1 makes its choice in move α_1^2.
- P_1 may now repeat its first offer o_1 and receive $\pi_1 = 0$ by reaching node 2:h, or accept its opponent's P_2 offer o_2 to get $\pi_1 = 0.14$ and reach node 2:i. Its opponent P_2 gets then $\pi_2 = 0.06$. At node 2:i both offers are discounted.
- If P_1 chooses o_2 at node 1:a , node 2:c is reached. Now P_2 has to choose between o_1 and o_2.
- If P_2 chooses offer o_2, the game is concluded. Reaching node 1:g gives the respective payoffs $\pi_1 = 0.1$ and $\pi_2 = 0.2$.
- If P_2 chooses offer o_1, node 1:f is reached.
- At node 1:f player P_1 chooses between the offers o_1 and o_2. If it chooses o_1, it gets $\pi_1 = 0.12$ and its opponent $\pi_2 = 0.07$. At node 2:j both

offers are discounted.

Representation of games with trees, like the one shown in Figure 1.3, is often called the *extensive form* [66]. This form is suitable for finding a game solution with a *backward induction* method. A solution obtained in this way is called a *Perfect Nash Equilibrium*(PNE). A strategy profile is PNE if it determines a Nash equilibrium in every subgame of the given game, where a subgame is a game, which tree starts in any node of the original game tree [68].

Let us continue with the tree shown in Figure 1.3 and consider:

- node 1:f; in the game the only decision is taken by player P_1. Player P_1 chooses strategy o_1, since it gives $\pi_1 = 0.12$, while choosing o_2 would give it $\pi_1 = 0$. We mark strategy of choosing o_1 with a thick line.

- node 2:c; the decision is taken by P_2. Player P_2 knows that P_1 is rational and it chooses o_1 in subgame 1:f. It means that $\pi_2 = 0.7$. However, if P_2 chooses o_2, it gets $\pi_2 = 0.20$. So, P_2 chooses o_2 and we mark that fact in the tree with a thick line.

- node 1:e; player P_1 chooses o_2, what gives it payoff $\pi_1 = 0.6$ compared to payoff $\pi_1 = 0$ received it choosing o_1. We mark it also with a thick line.

- node 2:b; player P_2 chooses o_2 rather than o_1. Since P_1 is rational, P_2 knows, that in 1:e player P_1 chooses o_2. So, the choice of o_2 gives it payoff $\pi_2 = 0.14$, while o_1 would give it $\pi_2 = 0.10$. Player's P_2 strategy is to choose o_2, as marked with a thick line.

- node 1:a; player P_1 makes the decision. It knows that P_2 is rational and all decisions made by it so far were rational and knows all previous choices. It knows that if it chooses o_1 it can get $\pi_1 = 0.06$. The choice of o_2 will give $\pi_1 = 0.10$, so P_1 chooses o_2, as marked with a thick line.

A strategy profile determined by equilibria in all subgames is the PNE of the given game. In Figure 1.3 that profile is represented by path (1:a,2:c,1:g), marked with a thick line.

Unfortunately, the approach described before is not applicable to play SBG, which rules were described before on page 15. First, there would be a problem drawing a game tree when the bargaining set has more than two elements. Moreover, to draw a tree, such as the one shown in Figure 1.3, a common knowledge about both players' payoffs is necessary. So, we have to find another way to solve our game.

Games in which players have the full knowledge on their opponents, in particular their payoffs, are called the games with *perfect information*. Moreover, according to rule 2 players do not know their opponents' payoffs, so SBG is the game with *imperfect information* [13].

The game in Example 1 is multi-stage and has been presented in Figure 1.3 in the extensive form. The main advantage of the extensive form is the possibility to present games graphically. Unfortunately this feature fails short in the case of even a slightly larger bargaining set. Let us then consider games with richer bargaining sets than in Example 1 and attempt to find Nash equilibrium for them.

1.3.2 Multi-stage games

Let us now define our game in a more formal way:

Definition 6. *A* simple bargaining game *(SBG) is a two-person multi-stage game with imperfect information, represented as:*

$$SBG = \{P, D, U, C, S, \Pi, T\} \tag{1.9}$$

where:

- set of players $P = \{P_1, P_2\}$ and P_1 begins the game.
- $D = \{\delta_1, \delta_2\}$ is a set of their respective discount factors
- $U = \{U_1, U_2\}$ is a set of their respective utility functions; we denote $U_1 \equiv U_{-2}$ and $U_2 \equiv U_{-1}$.
- $C \subset A$ is the bargaining set. Furthermore, $C_i \subset C$ will denote offers submitted by P_i and C_{-i} offers submitted by its opponent P_{-i}.
- $S = S_i \cup S_{-i}$ is a finite set of strategies of players P_i, $i = 1, 2$, at each stage of the game, where $S_i = \{s_{i,j}(k) | j = 1, \ldots, |C|,\ k = 0, 2, \ldots, 2n_{max}\}$ and $S_{-i} = \{s_{-i,j}(k) | j = 1, \ldots, |C|,\ k = 1, 3, \ldots, 2n_{max} + 1\}$, where $n_{max} = \lceil |C|/2 \rceil$ is the maximum number of stages. There are $|C|$ possible strategies in S.
- $\Pi = \langle \pi_1, \pi_2 \rangle$ is a vector of players' payoffs calculated according to Definition 2 as

$$\pi_i(o, k) = \begin{cases} \delta_i^{\lfloor k/2 \rfloor} \cdot U_i(o), & \text{if } o \in C_{-i} \\ 0 & \text{if } o \in C_i \end{cases} \tag{1.10}$$

- $T(o)$ denotes a condition for concluding the game.

$$T(o) = \begin{cases} false & \text{if } o \notin (C_i \cup C_{-i}) \\ true & \text{otherwise} \end{cases} \tag{1.11}$$

1.3.3 A naive negotiation algorithm

We have introduced Algorithm 1.2 ($SAlg$) enabling players to find a solution of the bargaining game [35]. It may be termed *naive*, as neither player makes any attempt to guess if the submitted offer may be accepted by its opponent:

1. Opening offer is the highest player's offer.
2. If player P_i receives offer o', it compares its utility with the discounted utility of its highest offer:
   (a) if $U_i(o')$ is greater or equal, incoming offer o' is accepted, otherwise
   (b) offer o' is rejected and a new offer $o'' = \arg\max_{o \in C_{-i}} U_i(o)$ is submitted.

Algorithm 1.2 may be illustrated with the following example:

Example 2. *Let two players, P_1 and P_2, choose offers from bargaining set $C = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8\}$. Their respective utilities calculated with their utility functions are $map(U_1, C) = \{0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$ and $map(U_2, C) = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, discount factors are $\delta_1 = \delta_2 = 0.9$.*

The bargaining process would run as shown in Table 1.1. In stage $n = 4$ player P_1 has already offered o_1, o_2, o_3 and o_4, so repeating them would end the game by P_1 with payoff 0. Offer o_5 has the highest value of the rest of the offers, so in move α_1^8 player P_1 should choose offer o_5 for which $U_1(o_5) = 0.4$ and $\pi_1(o_5) = \delta_1 \cdot U_1(o_5) = 0.26244$ as shown in Table 1.2.

An advantage of Algorithm 1.2 is its simplicity. However, it omits some important data which should be taken into consideration. Consider a game over bargaining set $C = \{o_1, o_2, o_3, o_4\}$, and the following players' P_1 and P_2 utilities: $map(U_1, C) = \{0.8, 0.6, 0.4, 0.2\}$ and $map(U_2, C) = \{0.2, 0.4, 0.6, 0.8\}$. Moreover, assume $\delta_1 = 0.95$ and $\delta_2 = 0.8$. Now consider that P_1 has offered o_2. Algorithm 1.2 takes into consideration the value of player's P_2 discount factor and the value of its highest offer when deciding to continue the game. Since $\delta_2 \cdot U_2(o_4) > U_2(o_2)$ as $0.8 \cdot 0.8 > 0.4$, P_2 decides to continue the game. It is because P_2 assumes the future value of the game to be higher

Algorithm 1.2: submitOffer(SAlg)

```
1  Data: set of all offers C,
2  set of received offers C_R,
3  set of sent offers C_S,
4  player's discount factor δ_1,
5  received opponent's offer o (if the procedure is called the first time the
   value is empty)
6  Result: The chosen offer c. If the opponent's offer is accepted, the
           incoming opponent's offer is returned, otherwise the
           counter-offer is returned.
7  if o = null then
8      // if it is the initial move find the highest offer
9      c ← max(C);
10     return c;
11 else
12     if o ∈ C_S then
13         // If opponent accepted the offer
14         return o;
15     C_R ← C_R ∪ {o};
16     o ← max(C_R);
17     // set of offers not presented and received by player
       C_N ← C − (C_R ∪ C_S);
18     if isEmpty(C_N) then
19         // accept the best received offer
20         return o;
21     else
22         c ← max(C_N);
23         if U_i(o) ≥ δ_i · U_i(c) then
24             // If the incoming offer is better than a
                  discounted value of future proposal
25             return o;
26         else
27             C_S ← C_S ∪ {c};
28             return c;
```

Table 1.1: An SBG example

stage	*move*	*player*	*offer*	U_1	U_2	δ_1	δ_2
0	α_1^0	P_1	o_1	0.8	0.1	$0.9^0 = 1.0000$	$0.9^0 = 1.0000$
	α_2^1	P_2	o_8	0.1	0.8	$0.9^1 = 0.9000$	$0.9^1 = 0.9000$
1	α_1^2	P_1	o_2	0.7	0.2	$0.9^1 = 0.9000$	$0.9^1 = 0.9000$
	α_2^3	P_2	o_7	0.2	0.7	$0.9^2 = 0.8100$	$0.9^2 = 0.8100$
2	α_1^4	P_1	o_3	0.6	0.3	$0.9^2 = 0.8100$	$0.9^2 = 0.8100$
	α_2^5	P_2	o_6	0.3	0.6	$0.9^3 = 0.7290$	$0.9^3 = 0.7290$
3	α_1^6	P_1	o_4	0.5	0.4	$0.9^3 = 0.7290$	$0.9^3 = 0.7290$
	α_2^7	P_2	o_5	0.4	0.5	$0.9^4 = 0.6561$	$0.9^4 = 0.6561$

than $U_2(o_2) = 0.4$. But it is wrong, because its offer o_4 is likely to be rejected. Player P_1 has relatively high discount factor and it prefers to wait, rather than accepting a less profitable offer. This example illustrates that the opponent's discount factor matters.

Table 1.2: Continuation of an SBG example

stage	*move*	*player*	*offer*	U_1	U_2	δ_1	δ_2
4	α_1^8	P_1	o_5	0.5	0.5	$0.9^4 = 0.6561$	$0.9^4 = 0.6561$

So let us then consider an algorithm that respects both players' discount factors.

1.3.4 The best response negotiation algorithm

Let us assume the following:

1. All estimations are made from the player's P_i point of view.

2. If the player's P_i move is considered, it assumes that the best response is the choice of its most valued offer. Since player P_i knows the value, it only has to find the offer which gives the highest payoff.

3. If the player's P_{-i} move is considered, we assume that its best response is also the choice of its most valued offer. However, since player's P_{-i} utility function is unknown to P_i, it can only assume that P_{-i} is rational and maximizes its payoff. Since P_i views player's P_{-i} offers equally probable, and the range of their utility values assumed by P_{-i} the same as its own range, statistically the payoff should be equal to the average of that range. We will be calculating that average with the auxiliary function:

$$\rho(o, C, f) = \frac{1}{|C|} \sum_{o \in C} f(o) \tag{1.12}$$

An alternative to calculating the arithmetic average of the range would be finding the median offer instead. Closer to the end of the game, when most of the offers from the bargaining set have been submitted and none was accepted by the opponent, choosing the median may seem to be better. On the other hand, at the beginning of the game, when most of the offers from the bargaining set have not been submitted yet the offer with the average payoff would be better. For the sake of simplicity of the algorithms developed further in this chapter we have assumed to choose the average offer.

The above brings us to the issue of choosing the best offer by the player to be submitted in the current move. The major dilemma in each move it has to resolve during the game is to decide whether to accept one of the offers already received from the opponent or choose a new one from the part of the bargaining set not attempted yet by any player. There are four specific types of moves of the player in the SBG game that have to be considered when looking for the best response strategies [33]: the *last move*, when there are no new offers left in the bargaining set, the *penultimate move*, when exactly one such offer remained, the *intermediate move*, when at least one offer has been submitted and more than one offer remains to be chosen in the bargaining set, and the *first move*, when none of the offers from the bargaining set has been submitted yet.

Lemma 1. *In the last move, $k = k_{max}$, the best response strategy for P_i is:*

$$s_i(k) = \arg \max_{o \in C_{-i}} \pi_i(o) \tag{1.13}$$

and for P_{-i} is:

$$s_{-i}(k) = \arg_{o \in C_i} [\pi_i(o) = \rho(o, C_i, \pi_i)] \tag{1.14}$$

Proof. In order to prove that Equations 1.13 and 1.14 expressing the best response strategies we consider first P_i choosing its offer with $s_i(k)$. In the last move, P_i chooses one offer from all offers that have been already presented, i.e. $C = C_i \cup C_{-i}$. Since repeating any offer from C_i (any of its own offers) gives it payoff 0, player P_i has to choose the offer from C_{-i}. Since it maximizes its payoff it must be $\arg\max_{o \in C_{-i}} \pi_i(o)$.

If the last move is made by player P_{-i}, the choice has to be made from C_i. However, the offer of the average payoff in C_i is chosen, since π_{-i} is not known to P_i. ■

Lemma 2. *In the penultimate move $k = k_{max} - 1$ the best response strategies for P_i and P_{-i} are determined respectively by Equations 1.15 and 1.16.*

$$s_i(k) = \arg \max_{o \in \{o'_i, o''_i\}} (\pi_i(o'_i), \delta_i \cdot \pi_i(o''_i)) \tag{1.15}$$

$$s_{-i}(k) = \arg_{o \in C - C_{-i}} [\pi_i(o) = \frac{\pi_i(o'_{-i}) + \delta_{-i} \cdot \pi_i(o''_{-i})}{2}] \tag{1.16}$$

where auxiliary variables o'_i, o''_i, o'_{-i} and o''_{-i} are determined by the following equations:

$$o'_i = \arg \max_{o \in C_{-i}} \pi_i(o) \tag{1.17}$$

$$o''_i = \arg_{o \in (C - C_{-i})} [\pi_i(o) = \rho(o, C - C_{-i}, \pi_i)] \tag{1.18}$$

$$o'_{-i} = \arg_{o \in C_i} [\pi_i(o) = \rho(o, C_i, \pi_i)] \tag{1.19}$$

$$o''_{-i} = \arg_{o \in (C - C_i)} [\pi_i(o) = \rho(o, C - C_i, \pi_i)] \tag{1.20}$$

Proof. In the penultimate move, a player has a choice between accepting an opponent's offer and submitting its own offer and continuing the game. Then the game will be concluded in the next stage by the opponent's move.

Equations 1.15 and 1.16 show strategies in the penultimate move from the players' P_i and P_{-i} point of view. Both equations represent the choice between accepting the opponent's offer and submitting its own offer. Since submitting its own offer means the transition to the next move, the second element of both equations has to be discounted (see the rules of the game).

In the case of the first equation the choice is made by the player P_i. Since P_i knows its utility, the choice can be expressed as the choice of the maximum utility, what is compatible with p.2 of our assumptions on page 23.

In the case of the second equation the choice is made by the player P_{-i}. Therefore the choice is expressed by the average value of the elements of the equation, what is compatible with p.3 of our assumptions on page 23.

Variables o'_i and o''_i, in Equation 1.15, are the results of Equations 1.17 and 1.18. Variable o'_i expresses the highest offer received by P_i. The offer has been chosen from C_{-i}, since set C_{-i} is the set of offers submitted by P_{-i} and received by P_i. Variable o''_i expresses the choice from the rest of all offers, i.e. $C - C_{-i}$. Since the choice is made by P_{-i}, the average is calculated.

If the penultimate move is made by P_{-i} the latter has to choose between accepting offer from P_i and submitting its own. It is explained by Equation 1.16. Variables o'_{-i} and o''_{-i}, in Equation 1.16, are the results of

Equations 1.19 and 1.20. Variable o'_{-i} expresses the highest offer received by P_{-i} observed from the player's P_i point of view. Since P_i does not know the player's P_{-i} utility function, according to our assumptions, the average had to be calculated. The choice has been made from offers belonging to C_i, because C_i is the set of offers submitted by P_i and received by P_{-i}.

Variable o''_{-i} expresses the choice from the rest of all offers, i.e. $C - C_i$. Although the choice is made by P_i, the calculation is made from the player's P_{-i} point of view, so the average from $C - C_i$ has to be calculated.

Equation 1.15 determines the rational player's P_i choice, thus it is the best response. Equation 1.16 calculates the payoff resulting from the rational strategy, utilized by the recursive expressions presented in the intermediate move, what is necessary to obtain the best response. Therefore $s_i(k)$ and $s_{-i}(k)$, where k is the penultimate move, are the Nash equilibrium.

■

Lemma 3. *In the intermediate move, $0 < k < k_{max} - 1$, the best response strategies for P_i and P_{-i} are determined respectively by Equations 1.21 and 1.22:*

$$s_i(k) = \arg \max_{o \in \{o'_i, o''_i\}} (\pi_i(o'_i), \delta_i \cdot \pi_i(o''_i)) \tag{1.21}$$

$$s_{-i}(k) = \arg_{o \in C - C_{-i}} [\pi_i(o) = \frac{\pi_i(o'_{-i}) + \delta_{-i} \cdot \pi_i(o''_{-i})}{2}] \tag{1.22}$$

where auxiliary variables o'_i, o''_i, o'_{-i} and o''_{-i} are determined by the following equations:

$$o'_i = \arg \max_{o \in C_{-i}} \pi_i(o) \tag{1.23}$$

$$o''_i = \arg_{o \in C'} [\pi_i(o) = \max(1 - \delta_{-i})\pi_i(o) + \delta_{-i} \cdot U_i(s_{-i}(k+1))] \tag{1.24}$$

$$o'_{-i} = \arg_{o \in C_i} [\pi_i(o) = \rho(o, C_i, \pi_i)] \tag{1.25}$$

$$o''_{-i} = \arg_{o \in C'} [\pi_i(o) = (1 - \delta_i)\rho(o, C') + \delta_i \cdot \pi_i(s_i(k+1)))] \tag{1.26}$$

where $C' = C - (C_i \cup C_{-i})$ includes offers not yet submitted by any player.

Proof. The only difference between formulas concerning penultimate move and ones concerning the intermediate move concerns two auxiliary equations:

Equation 1.24 and 1.26, so we will need to explain only the rationale of the equations, since we regard the rest of equations as explained.

Variable o_i'' calculated by Equation 1.24 reflects later the fact of continuing the game by P_i and rejecting player's P_{-i} offer. In the case of the penultimate move, P_{-i} had to accept player's P_i offer by the fact that the player's P_{-i} move was the last one. In the case of the intermediate move, the offer may be accepted as well as rejected. Equation 1.24 says that continuing the game can be calculated as the weighted sum of the average value of not yet submitted offers and the value of recursive calling Equation 1.22.

The weights are expressed by the opponent's discount factor and its completion to 1, i.e. δ_{-i} and $1 - \delta_{-i}$.

In order to illustrate the impact of the opponent's discount factor on the fact of the accepting or rejecting an offer consider two hypothetical situations. The first is that opponent's discount factor $\delta_{-i} \approx 0$. Then P_{-i} will accept each player's P_i offer, since in the next stage of the game its payoff will be close to 0 (recall that the player's P_{-i} payoff is calculated by multiplying the utility of its offer by δ_{-i}). Now consider that $\delta_{-i} \approx 1$. Then P_{-i} will reject almost every player's P_i offer, except for the player's P_{-i} most valued one.

Therefore Equation 1.24 is built in the way that the greater the value of δ_{-1} the greater value of continuing the game and the less the value of δ_{-1} the greater the value of accepting the offer submitted by P_i.

Variable o_{-i}'' explained in Equation 1.26 expresses the value of continuing the game by P_{-i} and rejecting player's P_i offer. Equation 1.26 says that continuing the game can be calculated as the weighted sum of the average value of not yet submitted offers and the value of recursive calling Equation 1.21. The weights are created from δ_i by analogy to the weight created in Equation 1.24.

The remaining equations have been explained describing the penultimate move. So, similarly, Equation 1.21 determines the rational player's P_i choice, thus it is the best response. Equation 1.22 calculates the payoff resulting from the rational strategy, utilized by the recursive call in the both equations. Therefore $s_i(k)$ and $s_{-i}(k)$, where k is the intermediate move, are the Nash equilibrium.

■

Lemma 4. *In the first move the best response strategy is determined by the equation:*

$$s_i(0) = \arg\max_{o \in C} U_i(o) \tag{1.27}$$

Proof. Because player P_i makes the choice, its highest utility is calculated according to p.2 of the list on page 23. Since this is the first choice, the

entire bargaining set C is taken into consideration. In result Equation 1.27 is obtained. ■

Algorithm 1.3 implements the best response strategies, introduced by Lemmas 1– 4. Its use, however, may be limited to devices with sufficient memory resources, due to its recursive character. For mobile devices some possible simplifications may have to be considered. It may be readily seen that Equation 1.26 involves the recursive call $\delta_{-i} \cdot U(s_{-i}(k+1))$.

Let us introduce function

$$\lambda(k, \delta_i, \delta_{-i}) = \begin{cases} (1 - \delta_{-i}) & \text{if } k = 1 \\ (1 - \delta_{-i} \cdot \lambda(k-1, \delta_{-i}, \delta_i)) & \text{otherwise} \end{cases} \tag{1.28}$$

and replace the recursive part of o_i'' in Equation 1.24 by

$$\chi = \arg_{o \in C' - \{\tilde{o}\}}[\pi_i(o) = \rho(o, C' - \{\tilde{o}\}, \pi_i)] \tag{1.29}$$

where $\tilde{o}$ is the offer submitted in non-recursive part of Equation 1.24. Then we get

$$o_i'' = \arg_{o \in C'}[\pi_i(o) = \max \lambda(l, \delta_i, \delta_{-i})\pi_i(o) + (1 - \lambda(l, \delta_i, \delta_{-i})) \cdot \chi] \tag{1.30}$$

where $l = k_{max} - k - 1$, which is less computationally demanding than Formulas 1.24 and 1.26. So after substituting o_i'' from Equation 1.30 in place of o_i'' from Equation 1.24 into Formula 1.21 we receive the required simplification of our solution. The newly obtained version of Formula 1.21 does not include any recursive calls to Formula 1.22, and so it is self-reliant. If the utility values of the offers not yet submitted are distributed evenly, using the average utility instead of the recursive estimate of the opponent's offer in Equation 1.21 is a reasonable approximation.

Algorithm 1.4 (*EAlg*) implements the best response negotiation expressed by the substitution of Formula 1.30 into Formula 1.21; it has been used in the experiments described later in the Thesis. The algorithm calls procedure `discount()` which implements Equation 1.28. We will present the procedure in Appendix B. For short, the work of *EAlg* relies on that:

- Player P_2 receives offer o submitted by P_1 and decides on its acceptance.
- Payoff $\pi_2(o)$ is compared with discounted payoff $\pi_2(o_i'')$ (the result of calculating Equation 1.30). If $\pi(o) > \pi_2(o_i'')$ then o is accepted,
- otherwise P_2 submits its own offer.

Algorithm 1.3: submitOffer(equilibrium)

```
1  Data: Bargaining set C, received offers C_R, sent offers C_S, player's
         discount factor δ_i, opponent's discount factor δ_{-i}, bargaining
         set size n, current move k, incoming offer o
2  Result: The value of the chosen offer.
3  C_R ← C_R ∪ {o};
4  if ¬(k mod 2) then                                   /* Player P_i */
5      o ← max(C − (C_S ∪ C_R));
6      c ← max(C − C_R);
7      if k ≤ n − 2 then                  /* the intermediate move */
8          contValue ← /* Equations 1.23 and 1.24                */
           getContValue(estimateOffer(c,k + 1),δ_{-i});
9          if o ≥ δ_i·contValue then                /* Equation 1.21 */
10             return o;
11         else
12             C_S ← C_S ∪ {c};
13             return contValue;
14     else if k = n − 1 then              /* the penultimate move */
15         avgValue ←getAvgVal(C − (C_R));
16         if o ≥ δ_i·avgValue then                 /* Equation 1.15 */
17             return o;
18         else
19             C_S ← C_S ∪ {c};
20             return avgValue;
21 else                                              /* Player P_{-i} */
22     avgRecValue ← getAvgVal(C_R) ;
23     if k ≤ n − 2 then                  /* the intermediate move */
24         // Equation 1.26
25         contValue ←
           getOpContValue(estimateOffer(getAvgVal(C − (C_S ∪
           C_R)),k + 1),δ_i);
26         // Equations 1.25 and 1.22
27         sentValue ←(avgRecValue +δ_{-i}·contValue)/2;
28         C_S ← C_S + getAvgOffer(sentValue,C − C_S);
29         return sentValue;
30     else if k = n − 1 then              /* the penultimate move */
31         // Equation 1.16
32         avgValue ←avgRecValue +δ_{-i}max(C − C_R))/2 ;
33         C_S ← C_S ∪ {getAvgOffer(avgValue,C − C_S)};
34         return avgValue;
```

Algorithm 1.4: submitOffer(EAlg)

```
1  Data: set of all offers C, set of received offers C_R, set of sent offers
         C_S, player's discount factor δ_1, opponent's discount factor δ_2,
         received opponent's offer o (if the procedure is called the first
         time the value is empty), current move number k
2  Result: The chosen offer c. If the opponent's offer is accepted, the
           incoming opponent's offer is returned, otherwise the
           counter-offer is returned. If the process is prolonged, an
           updated value of k is returned, too.
3  if o = null then
4      // if it is the initial move find the highest offer
5      c ← max(C);
6      return c;
7  else
8      C_N ← C − (C_R ∪ C_S);
9      c ← max(C_N);
10     if c = o then
11         // accept the opponent's offer
12         return o;
13     else
14         // updating the content of the sets of offers
15         C_N ← C_N − {c, o};
16         C_N ← C_N − {o};
17         C_R ← C_R ∪ {o};
18         o ← max(C_R);
19         U_sum ← 0;
20         /* Function expressed by Equation 1.28.Its first
              argument is number l appearing in Equation 1.30 */
21         λ ← discount(|C_N| − k − 1,δ_1, δ_2);
22         foreach o_current in C_N do
23             U_sum ← U_sum + δ_1·(λ · o_current+(1 − λ)·o);
24         // if more than half offers is better than
              opponent's offer
25         if (U_sum/|C_N|) > U(o) then
26             // Continue the game and present your offer
27             return c;
28         else
29             // Accept opponent's offer
30             return o;
```

1.4 The economic model

To this point we have pursued finding a solution to Equation 1.7 in a game theoretic fashion, by modeling the negotiation process between the active document and its execution device with a simple bargaining game. Because of that our document-agents may be considered *economic agents*. Alternatively they could be modeled as *cognitive agents*, which communicate preferences one to another until a solution satisfying both parties could be found by the means of programming in logic. This approach, however, would require defining a complete logic system involving relations and inference rules representing respectively agent intentions or goals and arguments exchanged until an agreement can be reached. One example of such a practical system is PERSUADER, capable of resolving conflicts by agents persuading each other to change their intentions [39].

Building a system enabling interpretation of logical expressions is quite demanding, and such a system may require relatively more computational power than quite simple computation of the payoff function. This is an important issue, since our document-agents and their execution devices (most often personal mobile devices) have limited capabilities in terms of both, the available RAM and CPU power.

Document agents considered in this Thesis can travel in a distributed system and perform various activities with the help of their internal functionality in the execution contexts provided by visited devices. It would be highly desired to exploit that capability of proactive documents to augment them not only with the negotiation capability, but also machine learning.

1.5 An intelligent negotiation algorithm

The best offer negotiation Algorithms 1.3 and 1.4 have a certain limitation, related to the fact that the player does not know its opponent's payoff function. Estimates based on the average payoff of the player have been used in Formulas 1.14– 1.26, under the assumption that the opponent is sufficiently rational not to choose any 'worse offer' strategy instead. The question arises, whether it would be possible to find a better method for predicting the opponent's offers than 'throwing a dice' – given the fact that π_{-i} is not known to P_i. Later in the Thesis we will investigate the possibility of using machine learning techniques for that.

Usually documents-agents can perform their activities many times. Respective contexts configurations can thus repeat themselves as the total number of devices in a system is practically finite. In consequence that various

bargaining processes may be executed repeatedly arbitrary many times. We consider such situations *repeated encounters* of negotiating agents. A question arises whether an agent could learn from its previous encounters how to negotiate in the current one – in particular, whether it can recognize the current execution context based on the negotiation history. If so, a document agent should reach agreements faster, without repeating the entire negotiation process and consuming less resources of the execution device.

Earlier in the chapter we have introduced negotiation algorithms that approximate equilibrium strategies based on the assumption of equal distribution of offer choices by opponents. The repeated encounters give the possibility of approximating them better, based on the recorded history of negotiations concluded by them before.

Given the development of this Chapter the outline of the rest of the Thesis is the following. In Chapter 2 we investigate several candidate AI approaches which may provide a vehicle to improve the basic best offer negotiation algorithm developed in Subsection 1.3.4. Next in Chapter 3 coding of multi-issue offers will be introduced to enable representation of the execution contexts that may be handled by the negotiation algorithm. In Chapter 4 we will present the intelligent negotiation algorithms that uses neural networks to recognize devices and predict contracts. Finally in Chapter 5 results of the experiments with negotiation algorithm developed in the Thesis will be evaluated, and in Chapter 6 the development of the Thesis will be compared and contrasted to existing state of the art in the area of bilateral multi-issue negotiation models and intelligent negotiation algorithms.

Chapter 2

Machine learning and artificial intelligence strategies

In the current chapter we will consider the concept of using artificial intelligence techniques to answer two questions: how to classify groups of execution devices and how to discover sorting of offers in the bargaining set reflecting preferences of the player's opponent.

Let us focus on the first problem. The task of dividing a set of objects into smaller subsets in such a way that objects in each subset are similar one to another with some respect is called *clustering*. The way how we define their similarity implies a specific method of clustering.

Recall from Definition 1 that offer o consisted of m values, $\langle v_1, v_2, \ldots, v_m \rangle$, each one selected from the respective attribute set, i.e. $v_k \in A_k$, $k = 1, \ldots, m$. Let us index each possible value that may occur in any offer as follows:

$$\begin{array}{rl} v_1 & \in \{a_{11}, a_{12}, \ldots, a_{1|A_1|}\} \\ v_2 & \in \{a_{21}, a_{22}, \ldots, a_{2|A_2|}\} \\ & \vdots \\ v_m & \in \{a_{m1}, a_{m2}, \ldots, a_{m|A_m|}\} \end{array} \tag{2.1}$$

Consider bit fields ω_i of length $|A_i|$, $i = 1, \ldots, m$, indicating which value of the respective set A_i occurs in the offer; if the j-th bit of ω_i, $j = 1, \ldots, |A_i|$, is set to 1 it indicates that value a_{ij} occurs in the offer, otherwise it is set to 0.

By concatenating all respective bit fields, we get the bit word

$$\beta = \omega_1 \omega_2 \ldots \omega_m \tag{2.2}$$

of length $n = |A_1| + |A_2| + \ldots + |A_m|$, which we call the *occurrence vector*. By B we denote a set of all possible words β for A. We will use occurrence vectors

to measure how close offers are one to another in the space of offers. We use the term 'vector' because of the data type used in the implementation of the algorithms proposed in this Thesis (in high-level programming languages the type `vector` stands for a collection similar to the array but dynamically allocated and resizable).

Any bit word ω_i corresponds then to the vector, which elements indicate which element a_{k_i} of attribute set A_k occurs in the offer. So, ω_i would canonically contain exactly one bit set to 1, with all other bits set to 0. To subsume alternative values of one offer we may relax that limitation and allow more bits of ω_i to be set to 1.

In Chapter 1 we have defined the structure of offers. We have described the negotiation process and the role of the order of submitted offers. In the current chapter we will explain specific methods of grouping offers by their structure as well as the order of their appearance. We begin our investigations from the methods of grouping offers according to how they are built.

2.1 The k-means algorithm

A popular algorithm for grouping elements in a set with regard to their similarity, measured formally with some distance metric, is the k-means algorithm [42]. The following definition of the Hamming distance [22] may be used:

Definition 7. *Distance*

$$d : B \times B \to \mathbb{Z}_{\geq 0}$$

calculated for any pair of m-bit words $\beta_1, \beta_2 \in B$ is a number of bit positions in which the two words differ.

It may be readily seen that in order to calculate a number of bits any two words β_1, β_2 differ, it suffices to calculate the number of bits in $\beta_1 \oplus \beta_2$ where operator $\oplus$ denotes bitwise XOR (exclusive or) of β_1 and β_2. We will denote a distance between β_1 and β_2 by $d(\beta_1, \beta_2)$

We can eliminate parameter m (length of the bit word) from further consideration by normalizing the distance metric. With $\Delta = \frac{d(\beta_1,\beta_2)}{m}$ we get a range of distances $[0, 1]$, where 0 is a distance between two identical words and 1, respectively, between two words different on all m positions.

We define *input data matrix* $\mathbf{R}$ a matrix, which rows are built of occurrence vectors.

2.1.1 An overview of the algorithm

Procedure `kMeans()` specified by Algorithm 2.1 calculates the k-means algorithm [58].

There are k target clusters and $n = |A|$ of m-bit words that have to be divided into k clusters. The algorithm relies on modifying contents of the clusters according to k-bit vectors, called *centroids*, which provide axis of each division. New centroids are updated until they reach the termination condition.

We start by initializing the first k centroids, i.e., the first k vectors from vector `data`. Then we have to create the first set of clusters according to the new centroids. It is implemented by procedure `updateClusters()` calculated by Algorithm 2.2 (the pseudo-code presented in the Chapter specifies the algorithms implemented by us for the experiments described in Chapter 5). For each data vector its distance to each centroid is examined. Then, the nearest centroid is chosen. Vectors, for which the same centroid has been chosen, belong to the same cluster.

Next, the following operation is performed: in each cluster the new centroids are determined by calculating the mean distance to each vector in the cluster. It is implemented by procedure `updateCentroids()` calculated by Algorithm 2.5.

After that procedure `updateClusters()` is called again. The clusters are re-created according to the new created centroids. As long as centroids change, procedures `updateClusters()` and `updateCentroids()` are called.

2.1.2 The algorithm at work

The main purpose of data clustering is the division of the examples into categories in such a way that it assures their maximal similarity in each category [7]. The pseudo-code shown in the current chapter refers to the real code which was utilized to build the software (among others in procedure `kMeans()`), in the CD Rom included in this Thesis.

Below we exercise the k-means algorithm applied to analyzing sets of offers from the space of offers by negotiating parties. Table 2.1 shows twelve bit words. They belong to one of three classes: 0, 1 or 2, what is indicated by numbers in column *class*. The classes correspond to the device or document classes which a given offer belongs to. We would like to see if the k-means algorithm can divide offers in a meaningful way. If it can, its result should agree with the expected class membership presented in Table 2.1.
The division into the three classes is a consequence of the following facts related to Table 2.1:

Algorithm 2.1: kMeans()

```
1  Data: input data data,
2  vector of clusters clusters,
3  the number of clusters kcount
4  // The first k-points - centroids - are the first k points
       from data vector.
5  initializeCentroids (centroids);
6  // The first clusters are appointed according to the first
       centroids.
7  updateClusters (centroids, data, clusters);
8  repeat
9  |   // Copy the current centroids.
10 |   oldCentroids ← centroids;
11 |   // Create new centroids
12 |   updateCentroids (centroids,clusters);
13 |   // Update clusters according to the new centroids.
14 |   updateClusters (centroids,data, clusters);
15 |   // Until centroids stop changing.
16 until oldCentroids = centroids;
```

Algorithm 2.2: updateClusters()

```
1  Data: current centroids,
2  input data data,
3  vector of clusters clusters
4  foreach dataRow in data do
5  |   chooseCentroid (centroids, dataRow, clusters);
```

Algorithm 2.3: chooseCentroid()

```
1  Data: current centroids,
2  input data dataRow,
3  vector of clusters clusters
4  distance ← 100000;
5  properIndex ← −1;
6  for i ← 0 to centroids.size do
7      temp ← getDistance (dataRow,centroids[i]);
8      if temp < distance then
9          distance ← temp;
10         properIndex ← i;
11     i ← i + 1;
12 clusters[properIndex].insert(dataRow);
```

Algorithm 2.4: updateCentroids()

```
1  Data: current centroids,
2  vector of clusters clusters
3  foreach dataRow in data do
4      updateCentroid (centroids, clusters);
```

Algorithm 2.5: updateCentroid()

```
1  Data: current centroid,
2  cluster cluster
3  initializeCentroid (0);
4  foreach bits in cluster do
5      for i ← 0 to bits.size do
6          if bits[i] > 0 then
7              centroids[i] ← centroids[i] + 1;
8          else
9              centroids[i] ← centroids[i] − 1;
10 foreach bit in centroid do
11     if bit > 0 then
12         bit ← 1;
13     else
14         bit ← 0;
```

Table 2.1: Desired classification of data

l	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	*class*
1	1	0	0	1	0	0	1	0	0	1	0
2	1	0	0	0	0	1	0	0	0	1	
3	0	0	0	1	0	1	1	0	0	0	
4	1	0	0	1	0	1	1	0	0	1	
5	1	1	0	1	0	0	0	0	0	0	1
6	1	0	1	1	0	0	0	0	1	0	
7	1	1	1	0	0	0	0	0	1	0	
8	1	1	1	1	0	0	0	0	1	0	
9	0	1	1	1	0	0	0	0	1	0	
10	0	0	1	0	1	1	1	1	0	0	2
11	1	1	0	0	0	1	1	1	0	0	
12	0	1	0	0	0	1	1	1	0	0	

1. Each row in is occurrence vector $\beta = \omega_1$, $\omega_1 = b_1 \dots b_{10}$. Since each occurrence vector is a bit word, we will use both terms interchangeably.

2. Each β is a logical sum (a bitwise OR) of occurrence vector of all offers in the related bargaining set.

3. The rows which fields $b_2 = b_3 = b_5 = b_9 = 0$ belong to class 0.

4. The rows which fields $b_5 = b_6 = b_7 = b_{10} = 0$ belong to class 1.

5. The rows which fields $b_4 = b_9 = b_{10} = 0$ belong to class 2.

If we want to use the k-means algorithm, we have to choose the value of k, which is the number of partitions we want to reach. Then the first step is the choice of the first k *centroids*; recall that a centroid is a bit word which is the center of some group of bit words. The way of calculating the centroids will be shown later, after introducing necessary definitions.

In our notation we adopt the following convention: given matrix $\mathbf{A}$, its i-th row $i = 1, \dots, n$, will be denoted as $A_{i\bullet}$, while its j-th column, $j = 1, \dots, m$, as $A_{\bullet j}$. Each row and column at matrix $\mathbf{A}$ are respectively m- and an n-vectors.

Definition 8. Mean ϕ *is function*

$$\phi : B \to \{0, 1\} \tag{2.3}$$

which maps a bit vector β *into* $\{0, 1\}$ *and indicates whether bit* 0 *or* 1 *occurs more often in* β*, where* B *is a set of bit words.*

Let us denote the number of bits equal to 1 in any bit word β as $\eta(\beta)$, and its length, respectively, by $\iota(\beta)$. Then function ϕ may be calculated as:

$$\phi(\beta) = \begin{cases} 1 & \text{if } \frac{\eta(\beta)}{\iota(\beta)} > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Definition 9. *Let $\boldsymbol{L} \subset \boldsymbol{R}$, $\boldsymbol{L} \neq \emptyset$ be a matrix consisting of any subset of rows in $\boldsymbol{R}$.* Centroid *is the bit vector Φ built of the mean value of each column of matrix $\boldsymbol{L}$:*

$$\Phi = \langle \phi(L_{\bullet 1}), \phi(L_{\bullet 2}), \ldots, \phi(L_{\bullet m}) \rangle \tag{2.5}$$

where $m \in \mathbb{N}$ is the length of any row $L_{i\bullet}$

Definition 10. *Consider subset $\{\Phi_1, \Phi_2, \ldots, \Phi_k\}$ including any $k > 1$ rows of $\boldsymbol{R}$. For each Φ_i, $i = 1, \ldots, k$, we select a subset of rows in $\boldsymbol{R}$ which are the closest to Φ_i with regard to the distance metric d. We will call each Φ_i a* centroid *and the related set of rows Ψ_i a* cluster.

At the beginning of the process the centroids can be chosen randomly – it can be any bit word from the group we want to partition. Let it be the first three bit words from Table 2.2.

Table 2.2: Initial k centroids

β_i	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
1	1	0	0	1	0	0	1	0	0	1
2	1	0	0	0	0	1	0	0	0	1
3	0	0	0	1	0	1	1	0	0	0

In order to assign β to the appropriate cluster Ψ, distances between β and each respective centroid Φ_i, $i = 1, \ldots, k$, should be compared and the shortest one chosen, as shown by Formula 2.6:

$$\beta \in \Psi_i \Leftrightarrow \underset{j \leq k, j \neq i}{\forall} d(\beta, \Phi_j) \geq d(\beta, \Phi_i). \tag{2.6}$$

Next, new clusters are built on the basis of the created centroids. It is done by Algorithm 2.1 in the following way: there are k initially empty clusters, each of them associated with a relevant centroid. For each bit word β distance d between β and each centroid Φ_i, $i = 1, \ldots, k$ in group of k centroids, is calculated; β is assigned to the cluster to which centroid distance d is the smallest.

Table 2.3: Distances of bit words from clusters

Ψ_i	*Bit words* β											
	1	2	3	4	5	6	7	8	9	10	11	12
1	0	2	3	1	3	4	6	5	6	7	5	6
2	2	0	4	2	4	5	5	6	7	6	4	5
3	3	4	0	2	4	5	7	6	7	4	4	3

Distances between bit words listed in Table 2.1 and centroids listed in Table 2.2 are listed in Table 2.3; it can be seen for example that $d(\Psi_1, \beta_1) = 0$, while $d(\Psi_2, \beta_3) = 4$ and $d(\Psi_3, \beta_9) = 7$.

Based on the table of distances (like the example Table 2.3) Algorithm 2.1 can build clusters corresponding to centroids shown in Table 2.2; for example, the cluster for centroid Ψ_1 in Table 2.2 will contain bit words shown in Table 2.4.

Table 2.4: Members of cluster Ψ_1 (first round)

l	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
1	1	0	0	1	0	0	1	0	0	1
4	1	0	0	1	0	1	1	0	0	1
5	1	1	0	1	0	0	0	0	0	0
6	1	0	1	1	0	0	0	0	1	0
8	1	1	1	1	0	0	0	0	1	0
9	0	1	1	1	0	0	0	0	1	0

A new centroid is chosen in the next round based on the cluster created in the previous one, using the mean function ϕ defined before: if the number of 1-bits in a given column is greater than the half of the number of all its bits function ϕ returns 1, otherwise 0.

Let us look at column b_1 in Table 2.4. All columns consist of six elements. In column b_1 in Table 2.4 there are $\eta(L_{\bullet 1}) = 5$ bits set to 1 and according to Formula 2.4, $\phi(L_{\bullet 1}) = 1$. Next, for the next column b_2, $\eta(L_{\bullet 2}) = 3$ and $\phi(b_2 L_{\bullet 2}) = 0$. In that way we can create three new centroids shown in Table 2.5.

In the second round, again the respective clusters for the new centroids in Table 2.5 are found. Distances between the new centroids and all bit words are presented in Table 2.6.

Now, when distances are calculated, the new clusters can be determined. They are chosen according to the nearest centroid, as was shown earlier. The new clusters are listed in Table 2.7.

Table 2.5: Next k centroids (second round)

β_i	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
1	1	0	0	1	0	0	1	0	0	0
2	1	1	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	1	1	1	0

Table 2.6: Distances of bit words from Table 2.5

Ψ_i	*Bit words* β											
	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	3	3	1	2	4	3	4	7	5	6
2	5	2	4	4	2	5	3	4	5	6	2	3
3	5	4	2	5	6	6	7	8	7	2	2	1

Table 2.7: Clusters found in Table 2.1

Ψ_i	β_i
1	$1, 4, 5, 6, 8, 9$
2	$2, 7, 11$
3	$3, 10, 12$

Because clusters determined in this step do not differ from the clusters calculated in the earlier step, the algorithm terminates and returns them as the result.

Unfortunately, the results listed in Table 2.1 do not match data presented in Table 2.7; contents of clusters $\Psi_1, \ldots, \Psi_3$ do not correspond to device classes presented originally in Table 2.1.

The unsatisfactory result of the above clustering exercise stems from the problem of finding a proper definition of the mean function ϕ given by Formula 2.4, which may require a specific a priori knowledge on the possible structure of bit words coding attribute values in each respective offer. Gaining such a knowledge may require analyzing a sufficiently large set of previously agreed contracts or even a set of all possible contracts. Once defined, it may be used for clustering as demonstrated in the example. For this deficiency, this approach would be impractical for solving the problem of classifying categories of similar offers raised in the previous chapter; it is not clear if it is possible in general to define function ϕ in such a way that it will suite all conflicting situations between active document agents and their execution devices. Therefore we would like to investigate methods allowing us to *derive* functions that can separate offers into categories.

2.2 Decision trees

Deriving of functions that can separate offers into categories is the matter of supervised learning. It consist of two stages: *training* to derive classification rules based on predefined data and next *applying* these rules to other data. In the first stage occurrence bit words β that code offers observed in the past will be associated with predefined *classes* of offers based on our understanding of execution contexts. In the second stage bit words coding offers not classified yet will be assigned to one of the predefined classes based on the classification rules developed in the first stage.

In Chapter 3 we will present a detailed structure of offers based on concrete options of services provided by real execution devices. At this point let us just acknowledge the fact that in some cases two bit words shall be considered different because of just one bit value, while in other cases only the specific bit pattern would be required to differentiate them. Thus instead of grouping data by measuring their average distance, it can make more sense to teach agents successively which bits really matter.

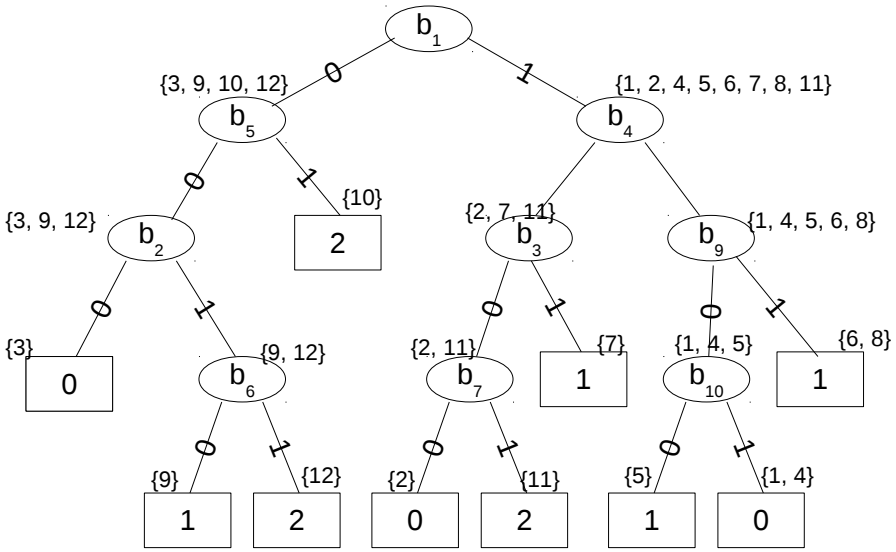


Figure 2.1: Decision tree built of the data from Table 2.1

Learning based on a decision tree relies on building a tree based on training data [23]. An example decision tree T has been built of the data from Table 2.1 is presented in Figure 2.1. It contains elements of the following kinds:

- Leaf L, which is a terminal node of tree $\mathbb{T}$. It has one field, class of index $i \in \mathbb{Z}_{\geq 0}$.
- Edge E, which connects nodes with descendant nodes. Each edge is labeled with an appropriate *category* F. *Category* F is the value of the corresponding attribute value bit, so $F \in \{0, 1\}$.
- Node N, which has descendant nodes connected with edges. The name of the node is an attribute name, held in the respective fields:

Besides the elements listed above, some numerical values in braces that label nodes of the tree, are shown in Figure 2.1. They are the numbers of rows in Table 2.1, which are used by the process of building the tree, specified formally by Algorithm 2.6. Procedure `addNode()` works in the following way:

1. Gets the most important attribute a in the data table,
2. Creates new node N and assign it the name of attribute a,
3. If node N is not the root, adds N to the parent node, connects it using edge E and assigns to E category F, which is the value of attribute a,
4. Divides the table into so many subtables as many values a given attribute may have (in our case the number is always *two*),
5. Proceeds to call procedure `addNode()` on each subtable,
6. Otherwise creates a leaf and assigns it the name of the class.

Two points have to be made here. One is that the termination condition is satisfied when each row in the table belongs to the same class. Another is that the most important attributes have been mentioned. In our case, the criterion of the most importance is the *information gain*; we will measure it with *entropy*.

2.2.1 Creation of the tree

Let us introduce the following notions (without losing generality we will consider matrix $\mathbf{R}$ a set, since in the real implementation we have used generic types, (a collection may be generalization of an array as well as a set). Since matrix $\mathbf{R}$ is an array of arrays, it can be easily converted to a set:

- R^c is the subset of training set R, consisting of elements of a class of index c. For example, R^2 means subset of set R which elements belong to class of index 2,

Algorithm 2.6: addNode

```
1  Data: given matrix matrix,
2  given node node
3  terminal node terminalNode
4  newMatrices ← divideMatrix (matrix);
5  foreach newMatrix in newMatrices do
6      if hasMoreThanOneClasses (newMatrix) then
7          node ← newNode ();
8          setNodeName (node,getHighestEntropy (newMatrix));
9          addNode (newMatrix);
10     else
11         terminalNode ← newNode ();
12         setNodeClass (terminalNode,newMatrix);
```

- R_{b_iF}, $i = 1, \ldots, M$, M is the length of bit word β, $F \in \{0, 1\}$, is the subset of the training set R, which contains only these rows, which have the attribute value bit $b_i = F$; for example, R_{b_41} is the subset of R where bit b_4 has value 1.

- $R^c_{b_iF}$ is the subset of training set R, containing only the rows, which belongs to class of index c that have attribute value bit $b_i = F$.

Entropy $I(b_i)$ of attribute value bit b_i is a sum of *partial entropies* $E_F(b_i)$ of subsequent categories F, calculated as:

$$I(b_i) = \sum_F E(b_i) \cdot \frac{|R_{b_iF}|}{n} \tag{2.7}$$

where n is a number of rows in matrix R.

Partial entropy $E_F(b_i)$ is a sum of probabilities multiplied by the logarithm of the probabilities of finding data of a given class of index c among subset of data of one of the categories R_{b_iF}, calculated as:

$$E_F(b_i) = -\sum_c \frac{|R^c_{b_iF}|}{|R_{b_iF}|} \log_2 \frac{|R^c_{b_iF}|}{|R_{b_iF}|} \tag{2.8}$$

For example, let us calculate the value of entropy for data from column b_1 of Table 2.1 considered before. The number of training examples is $n = 12$. The length of vector made of non-occurrence of attribute b_1 (number of zeros in the column) is $|R_{b_10}| = 4$. The occurrence vector length is $|R_{b_11}| = 8$.

The lengths of non-occurrence vectors for respective classes 0, 1 and 2 are: $|R^0_{b_10}| = 1$, $|R^1_{b_10}| = 1$ and $|R^2_{b_10}| = 2$. So, the value of entropy $E_0(b_1)$ due to non-occurrence of the attribute value bit b_1 is:

$$E_0(b_1) = \frac{|R^0_{b_10}|}{|R_{b_10}|} + \frac{|R^1_{b_10}|}{|R_{b_10}|} + \frac{|R^2_{b_10}|}{|R_{b_10}|} = -\frac{1}{4} \cdot \log_2 \frac{1}{4} - \frac{1}{4} \cdot \log_2 \frac{1}{4} - \frac{1}{2} \cdot \log_2 \frac{1}{2} = 1.5$$

Further, the length of vector made of occurrence of attribute b_1 (number of ones in the column) is $|R_{b_11}| = 8$. The lengths of occurrence vectors for respective classes 0, 1 and 2 are: $|R^0_{b_11}| = 3$, $|R^1_{b_11}| = 4$ and $|R^2_{b_11}| = 1$. So, the value of entropy $E_1(b_1)$ due to the occurrence of attribute b_1 is:

$$E_1(b_1) = \frac{|R^0_{b_11}|}{|R_{b_11}|} + \frac{|R^1_{b_11}|}{|R_{b_11}|} + \frac{|R^2_{b_11}|}{|R_{b_11}|} = -\frac{3}{8} \cdot \log_2 \frac{3}{8} - \frac{4}{8} \cdot \log_2 \frac{4}{8} - \frac{1}{8} \cdot \log_2 \frac{1}{8} = 1.4056$$

Finally, entropy is a weighted sum of entropies $E_0(b_1)$ and $E_1(b_1)$:

$$I(b_1) = E_0(b_1) + E_1(b_1) = \frac{4}{12} \cdot 1.5 + \frac{8}{12} \cdot 1.4056 = 1.437067.$$

Consider the following exercise of building a tree of data from Table 2.1; we will use for that Algorithm 2.6 calculated by procedure `addNode()`:

1. The root is created; it is not a leaf, since there are more classes.
2. The attribute value bit with the highest entropy is chosen. It is b_1, so the root is assigned b_1.
3. Because b_1 can have two possible values, two subtables are created. These are Table 2.8 for category (value of attribute) $F = 0$ and Table 2.9 for $F = 1$.

Procedure `addNode()` is called for each of these subtables.

Table 2.8 is the subtable of Table 2.1 made of rows 3, 9, 10 and 12.

Table 2.8: Table for $b_1 = 0$

l	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	$class$
3	0	0	0	1	0	1	1	0	0	0	0
9	0	1	1	1	0	0	0	0	1	0	1
10	0	0	1	0	1	1	1	1	0	0	2
12	0	1	0	0	0	1	1	1	0	0	

Calling `addNode()` implies the following:

1. The new node is created. It is not a leaf, since there are more classes. The node is connected to its parent using edge, labeled with category $F = 0$.

2. The attribute value bit with the highest entropy is b_5 and it is the label of the new node.

3. Two subtables are created, one for $F = 0$ for rows where $b_5 = 0$, i.e. rows 3, 9 and 12, and another for category $F = 1$, for rows where $b_5 = 1$, i.e. row 10. Procedure `addNode()` is called for each of these subtables.

Table 2.9: Table for $b_1 = 1$

l	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	*class*
1	1	0	0	1	0	0	1	0	0	1	0
2	1	0	0	0	0	1	0	0	0	1	
4	1	0	0	1	0	1	1	0	0	1	
5	1	1	0	1	0	0	0	0	0	0	1
6	1	0	1	1	0	0	0	0	1	0	
7	1	1	1	0	0	0	0	0	1	0	
8	1	1	1	1	0	0	0	0	1	0	
11	1	1	0	0	0	1	1	1	0	0	2

When calling procedure `addNode()` for Table 2.9, which is the subtable of Table 2.1 made of rows 1, 2, 4, 5, 6, 7, 8 and 11, we get:

1. The new node is created; it is not a leaf, since there are more classes. The node is connected to its parent using edge, labeled with category $F = 1$.

2. The attribute value bit with the highest entropy is b_4 and it is the label of the new node.

3. Two subtables are created, one for category $F = 0$, for rows where $b_5 = 0$, i.e. rows 2, 7 and 11, and another for category $F = 1$, for rows where $b_5 = 1$, i.e. rows 1, 4, 5, 6 and 8. Procedure `addNode()` is called for each of these subtables.

Procedure `addNode()` is called until the table cannot be divided any more, i.e. the attribute value bit with the highest entropy belongs to one category.

The tree shown in Figure 2.1 is the final result of the above exercise. It should be able to classify occurrence vectors not belonging to Table 2.1.

Consider then example $\beta = 1111000110$. According to our assumptions listed on page 35 word β should be assigned to class 1; our procedure starts from the root and checks the value of b_1. It is 1, so the procedure goes to b_4. It checks the value of b_4. The value is 1 and the procedure goes to b_9, which equals 1, so the given vector belongs to class of index 1. Let us compare the result with our presumptions included in Section 2.1.2 on page 43. Our example is compatible with step 4 of the procedure `addNode()` (see page 43), because fields b_5, b_6,b_7 and b_{10} of β are equal to 0. It means that β belongs to class 1. So, the example classification result produced with the decision tree is correct.

Let us consider another example with $\beta' = 0011000010$. The value of b_1 is 0, so the procedure goes to b_5. It checks the value of b_5, which is 0. Then the procedure goes to b_2. The value of b_2 is 0. Since edge 0 of node b_2 indicates class 0, it can be deduced that β' belongs to class 0. There, however, the classification result, produced by the decision tree contradicts our expectation expressed in Section 2.1.2, as according to p.4 on page 43, word β' should belong to class 1.

The reason of the above incompatibility underlies in the fact that the decision tree classification works well when differences between classes are related to a single feature. However, the division included in Section 2.1.2 is more complex, and involves multiple features. So we have to look for a yet another classification mechanism.

2.3 Artificial neural network

In this approach, supervised learning is used to train the network, instead of to build a tree as in the previous section. Essentially, training the network and building a tree represents deriving the classification rules, when applying these rules is classification of input data – either as input to the network or the tree. The approach based on neural networks may, however, provide a more attractive alternative, because they take into account all elements of the input during classification, thus may be expected to handle offers with interrelated attribute values better than decision trees.

2.3.1 Neural network architecture

A neural network consists of layers of neurons, where each neuron calculates a function:

$$y : X \to \mathbb{R} \tag{2.9}$$

Its arguments are n-vectors $\langle x_1, x_2 \dots x_n \rangle$ of real numbers called *stimulations*.

We will denote a single neuron by L_i, where L indicates a particular layer of the network, while i indexes the neuron within the layer. A structure of L_i is shown schematically in Figure 2.2.

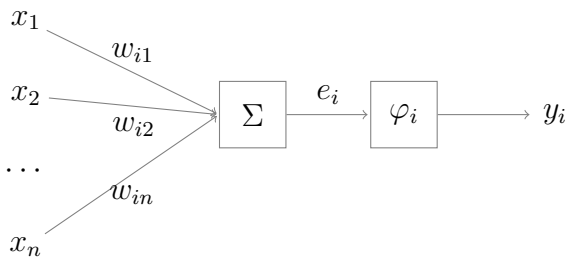


Figure 2.2: A single neuron

Block Σ calculates a weighted sum of stimulations; if we neglect for a while the neuron's order number within the layer, its formula would be the following:

$$e = \sum_{j=1}^{n} w_j x_j + w_0, \tag{2.10}$$

where *weights* w_j, $j = 1, .., n$, and *bias* w_0, are real numbers. Output y is calculated as $y = \varphi(e)$, where φ is called a *transfer function*. Function φ and vector $W = \langle w_1, w_2 \dots w_n \rangle$ of weights define a neuron. Consequently, φ_i and $W_i = \langle w_{i1}, w_{i2} \dots w_{ni} \rangle$ define neuron L_i, as shown in Figure 2.2. Concrete values of weights are set during a training phase. Inputs $\langle x_1, x_2 \dots x_n \rangle$ are connected to respective weights $\langle w_1, w_2 \dots w_n \rangle$. Products of the respective inputs and weights are summarized. An output of the operation is denoted by symbol e. Next, function φ_i takes e_i as an argument and returns output y_i.

The individual neuron can be trained to recognize single objects. However, neurons may be better trained if they are joined in a layer [64]. The number of layers has an impact on the network capability to generalize [24]. When several layers are joined in the way that the output of the preceding layer provides input to the next layer, a *multi-layer* neural network is created. The last layer is the *output layer*. The other layers are *hidden*.

An example of a two layer network is shown in Figure 2.3, in which for $i = 1, ..., 10$, x_i are inputs and H_i neurons of the input (hidden) layer, and for $j = 1, ..., 3$, O_j are neurons of the output layer. Each single neuron H_i has its φ_i transfer function and a 10-element vector of weights W_i, which fully specify the hidden layer.

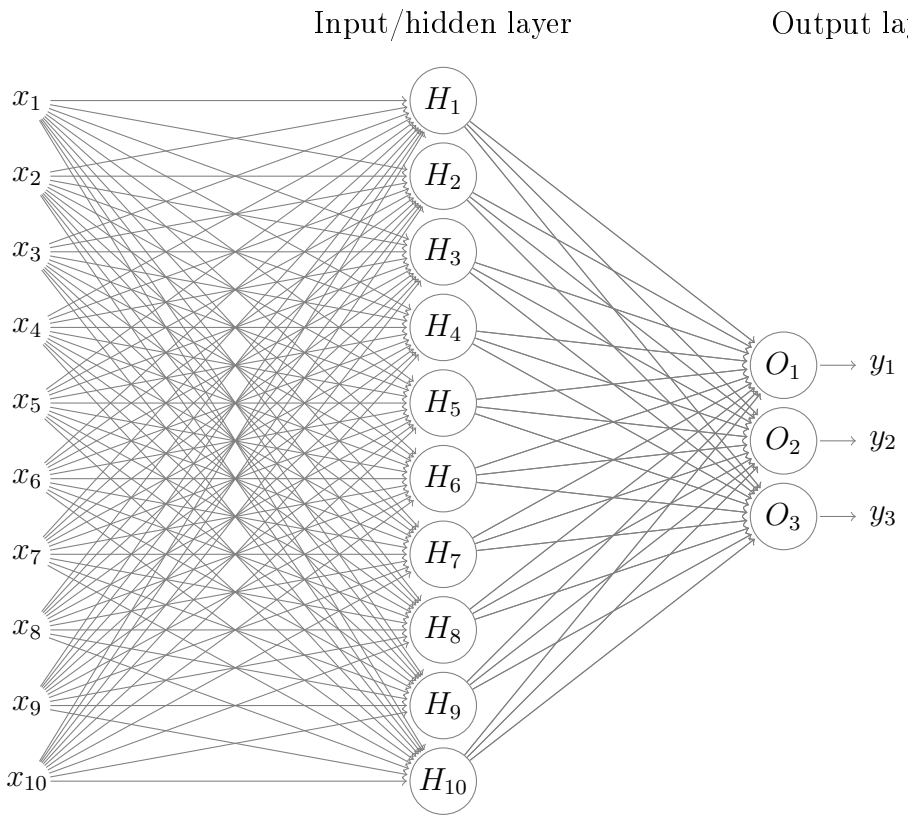


Figure 2.3: An example neural network with one hidden layer

2.3.2 Training of the neural network

Users of any neural network software package do not need to know details of the supported learning processes. They are also beyond the scope of this work. Below we only outline a general principle of putting a neural network to work for the learning task considered in this Thesis, which is setting its weights to classify offers in the bargaining set.

Training of a neural network uses the backpropagation method [24] which relies on the following:

- Output values are computed for the set of training data using initially randomly generated values of weights. In the next iteration the weights are modified according to the values calculated in the next step of the algorithm. This stage is called *forward propagation*.

- The error in the output layer is calculated by comparing output values with expected targets. Next, the values of errors are used to calculate gradients of transfer function of the output layer. Based on the

Table 2.10: Training data for the example neural network

l	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	*class*
1	1	0	0	1	0	0	1	0	0	1	001
2	1	0	0	0	0	1	0	0	0	1	
3	0	0	0	1	0	1	1	0	0	0	
4	1	0	0	1	0	1	1	0	0	1	
5	1	1	0	1	0	0	0	0	0	0	010
6	1	0	1	1	0	0	0	0	1	0	
7	1	1	1	0	0	0	0	0	1	0	
8	1	1	1	1	0	0	0	0	1	0	
9	0	1	1	1	0	0	0	0	1	0	
10	0	0	1	0	1	1	1	1	0	0	100
11	1	1	0	0	0	1	1	1	0	0	
12	0	1	0	0	0	1	1	1	0	0	

gradients of the output layer, gradients in hidden layers are computed. The goal of this stage is to find the values we will use to alter current weights and biases. The values are called *deltas*. Next, the weights are modified due to the calculated gradients. This stage is called *backward propagation*.

In order to demonstrate the training process we will perform the following exercise by using the same data which have been used to build the decision tree in the previous section – with the last column in Table 2.1 slightly modified, i.e. with class labels coded by binary numbers, as shown in Table 2.10.

We built neural network like the one shown in Figure 2.3 consisting of 10 inputs, the hidden layer consisting of 10 neurons and the output layer consisting of three neurons. As a transfer function for both layers, a popular *sigmoid function* $\varphi(x) = \frac{1}{1+e^{-x}}$ was chosen [24].

The initial weights, biases, input and target data as well as neuron outputs and gradients necessary to calculate deltas are presented in Table 2.11. The hidden neurons are denoted by a symbol H with appropriate index. The output neurons are denoted by O.

The first input 10-element vector in Table 2.10 is X. Vector X, as well as 3-element target vector T, are placed in the first column of Table 2.11. The first weights and biases are chosen randomly. The weights of the first neuron of the hidden layer and the associated bias are displayed in the first row of Table 2.11.

Each input is associated with the respective weight. Outputs are created by summing up the products of associated inputs and weights. The transfer function of the first layer is φ_1 and of the transfer function of the second one is φ_2. Thus, the value of function e_1 from Equation 2.10 in L_1 neuron is:

Table 2.11: Initial weights and biases

		Weights												
Neuron	*Input*	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	*Bias*	*Output*	*Gradient*
H_1	1	0.04	0.03	0.04	0.01	0.02	0.04	0.03	0.01	0.03	0.01	3	0.95647	0.0008752
H_2	0	0.05	0.04	0.05	0.01	0.05	0.04	0.04	0.02	0.05	0.03	2	0.89378	0.0021250
H_3	0	0.05	0.02	0.02	0.01	0.04	0.04	0.05	0.05	0.05	0.01	1	0.75398	0.0032892
H_4	1	0.03	0.01	0.02	0.02	0.04	0.04	0.02	0.01	0.03	0.05	0	0.52996	0.0009244
H_5	0	0.04	0.03	0.03	0.02	0.05	0.02	0.04	0.03	0.03	0.03	-1	0.29525	0.0015972
H_6	0	0.05	0.01	0.02	0.04	0.02	0.02	0.02	0.03	0.03	0.04	-2	0.13587	0.0004390
H_7	1	0.05	0.03	0.01	0.03	0.04	0.04	0.03	0.04	0.03	0.01	-3	0.05315	0.0000241
H_8	0	0.02	0.04	0.05	0.05	0.05	0.03	0.05	0.01	0.02	0.02	-4	0.02063	0.0000114
H_9	0	0.05	0.02	0.02	0.02	0.01	0.02	0.05	0.04	0.05	0.02	-5	0.00769	0.0000012
H_{10}	1	0.03	0.04	0.03	0.04	0.01	0.01	0.02	0.05	0.04	0.01	-6	0.00273	0.0000001
	Target													
O_1	0	0.03	0.02	0.02	0.01	0.04	0.04	0.01	0.05	0.02	0.02	-2	0.12852	0.2489703
O_2	0	0.05	0.04	0.05	0.01	0.04	0.03	0.01	0.02	0.05	0.04	-6	0.00285	0.2499994
O_3	1	0.01	0.05	0.03	0.01	0.03	0.05	0.02	0.05	0.02	0.05	-3	0.05216	0.2013004

$$e_1 = \begin{aligned} & 1 \cdot 0.04 + 0 \cdot 0.03 + 0 \cdot 0.04 + 1 \cdot 0.01 + 0 \cdot 0.02 + 0 \cdot 0.04 + 1 \cdot 0.03 + \\ & 0 \cdot 0.01 + 0 \cdot 0.03 + 1 \cdot 0.01 + 3 = 3.09 \end{aligned}$$

The output of L_1 is $\varphi_2(e_1) = 0.9564$. All respective outputs of the hidden layer, calculated in that way may be found in Table 2.11. Let us denote the output vector of the hidden layer by Y.

Outputs of O_j, $j = 1, ..., 3$ are created in a similar way – the input to it is provided by vector Y.

Further, products of the elements of vector Y and weights of neurons O_1, O_2 and O_3, with the same indexes, are summarized and function φ is called. In this way each one of the three outputs of the network is obtained, the complete output vector is $V = \langle 0.12852, 0.00285, 0.05216 \rangle$, and forward propagation (the first stage of the algorithm) is performed.

The second, back propagation stage, involves calculations in the output layer and the results are propagated in the hidden layer. It starts from calculating errors. The error vector is a difference between the target and the output:

$$E = T - V = \langle -0.128528, -0.002854, 0.947836 \rangle$$

Derivative of the sigmoid function is:

$$\varphi(x)' = \varphi(x) \cdot (1 - \varphi(x)). \tag{2.11}$$

Gradients of the output layer are calculated by applying Formula 2.11 to elements of error vector E calculated above. The results are listed in the last three cells of the last column of Table 2.11. The gradient of a hidden layer neuron is equal to the calculus derivative of the activation function of the hidden layer evaluated at the output of the neuron multiplied by the sum of the product of the outputs and their associated weights of the hidden layer. The result is shown in the first ten cells of the last column of Table 2.11.

The last step of the backpropagation algorithm is calculating values which have to be added to the current weights in order to update them, i.e. *deltas*. They are calculated by multiplying the coefficient η by the gradient associated with the weight and the input value associated with the weight.

The coefficient η is responsible for the speed of learning. Its value is set experimentally. The value of another coefficient, called *momentum*, is also set experimentally. The momentum is a parameter having an impact on the stabilization of learning process. In our case both parameters have respectively values 0.9 and 0.4.

Table 2.12: Weights and biases of the trained network

Neuron	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	*Bias*
H_1	1.54	3.04	-3.95	-3.42	-3.82	0.92	0.18	0.02	0.03	0.01	1.52
H_2	-1.44	-0.73	-1.22	-0.89	2.15	0.16	0.1	0.03	0.05	0.03	-0.59
H_3	-0.31	-0.3	-3.32	-2.86	0.71	0.27	0.14	0.06	0.05	0.01	5.56
H_4	0.42	-1.06	-2.49	-1.89	-0.69	1.09	0.13	0.02	0.03	0.05	3.94
H_5	0.11	-0.66	-1.92	-1.72	0.56	0.14	0.09	0.04	0.03	0.03	-0.27
H_6	-0.29	-2.62	-2.55	-1.76	0.72	1.21	0.14	0.04	0.03	0.04	-2.31
H_7	-0.39	-2.19	0.13	0.25	0.67	0.71	0.11	0.05	0.03	0.01	-2.95
H_8	-1.26	-1.9	-0.27	-0.23	2.96	-0.17	0.08	0.02	0.02	0.02	-4
H_9	-0.38	0.39	-1.36	-1.11	0.17	0.14	0.09	0.05	0.05	0.02	-5
H_{10}	0.29	-2.59	1.95	1.91	-0.91	-0.26	0.01	0.04	0.04	0.01	-6
O_1	0.3	4.26	-0.84	-4.16	3.22	1.23	-4.34	-4.46	6.67	-3.89	-1.23
O_2	-3.15	5.16	5.06	-3.78	0.68	-0.02	2.8	-1.02	0.19	0.33	-2.45
O_3	-0.19	0.08	0.11	-0.03	0.04	0.08	0	0.05	0.03	0.04	-4.1

After modifying the values of weights and biases, new outputs are computed. The iteration ends, when the error is below a minimum – in our case set up on value 0.03. The value has been reached after about 300 iterations. The target weights and biases are presented in Table 2.12.

Detailed explanation of how the neural network works and the related numerical content of Table 2.11 may seen a little bit redundant and obvious in the context of this Chapter, but we would like to refer to it later on, when considering in Chapter 5 the minimum computational capability of document agents they have to exhibit in order to interact with execution devices.

Recall that the MIND system outlined in Figure 1.1 is a loosely coupled distributed system where agents in general have no any specific computational support except their own embedded functionality, to perform all the required calculations described above. The same refers to their memory resources, as they would have to carry all the numbers listed in Table 2.11 during migration.

2.3.3 Utilization of software packages

In the experiments we used Matlab 2012b, which facilitated the creation of the neural network [8].

Now we may examine it by classifying the same occurrence vector $\beta = 1111000110$ we have used to test the decision tree. We will use the training data from Table 2.10.

The required response of the network should be a zero-one vector. Recall

Formula 2.9, which defines an artificial neuron. Its output is a real number. Since the output layer of our network consists of three neurons, we receive three real numbers. So, a function is needed, to convert the vector of the real numbers into the required bit vector.

That task is performed by function κ, which encodes the output (a vector of real numbers) in such a way that it creates a bit word with all bits set to 0 except the bit of the position corresponding to the maximum number of the input vector set to 1.

$$\kappa : D \to B \tag{2.12}$$

D is an array of real numbers and B is a bit word.

Let us go back to the example we have started considering in Section 2.3.2. The response of the neural network that we have obtained in that exercise is $Y = \langle 0.0235, 0.9971, 0.0065 \rangle$. After interpretation made by function κ we get $\kappa(Y) = \langle 0, 1, 0 \rangle$. So, the classified object belongs to class $c = 010$.

The above example demonstrates that an artificial neural network may potentially provide a useful mechanism for making MIND documents capable of learning to classify offers in the bargaining set. In particular its ability to handle the entire offer during analysis gives us a reason to believe that they might learn faster and classify more precisely than decision trees. Moreover, knowledge 'learned' by the agent may be represented with engaging realistic volume of its memory resources.

2.3.4 Recognizing sequences

In Chapter 1 we have used the notion of sequences of offers in connection to trees representing games in an extensive form. Below we will continue on that in order to find out if it is possible to classify execution devices based on sequences of offers they return during negotiations. As mentioned in Section 1.5 we consider repeated encounters of document-agents and execution devices, therefore a sufficient training set of negotiation histories may reasonably be expected.

Origin of the sequence

A game tree, like the one shown in Figure 1.3, specifies all possible sequences of offers that player P_i may ever respond to its opponent P_{-i}. Let us assume that P_1 starting a game is an execution device and its opponent P_2 is a document-agent. Each sequence of device responses corresponds to a specific game tree path, from the first response to the initial player's P_1 offer down to the last one, ending at the tree leaf if it happened that one of players P_i concluded the game by repeating one of the previous offers made by P_{-i}, or

repeating one of its own offers. Each recorded sequence of offers returned by player P_i during its interaction with P_{-i} will be called a *negotiation history* h. Negotiation history h is a sequence of two interleaving sequences ς_i and ς_{-i} belonging respectively to P_1 and to P_2. Our goal is to predict the future opponent's moves basing on the negotiation history, so we are in fact interested only in the order of moves made by the opponent. Thus in our further development we will be considering only sequences made by one player, the execution device. Moreover, whenever it may cause no confusion we will consider a negotiation history just as a sequence of offers by P_i, or responses by P_{-i}.

Definition 11. *Sequence ς_i^k determines the order of offers, submitted by player P_i, ended with an agreed contract offer.*

Sequence ς_i^k belongs to negotiation history h_k, where $k \in \mathbb{Z}_{>0}$ indicates the history number, an upper index k associates it with the respective negotiation history.

In Section 1.3.3 we have shown how to make use of the knowledge about sequences obtained during the negotiation process to calculate a Nash equilibrium. Now we show how to make use of knowledge about sequences in our classification problem.

In the previous section we have classified offers based on attribute values occurring in the offer. It is important to distinguish classes of offers in the bargaining set from classes of execution devices and document-agents, which exchange these offers. It may happen that two offers consisting of the same attribute values may be selected from the bargaining set by execution devices belonging to different classes. They can be distinguished, but it requires a document-agent to analyze not only the structure of each offer returned by the execution device (its opponent), but also the *order* in which they are submitted.

Example 3. *Consider information about negotiation histories and associated classes stored in Table 2.13. When the bargaining process starts, player P_1 submits its offers and observes player's P_2 counteroffers (player's P_1 offers are omitted). Player's P_1 reasoning may be the following:*

1. *When P_2 submits o_3, P_1 excludes class 0 from further considerations, since there is offer o_4 on the beginning of sequence ς^0.*

2. *Player P_2 submits o_5. Player P_1 concludes that submitted offers belong to class 1, since only in the case of sequence ς^1 offer o_5 follows offer o_3.*

Table 2.13: Sequences of player's P_1 offers (device)

sequence name	sequence	device class
ς^0	$o_4o_1o_2o_3o_6o_5$	0
ς^1	$o_3o_5o_1o_2o_6o_4$	1
ς^2	$o_3o_1o_2o_4o_5o_6$	2

Certainly one cannot expect that sequences will always be identical as in the simplistic example above. More likely, it would be realistic to attempt machine learning techniques to use negotiation histories as the training material – to train documents to recognize classes of execution devices and possible contracts associated with them.

The structure of the sequence

Let us denote the j-th offer in sequence ς returned by P_2 by o^j. The order of elements in a sequence depends on utilities of offers in the sequence. If ς^k is subsequence of h_k and h_k represents behavior of player P_2, then $U_i(o^j) > U_i(o^{j+1})$ for $j = 1, ..., N$, where N is the length of sequence ς^k.

Sequence encoding

Given sequence $\varsigma = o^1 \dots o^N$, we get for player P_i:

$$U_i(o^1) > U_i(o^2) > \dots > U_i(o^N) \tag{2.13}$$

We have demonstrated in the previous section (see page 52) how to code offers to put a neural network to work. Now we would like to do the same with sequences to classify them. Let us represent sequences by precedence relations. Then, taking into consideration Equation 2.13, sequence $\varsigma = o^1o^2 \dots o^{N-1}o^N$ would correspond to the following set of inequalities:

$$\begin{array}{rcl} U_i(o^1) & > & U_i(o^2) \\ U_i(o^2) & < & U_i(o^1) \\ & \vdots & \\ U_i(o^{N-1}) & > & U_i(o^N) \\ U_i(o^N) & < & U_i(o^{N-1}). \end{array} \tag{2.14}$$

With that we can convert each k-element sequence into $k \cdot (k-1)$ relations including '>' and '<' operators. Next, the relation operator '>' may be

encoded as 1, and respectively '<' as 0. For example, sequence $\varsigma^1 = o^1 o^2 o^3 o^4$ may be transformed into the following twelve relations:

$$\begin{aligned}
o^1 o^2 &= 1 \\
o^1 o^3 &= 1 \\
o^1 o^4 &= 1 \\
o^2 o^1 &= 0 \\
o^2 o^3 &= 1 \\
o^2 o^4 &= 1 \\
o^3 o^1 &= 0 \\
o^3 o^2 &= 0 \\
o^3 o^4 &= 1 \\
o^4 o^1 &= 0 \\
o^4 o^2 &= 0 \\
o^4 o^3 &= 0.
\end{aligned}$$

Sequence as a system of equations

Equation 2.14 may be expanded using Formula 1.5 to obtain the expression describing the classification task:

$$\begin{aligned}
u_i(v_1^1) + \cdots + u_i(v_m^1) &> u_i(v_1^2) + \cdots + u_i(v_m^2) \\
&\vdots \\
u_i(v_1^{N-1}) + \cdots + u_i(v_m^{N-1}) &> u_i(v_1^N) + \cdots + u_i(v_m^N)
\end{aligned} \tag{2.15}$$

where m is the offer length and u_i is the utility of a given attribute value. Now recall that each v_m is a value chosen from $A_m = \{a_{im} | i = 1, \ldots, |A_m|\}$. We denote value of attribute index by x_i, $i = 1, \ldots, m$. It allows to transform Equation 2.15 above into the following form:

$$\begin{aligned}
u_i(a_{x_1 1}^1) + \cdots + u_i(a_{x_m m}^1) &> u_i(a_{x_1 1}^2) + \cdots + u_i(a_{x_m m}^2) \\
&\vdots \\
u_i(a_{x_1 1}^{N-1}) + \cdots + u_i(a_{x_m m}^{N-1}) &> u_i(a_{x_1 1}^N) + \cdots + u_i(a_{x_m m}^N)
\end{aligned} \tag{2.16}$$

Since indexes x unambiguously represent attributes, the transformation of Equation 2.16 into the form presented below is straightforward.

$$\begin{aligned}
x_1^1 + \cdots + x_m^1 &> x_1^2 + \cdots + x_m^2 \\
&\vdots \\
x_1^{N-1} + \cdots + x_m^{N-1} &> x_1^N + \cdots + x_m^N
\end{aligned} \tag{2.17}$$

Formula 2.17 provides a desired representation of our problem, with negotiation histories (training data) represented by a set of inequalities, where operators '<' and '>' have been encoded respectively by 0 and 1.

Network training

Table 2.14 provides an example history including four sequences consisting of four offers with five elements each.

Table 2.14: Coding of sequences of player's P_1 (device) offers

sequences of offers				
$\varsigma^0 = o_5 o_2 o_6 o_7$	$\langle 7,2,7,15,13\rangle$	$\langle 6,2,6,0,15\rangle$	$\langle 4,2,7,15,12\rangle$	$\langle 3,9,6,1,12\rangle$
$\varsigma^1 = o_1 o_2 o_3 o_4$	$\langle 1,2,6,3,14\rangle$	$\langle 6,2,6,2,15\rangle$	$\langle 8,2,6,3,12\rangle$	$\langle 2,9,6,1,13\rangle$
$\varsigma^2 = o_8 o_6 o_9 o_{10}$	$\langle 0,2,5,10,14\rangle$	$\langle 4,2,5,10,15\rangle$	$\langle 8,9,5,9,13\rangle$	$\langle 5,9,5,8,15\rangle$
$\varsigma^3 = o_5 o_{11} o_{12} o_4$	$\langle 7,2,5,11,13\rangle$	$\langle 2,2,4,5,13\rangle$	$\langle 6,9,5,4,14\rangle$	$\langle 3,9,4,4,12\rangle$

The offers have been coded as described in the previous paragraph in a form suitable for neural networks. Originally, the offers are stored in the symbolic form. However, neural networks require the numeric inputs. So, the symbols have to be encoded into numbers. We will explain in detail how to code the symbols in Chapter 3.

The leftmost column indicates the offers from which the sequences consist of. Consider for example sequence $\varsigma^1 = o^1 o^2 o^3 o^4$ and offer o^2.

After encoding, we obtain offers and their sequences in the numeric form, what can be seen in Table 2.14, which is encoded as $\langle 6,2,6,2,15\rangle$.

If the sequences of offers are to be trained, they have to be converted into pairs of offers and the relations between them, what has been shown in Section 2.3.4 (Algorithm 4.2 is presented in Chapter 4). It has been shown there that a sequence consisting of four offers can be converted into 12 relations.

Table 2.14 contains four sequences, each of them consisting of four offers. So the sequences can be converted into 48 pairs which are shown in Table 2.15. One half of these relations are ' $>'$ relation (encoded as 1), while the second one – ' $<'$ (encoded as 0).

Look at the first leftmost cell in Table 2.15. It includes the pair of two offers, $o_2 = \langle 3,9,4,4,12\rangle$ and $o_1 = \langle 2,2,4,5,13\rangle$. Since the relation has been coded as 0 then $o_2 < o_1$. The pairs of offers and associated with them relations, as we can see in Table 2.15 are called the *relation vectors*.

The training of relations relies on introducing to the network an encoded pair of offers, such as a row in Table 2.15 and the corresponding target class, such as the relation in the table.

Table 2.15: Sequences converted to pairs and relations (relation vectors)

sequences converted to pairs			
pairs in relation 0		*pairs in relation* 1	
$3,9,4,4,12$	$2,2,4,5,13$	$2,2,4,5,13$	$3,9,4,4,12$
$6,9,5,4,14$	$2,2,4,5,13$	$2,2,4,5,13$	$6,9,5,4,14$
$3,9,4,4,12$	$6,9,5,4,14$	$6,9,5,4,14$	$3,9,4,4,12$
$2,2,4,5,13$	$7,2,5,11,13$	$7,2,5,11,13$	$2,2,4,5,13$
$3,9,4,4,12$	$7,2,5,11,13$	$7,2,5,11,13$	$3,9,4,4,12$
$6,9,5,4,14$	$7,2,5,11,13$	$7,2,5,11,13$	$6,9,5,4,14$
$8,9,5,9,13$	$4,2,5,10,15$	$4,2,5,10,15$	$8,9,5,9,13$
$5,9,5,8,15$	$4,2,5,10,15$	$4,2,5,10,15$	$5,9,5,8,15$
$4,2,5,10,15$	$0,2,5,10,14$	$0,2,5,10,14$	$4,2,5,10,15$
$8,9,5,9,13$	$0,2,5,10,14$	$0,2,5,10,14$	$8,9,5,9,13$
$5,9,5,8,15$	$0,2,5,10,14$	$0,2,5,10,14$	$5,9,5,8,15$
$5,9,5,8,15$	$8,9,5,9,13$	$8,9,5,9,13$	$5,9,5,8,15$
$8,2,6,3,12$	$1,2,6,3,14$	$1,2,6,3,14$	$8,2,6,3,12$
$6,2,6,2,15$	$1,2,6,3,14$	$1,2,6,3,14$	$6,2,6,2,15$
$2,9,6,1,13$	$1,2,6,3,14$	$1,2,6,3,14$	$2,9,6,1,13$
$2,9,6,1,13$	$8,2,6,3,12$	$8,2,6,3,12$	$2,9,6,1,13$
$8,2,6,3,12$	$6,2,6,2,15$	$6,2,6,2,15$	$8,2,6,3,12$
$2,9,6,1,13$	$6,2,6,2,15$	$6,2,6,2,15$	$2,9,6,1,13$
$3,9,6,1,12$	$6,2,6,0,15$	$6,2,6,0,15$	$3,9,6,1,12$
$4,2,7,15,12$	$6,2,6,0,15$	$6,2,6,0,15$	$4,2,7,15,12$
$3,9,6,1,12$	$7,2,7,15,13$	$7,2,7,15,13$	$3,9,6,1,12$
$6,2,6,0,15$	$7,2,7,15,13$	$7,2,7,15,13$	$6,2,6,0,15$
$4,2,7,15,12$	$7,2,7,15,13$	$7,2,7,15,13$	$4,2,7,15,12$
$3,9,6,1,12$	$4,2,7,15,12$	$4,2,7,15,12$	$3,9,6,1,12$

2.3.5 Utilization of the learned knowledge

Utilization of the acquired knowledge for classifying sequences is a little bit more complicated than in the case of classifying offers demonstrated before. We need an algorithm that can transform relations recognized by the neural network back to sequences. It consists mostly of conditional expressions. For brevity we illustrate its basic idea with a graph shown in Figure 2.4. Complete version of the algorithm is presented in Chapter 4 (Algorithm 4.3).

There is a list of relations on the right-most side of the figure, which is the source of data. The target data are the sequences on the left side of the figure:

1. The first relation in the queue is $o_1 > o_3$. Because the target sequence is empty, we have to initialize it. Since elements with the higher utility should be at the beginning of the sequence, we put offer o_1 at the beginning of the new sequence and offer o_3 at the end.

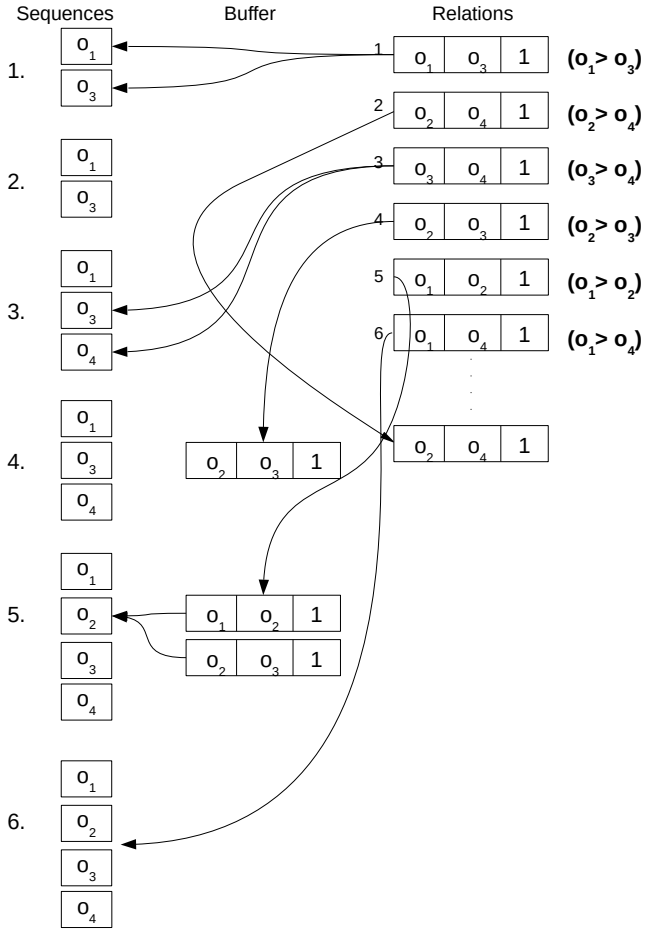


Figure 2.4: Inserting data stored in relations into a sequence

2. The second relation is $o_2 > o_4$. Because neither o_2 nor o_4 exist in the recovered sequence, we cannot use it yet so we put the relation back to the end of the queue – to use it later.

3. The next relation is $o_3 > o_4$. Because o_3 is the last element in the sequence, it means that it has the lowest utility. Relation $o_3 > o_4$ means, that element o_4 is of even the lesser utility than element o_3, so we put it in the sequence as the last element.

4. Relation $o_2 > o_3$, unfortunately, does not provide enough information to put it into the output sequence. We know that o_2 should precede o_3 in the sequence, but we do not know, if on the fist or the second

position. So, we put it in the buffer waiting for more information from other relations.

5. Relation $o_1 > o_2$ is not useful unless it is combined with the relation stored in the buffer. In the previous step it was not known if o_2 should be on the first or the second position. Now there is information that $o_1 > o_2$, so we put o_2 on the second position.

6. The last relation is $o_2 > o_4$. In step 2 above we were not able to make use of this relation because offers o_2 and o_4 were not present in the output sequence. Now both of them are in the sequence, o_2 on the second, and respectively, o_4 on the last position – as it has been inferred from other relations earlier.

The example in Figure 2.4 does not involve situations when analyzed input relations are contradictory. Chances for that increase when the network is insufficiently trained – for example, instead of $o_1 > o_2$ it would return $o_1 < o_2$. We counteract that by not taking contradictory (non-conclusive) relations into account.

To this point we have presented a method of grouping offers and their sequences using artificial intelligence approaches. We have investigated three common techniques, unsupervised learning based on clustering with a plain k-means algorithm, and two supervised learning techniques – decision trees and artificial neural networks. These techniques have been illustrated with practical examples, demonstrating how to utilize them in classifying offers and sequences of offers. Neural networks have been indicated as the candidate for further development in this Thesis. Before utilizing them, however, we have to analyze in full detail the structure of offers exchanged between document-agents and execution devices.

Chapter 3

Bargaining Set

In Chapter 1 we have defined the problem of resolving a conflict between two parties that search in a certain set of elements the element that can best satisfy each party with respect to some utility. We have assumed that either party provides its own valuation of the utility of any element of that set, and that no public information on the preferred elements is available a priori. In consequence, narrowing the searched set of elements to some non-empty intersection of subsets of the elements preferred most by both parties is not possible.

We have proposed a method for finding a solution to this problem with a simple bargaining game that uses the notions of utility functions and discount factors. Elements of the set in which the solution is searched are modeled as tuples of items, where each single item is a value of a single attribute of the solution being sought. Since a mobile interactive (MIND) document arrives to its execution device (as shown in Figure 1.1) with the intention to complete a relevant activity specified in its workflow and using whatever resources of the device are available, the negotiated contract specifies what device resources will really be engaged and how the current document activity will be performed. In other words, each tuple describes a specific set of *options* available in the *execution context* provided by the device when performing the activity.

Throughout the rest of this chapter we develop the concept of *option trees* to handle hierarchical dependencies between attributes of negotiated contracts. We start by distinguishing five attributes, which values indicate all possible execution contexts:

1. Who shall be the *performer* of the activity?
2. What is the required *availability* of various resources to complete the activity?

3. What shall be the *performance* of the execution device hardware?

4. What *security* mechanisms will have to be involved when performing the activity?

5. What functionality may contribute to interaction *reliability* when performing the activity?

3.1 Option trees

The order of attributes listed before indicates how they should be taken into account by parties when negotiating the contract. A question that may be asked here is how to represent the hierarchy of attributes directly in the bargaining set? In order to provide such a representation we have developed the model inspired by the *Collaboration Protocol Profile and Agreement (CPPA)* specifications used by the *Electronic Business using eXtensible Markup Language*, commonly known as ebXML – an XML based standard sponsored by OASIS and UN/CEFACT [47]. CPPA specify respective XML Schemas of *Collaboration Protocol Profile (CPP)*, and *Collaboration Protocol Agreement (CPA)* documents. CPP is an XML tree with elements describing options of a single party willing to be a side of a business transaction, while CPA is an XML tree containing CPP elements agreed (or negotiated) by parties participating in the transaction. CPPs of two collaborating parties are in conflict if the corresponding nodes of their XML trees differ in content. A simplistic solution (used for a long time by the ebXML community) has been creating by one party a complete CPA document with empty places for another party to fill them out with missing data. Extension of the standard proposed by OASIS to introduce negotiations as the means for reaching a more precise agreement on elements of CPPs that should be incorporated in CPA [16], called *Automated Negotiation of Collaboration Protocol Agreements*, has never been published [44]. The model of option trees presented in this chapter and the SBG scheme defined in Chapter 1 provides such an extension [35].

Recall from Chapter 1 that bargaining set C consists of all offers that parties may select during negotiations. Each party distinguishes offers in C independently, with regard to the utility of each single offer to the party. Based on that, the party sorts all offers in C from the most to the least preferred ones. By denoting a single offer of value options for the five attributes defined before as a 5×1 column vector, the ordering of offers preferred in C by the party may be represented as a $5 \times |C|$ matrix of attribute options.

In Figure 3.1 five offers $C = \{o_i | i = 1, .., 5\}$ have been sorted in the order $\{o_3, o_2, o_1, o_5, o_4\}$.

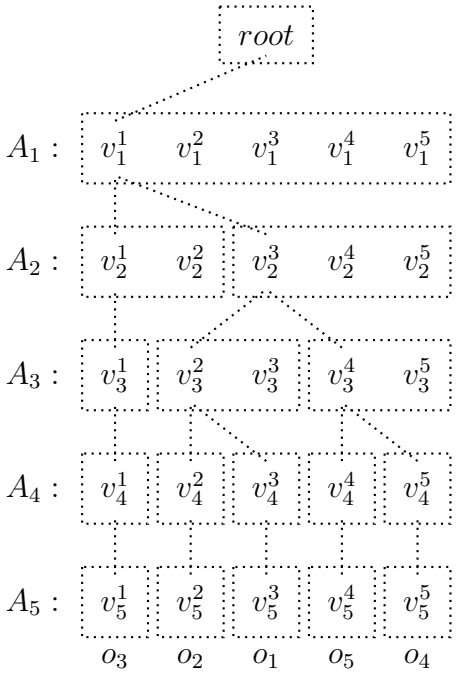


Figure 3.1: Building the option tree

Each row of the $5 \times |C|$ matrix specifies the related attribute value options considered by the party when determining its preference of offers. If for some adjacent columns in the matrix it happens that values in a given section of some row intersecting with these columns are equal, it will mean that when evaluating utility of the offers represented by these columns this particular attribute value dominates values of the next attribute of these offers, located in the respective section of the row immediately below. This hierarchical dependency may be represented by a tree, built of the matrix elements, where each group of adjacent elements in a given row that are equal, represent a single tree node, and each node which value dominates a value in the row below is connected with the node of the latter value. For example, the tree shown in Figure 3.1 has been built under the following assumptions: $v_1^1 = v_1^2 = v_1^3 = v_1^4 = v_1^5$, $v_2^1 = v_2^2$, $v_2^3 = v_2^4 = v_2^5$, $v_3^2 = v_3^3$, $v_3^4 = v_3^5$. A tree built of the matrix specifying argument value options of all offers in the bargaining set, as shown above, will be called the *option tree*.

Before analyzing how the option trees may be sorted in a system of MIND documents let us first consider what attribute value options they may consist of. This idea has been published by the Author before in [32].

3.1.1 Performer options

Depending on the document capability to perform certain actions on its own, actual performers of a given activity may vary. Recall that MIND documents [18] are capable of performing services embedded in their code. With that mechanism in place MIND documents can call external services, if only the execution device can provide access to some network, as well as local services, if only any are available on the execution device. If for some reason embedded services are missing or inactive (e.g. switched off by a local antivirus system), a knowledge worker of the organization must take care of completing the activity and work with a document content using tools available locally on the device. When doing that, he/she may still use whatever local and external services are available. It may also happen that a document with embedded services – in place and active – may still require a knowledge worker to perform certain actions, beyond its standard capacity, in order to properly complete the activity.

Table 3.1: Values of the attribute 'performer'

Knowledge worker	*Embedded service*	*External service*	*Local tool*	*Local service*	*Attribute value*	*Execution context*
0	**1**	*****	**0**	*****		*Document (D)*
0	1	0	0	0	**D1**	
0	1	0	0	1	**D2**	
0	1	1	0	∗	**D3**	
1	**0**	*****	**1**	*****		*Worker (W)*
1	0	0	1	∗	**W1**	
1	0	1	1	∗	**W2**	
1	**1**	*****	*****	*****		*Document and worker (J)*
1	1	0	0	0	**J1**	
1	1	0	0	1	**J2**	
1	1	0	1	∗	**J3**	
1	1	1	∗	∗	**J4**	

Based on the above, three execution contexts are possible with respect to the *performer* attribute, as shown in Table 3.1: D, without any involvement of the knowledge worker, W, without any involvement of the document embedded services, and J, when the document and worker jointly perform the activity. Its values are marked with labels in boldface at the right side of

the table, while binary flags at the left side represent availability of the respective options used to calculate these values: *knowledge worker*, *embedded service*, *external service*, *local tool* and *local service*. By '$*$' we denote flag values which are irrelevant to the particular context, that may be either set to 0 or 1.

3.1.2 Availability options

A detailed list of optional resources, required to complete a given activity by a MIND document, strongly depends on document semantics and may be difficult to encode as set of values of a manageable size. Nevertheless, a realistically small set of options may be specified given the provenance of MIND documents, which are in fact autonomous software agents with a significant degree of independence of their local execution environments. As such, MIND documents may expect just a few key options available at the execution device that may affect their mission. One is whether a network connection is currently available at the execution device (or allowed by its owner), and if so, whether the device's IP is currently *local* or *external* to the organization of the knowledge worker who uses the device. Most likely, if a network connection is needed by the document it may need access to a locally available browser; it may be relevant whether a *specific browser*, with some unique functionality, or just *any browser*, with a generic functionality, is available. A similar type of option has to be considered if the document may request yet another *specific tool* or accept an optional *substitute tool*. Finally completion of the activity may require the execution device to provide the *full keyboard*, or just *selection buttons*, either physical or emulated.

By combining these options we get three execution contexts with respect to the availability attribute, as shown in Table 3.2: S, when the actual execution device is separated from the worker's organization (either physically or on purpose by the worker), E, when the execution device is connected from outside, and I, when connected from inside of the worker's organization. Note that we do not distinguish options related to the availability of browsers when the device is not separated from the organization; the reason has been limiting the number of different attribute values used to compose offers and consequently to limit the size of the bargaining set when implementing negotiation exercises further in the thesis. With that, however, we are not loosing generality of the methodology for determining attribute values presented in this chapter.

Table 3.2: Values of the attribute 'availability'

Local IP/VPN	*External IP*	*Specific browser*	*Any browser*	*Specific tool*	*Substitute tool*	*Full keyboard*	*Selection buttons*	*Attribute value*	*Execution context*
0	**0**	**0**	**0**	*	*	*	*		*Device separated from the organization (S)*
0	0	0	0	0	0	0	0	**S1**	
0	0	0	0	0	0	0	1	**S2**	
0	0	0	0	0	0	1	*	**S3**	
0	0	0	0	0	1	0	0	**S4**	
0	0	0	0	0	1	0	1	**S5**	
0	0	0	0	0	1	1	*	**S6**	
0	0	0	0	1	0	*	*	**S7**	
0	0	0	0	1	1	*	*	**S8**	
0	**1**	*	*	*	*	*	*		*Device connected from outside of the organization (E)*
0	1	*	*	0	0	0	0	**E1**	
0	1	*	*	0	0	0	1	**E2**	
0	1	*	*	0	0	1	*	**E3**	
0	1	*	*	0	1	0	0	**E4**	
0	1	*	*	0	1	0	1	**E5**	
0	1	*	*	0	1	1	*	**E6**	
0	1	*	*	1	*	*	*	**E7**	
1	**0**	*	*	*	*	*	*		*Device connected from inside of the organization (I)*
1	0	*	*	0	0	0	0	**I1**	
1	0	*	*	0	0	0	1	**I2**	
1	0	*	*	0	0	1	*	**I3**	
1	0	*	*	0	1	0	0	**I4**	
1	0	*	*	0	1	0	1	**I5**	
1	0	*	*	0	1	1	*	**I6**	
1	0	*	*	1	*	0	*	**I7**	

3.1.3 Performance options

Hardware of the execution device may affect the current activity in several ways. First of all, performance characteristics of the underlying network would determine document capability in effective exploitation of external services, if only allowed by the worker operating the execution device. In order to capture the widest possible range of networking hardware types, used by execution devices today, we distinguish several options. If the device remains stationary during the activity it may be connected physically to the network with a copper or fiber *wire*, as well as with a TV *cable* or telephone *line*. These connections may be diversified further with regard to specific

technologies they use and throughput they provide; without loosing generality and for the sake of limiting the volume of this attribute values in the experiments described further in the thesis we just assume the former to be of better performance than the latter. On the other hand *wire* and *cable/line* options do not differ much in terms of data transfer fees to be paid by the device user – the former usually is free of charge if used inside of the worker's company, while the latter is charged monthly if used at home. The *telephone modem* option offers typically a lower throughput and involves charging users for the transferred data or for making a connection. Certainly, workers owning execution devices may be less willing to allow arriving documents to make such connections at their cost. On the other hand a cellular network (if a mobile phone is used) offers a significant range of mobility. Another option to be considered is a *wireless* connection; it may be paid (e.g. in a hotel) or free of charge (e.g. at the airport or inside the organization), and offers a reasonable level of mobility to the worker.

Besides networking, other hardware options of the execution device that have to be involved in composing offers shall refer to the *processor* speed and the volume of available *RAM*. Again, for the sake of brevity we consider just two cases for each such option: whether the document can get more memory and processor power than it may initially expect. Such options are not negligible in the case of battery operated devices, as using more processing power may consume more energy.

By combining the options discussed above we get five execution contexts with respect to the performance attribute, as shown in Table 3.3: U, when the underlying network is unknown (i.e. not available or not recommended), and R, M, A and N, when respectively some WiFi network, a device's telephone or ADSL modem, and classic (say twisted pair) Erhernet connection, can be used.

3.1.4 Security options

Performing of any activity specified in the document's workflow must provide a minimum security to both, the execution device and the document. Security options of the underlying network shall concern encryption mechanisms that may be used to protect against various types of attacks on the execution device – either when the document is about to be transferred to it, or when already activated and attempting connections to some external services. With regard to that we consider two available options: using a protocol that enables secure communication of the execution device over a computer network, like *HTTPS*, and if a wireless connection is considered, a protocol enabling verification who should be allowed to access to the net-

Table 3.3: Values of the attribute 'performance'

Wire	*Cable/Line*	*Telephone modem*	*Wireless*	*Processor (fast)*	*RAM (in excess)*	*Attribute value*	*Execution context*
0	**0**	**0**	**0**	*	*		*Network unknown (U)*
0	0	0	0	0	0	**U1**	
0	0	0	0	0	1	**U2**	
0	0	0	0	1	0	**U3**	
0	0	0	0	1	1	**U4**	
0	**0**	**0**	**1**	*	*		*WiFi (R)*
0	0	0	1	0	0	**R1**	
0	0	0	1	0	1	**R2**	
0	0	0	1	1	0	**R3**	
0	0	0	1	1	1	**R4**	
0	**0**	**1**	**0**	*	*		*Telephone modem (M)*
0	0	1	0	0	0	**M1**	
0	0	1	0	0	1	**M2**	
0	0	1	0	1	0	**M3**	
0	0	1	0	1	1	**M4**	
0	**1**	**0**	**0**	*	*		*ADSL modem (A)*
0	1	0	0	0	0	**A1**	
0	1	0	0	0	1	**A2**	
0	1	0	0	1	0	**A3**	
0	1	0	0	1	1	**A4**	
1	**0**	**0**	**0**	*	*		*Twisted pair (N)*
1	0	0	0	0	0	**N1**	
1	0	0	0	0	1	**N2**	
1	0	0	0	1	0	**N3**	
1	0	0	0	1	1	**N4**	

work, like *WPA2-PSK*. Besides these specific network security options more of them may be included to reflect how well the execution device and its system are protected against the document itself, which may be forged by some impostor or infected by a computer virus. Let the former option indicate whether the document content has a *digital signature*, and the latter one, whether the execution device has some *antivirus* system installed.

By combining the options discussed above we get four execution contexts with respect to the security attribute, as shown in Table 3.4: P, when the connection is insecure, K, when the wireless network is protected by the access key, S, when the secure transfer protocol is used, and C, when both the access key and secure transfer protocols are used.

Table 3.4: Values of the attribute 'security'

HTTPS	*WPA2-PSK*	*Digital signature*	*Antivirus*	*Attribute value*	*Execution context*
0	**0**	*	*		*Insecure connection (P)*
0	0	0	0	**P1**	
0	0	0	1	**P2**	
0	0	1	0	**P3**	
0	0	1	1	**P4**	
0	**1**	*	*		*Access key (K)*
0	1	0	0	**K1**	
0	1	0	1	**K2**	
0	1	1	0	**K3**	
0	1	1	1	**K4**	
1	**0**	*	*		*Secure transfer protocol (T)*
1	0	0	0	**T1**	
1	0	0	1	**T2**	
1	0	1	0	**T3**	
1	0	1	1	**T4**	
1	**1**	*	*		*Secure connection (C)*
1	1	0	0	**C1**	
1	1	0	1	**C2**	
1	1	1	0	**C3**	
1	1	1	1	**C4**	

3.1.5 Interaction reliability options

Reliable interaction of a document with its knowledge worker during the activity requires some elementary support by the execution device system to control reliability of interaction with the document content. It may include enabling a specific widgets on the device screen that the document would like to control with its embedded services, as well as providing by the execution device some tools or services capable of doing that. We distinguish a minimal set of four options related to reliable interaction between the worker and document content: an *acceptance button*, which may help the user to control termination of the activity, the *autosave* feature protecting the new document content from being lost during edition, *automatic check* to prevent incorrect input from the user, and the *undo button* facility allowing users to withdraw their most recent input.

By combining these options we get four execution contexts with respect to the interaction reliability, as shown in Table 3.5: L, when the execution

Table 3.5: Values of the attribute 'reliability'

Acceptance button	*Autosave*	*Automatic check*	*Undo button*	*Attribute value*	*Execution context*
0	**0**	*	*		*Low reliability (L)*
0	0	0	0	**L1**	
0	0	0	1	**L2**	
0	0	1	0	**L3**	
0	0	1	1	**L4**	
0	**1**	*	*		*Back-up content (B)*
0	1	0	0	**B1**	
0	1	0	1	**B2**	
0	1	1	0	**B3**	
0	1	1	1	**B4**	
1	**0**	*	*		*Failsafe (F)*
1	0	0	0	**F1**	
1	0	0	1	**F2**	
1	0	1	0	**F3**	
1	0	1	1	**F4**	
1	**1**	*	*		*High reliability (H)*
1	1	0	0	**H1**	
1	1	0	1	**H2**	
1	1	1	0	**H3**	
1	1	1	1	**H4**	

context does not provide any support to protect the document content from being lost by the user mistake, B, when some elementary back-up mechanisms for the document content are in place, F, when the document content is fail-safe, and H, when the execution device provides a sufficient level of document content reliability.

3.2 Policies

To this end we have presented attribute options used to build option trees. It may be readily seen in Tables 3.1-3.5 that due to the specificity of execution contexts implied by combinations of various options there may be many *types* of devices and documents – their respective option trees will include values from different subsets of attribute value sets, and in consequence bargaining sets for each specific pair of player types (devices and documents) may consist

of different offers.

Another issue is representing individual preferences of each party of each specific type, which we model in the option tree by sorting its paths in a certain order. Since each path in such a tree represents a single element of the bargaining set, the 'ordered' option tree represents preferences by the player of each possible offer it can respond to and reflects in a complete way utility of each attribute value in each offer. Rules providing ordering of the option tree constitute a party's *policy*.

Definition 12. Policy *is a set of rules of the form:*

$$Q_{A_k} \rightarrow Q_{A_{k+1}},$$

where Q_{A_k}, $Q_{A_{k+1}}$ denote ordered sets of attribute A_k and A_{k+1} value labels, k denotes an option tree level and '$\rightarrow$' denotes a parent-child relation between any element of Q_{A_k} and $Q_{A_{k+1}}$.

With a set of rules of the form specified by Definition 12 we can build option trees for each specific device and document type. When doing that we will use the following notation. With X denoting any execution context specified in Tables 3.1-3.5, Xi will denote a single attribute value label, $X^* = \{X1, X2, ..., Xn\}$ a set of all labels within that context, and $Xi..j$ their respective subset from Xi to Xj. We will assume ordering of labels within a given context X as $Xn, Xn-1, ..., X1$ (lexically descending). An ordered set of labels from several execution contexts of the same attribute A_k, e.g. X, Y, Z, will be denoted using content inside curly braces, e.g. $\{X^*, Yi..j, Z^*\}$. We will consider the content within curly braces to be partially ordered from left to right, and the content within each context in the descending lexical order of their labels.

3.2.1 Device policies

A major distinction of the device type is whether the device is connected to the network or not, and in consequence whether it can grant the document any access to external services if asked for. Besides that devices may differ in details of local services they can provide to documents. By taking that into account we have distinguished ten types of devices specified in Table 3.6. Throughout the rest of this subsection we will specify policy rules for each device type specified in this table.

Workstation policy rules

Workstation is potentially the most powerfull device providing an execution context for active documents. For that reason preference may be given to

Table 3.6: Device types

Device	*Not Connected*	*Connected*
workstation	x11	x12
laptop	x21	x22
tablet	x31	x32
smartphone	x41	x42
cellphone	x51	x52

documents with functionality supporting quite advanced interaction with workers and execution devices, i.e. in J and D contexts, less often in W contexts. Because of their immobility, workstations may either be located in the worker's office or at home. When in office they are wired to the organization's intranet (I and N contexts), and less often have to connect directly to other networks (E and N contexts). When located at home they may still be wired to the worker's organization intranet through a VPN tunnel (I and N contexts), otherwise use modem connections (I or E combined with A or M contexts). Alternatively, the workstation may be not connected to any network. It may be a rare situation, however, e.g. a service break due to some routine maintenance work. Nevertheless, specificity of the local system configuration may become visible.

Security most often would involve T contexts, rarely P contexts. Finally reliability contexts may be any of H, F, B, or L, depending on the particular software used to handle active documents. However, the most likely H contexts seem to be. Based on the above, policy rules like the ones specified in Table 3.7, may be used to build the option tree for the workstation execution device.

Table 3.7: Workstation policy

Attribute	**Workstation**	
	not connected (x11)	*connected (x12)*
Performer	$\{J^*, D^*, W^*\}$	$\{J^*, D^*, W^*\}$
Availability	$\{J^*, D^*, W^*\} \rightarrow \{S7, S3..1\}$	$\{J^*, D^*, W^*\} \rightarrow \{I^*, E^*\}$
Performance	$\{S7, S3..1\} \rightarrow U^*$	$I^* \rightarrow N^*$, $E^* \rightarrow \{N^*, A^*, M^*\}$
Security	$U^* \rightarrow P^*$	$\{N^*, A^*, M^*\} \rightarrow \{T^*, P^*\}$
Reliability	$P^* \rightarrow \{H^*, F^*, B^*, L^*\}$	$\{T^*, P^*\} \rightarrow \{H^*, F^*, B^*, L^*\}$

Laptop policy rules

Laptops used today are not less powerfull execution devices than stationary workstations. Therefore if not connected to any network they provide the

same execution contexts as not connected workstations. A subtle difference, however, is that laptops are mobile, therefore may be outside of any network more often than workstations. On the other hand, if connected, they provide diversified execution contexts with regard to the performance attribute options (R contexts). In fact their policy rules are workstation policy rules extended by rules involving R contexts, in which preference is given to N before R. Another difference might be that laptops are in general more 'personal' than workstations, therefore a worker may want more control on what an active document does when performing its activity on 'his/her' personal device and may prefer W contexts before J and D. Additionally, personalization may also become visible when the laptop device is out of any network. We reflect that in our model with the availability context S options. Policy rules satisfying the above are specified in Table 3.8.

Table 3.8: Laptop policy

Attribute	**Laptop**	
	not connected (x21)	*connected (x22)*
Performer	$\{W^*, J^*, D^*\}$	$\{W^*, J^*, D^*\}$
Availability	$\{W^*, J^*, D^*\} \rightarrow \{S8, S6..1\}$	$\{W^*, J^*, D^*\} \rightarrow \{I^*, E^*\}$
Performance	$\{S8, S6..1\} \rightarrow U^*$	$I^* \rightarrow \{N^*, R^*\}$, $E^* \rightarrow \{N^*, R^*, A^*, M^*\}$
Security	$U^* \rightarrow P^*$	$\{N^*, A^*, M^*\} \rightarrow \{T^*, P^*\}$, $R^* \rightarrow \{C^*, T^*, K^*, P^*\}$
Reliability	$P^* \rightarrow \{H^*, F^*, B^*, L^*\}$	$\{C^*, T^*, K^*, P^*\} \rightarrow \{H^*, F^*, B^*, L^*\}$

Tablet policy rules

Tablets constitute a relatively new type of execution devices in between laptops and smartphones. They lack the most classic components of a personal computer, which have been for decades the keyboard and mouse, but owing to the advanced touch-up screen technology they can easily compensate for that with novel interaction paradigms. They also can just emulate the keyboard of a size compared to a small laptop. In general they are very 'personal' devices, therefore their software systems may be customized by their users to the point that some local tools or services commonly installed on workstations or laptops may not be available to documents willing to complete their activities as in the case of the latter. For that reason the dominant contexts with regard to the performer attribute would be W contexts, giving the worker full control over the document content. If, however, active documents can limit their expectations to a less demanding set of services, or can complete a simpler activity on their own, D and J contexts

may be accepted by the tablet device. Because of their highly customized local systems, tablets can provide most often only substitutes of specific tools required by the document. Therefore only $I4..6$ and $E4..6$ contexts (see Table 3.2) should be considered. Another issue is the networking hardware of tablets. In the thesis we have assumed that tablets enable only WiFi and ADSL modems, i.e. R and A contexts. Results of this analysis are listed in Table 3.9.

Table 3.9: Tablet policy

Attribute	**Tablet**	
	not connected (x31)	*connected (x32)*
Performer	$\{W^*, D^*, J^*\}$	$\{W^*, D^*, J^*\}$
Availability	$\{W^*, D^*, J^*\} \rightarrow S6..4$	$\{W^*, D^*, J^*\} \rightarrow \{I6..4, E6..4\}$
Performance	$S6..4 \rightarrow U^*$	$I6..4 \rightarrow R^*$, $E6..4 \rightarrow \{R^*, A^*\}$
Security	$U^* \rightarrow P^*$	$R^* \rightarrow \{C^*, T^*, K^*, P^*\}$, $A^* \rightarrow \{C^*, T^*, P^*\}$
Reliability	$P^* \rightarrow \{H^*, F^*, B^*, L^*\}$	$\{C^*, T^*, K^*, P^*\} \rightarrow \{H^*, F^*, B^*, L^*\}$

Smartphone policy rules

Contexts provided by smartphone devices differ when compared to tablet devices. Their touch-up screens can still enable interaction almost to the same extent as their tablet counterparts, including keyboard emulation, but due to their yet more customized functionality, many tools and services may never be provided to the worker or to the document when performing a specific activity. Therefore only $I3$ and $E3$ contexts are possible (see Table 3.2), with the same order of preferred contexts related to the performer attribute as in the case of tablet devices, i.e. W, D and J. For the same reason contexts related to interaction reliability would realistically be L. Networking hardware of smartphone devices enable access to wireless and cellular networks alike, however preference shall be given to R contexts before M contexts for economic reasons explained before. Based on this analysis we propose policy rules for smartphone execution devices as listed in Table 3.10.

Cellphone policy rules

The least capable execution devices are plain cellphones. They typically lack a touch up screen, have a significantly small screen, often with resolutions preventing using sophisticated graphics. Their imported software execution

Table 3.10: Smartphone policy

Attribute	**Smartphone**	
	not connected (x41)	*connected (x42)*
Performer	$\{W^*, D^*, J^*\}$	$\{W^*, D^*, J^*\}$
Availability	$\{W^*, D^*, J^*\} \rightarrow S3$	$\{W^*, D^*, J^*\} \rightarrow \{I3, E3\}$
Performance	$S3 \rightarrow U^*$	$I3 \rightarrow R^*$, $E3 \rightarrow \{R^*, M^*\}$
Security	$U^* \rightarrow P^*$	$R^* \rightarrow \{C^*, T^*, K^*, P^*\}$, $M^* \rightarrow \{C^*, T^*, P^*\}$
Reliability	$P^* \rightarrow L^*$	$\{C^*, T^*, K^*, P^*\} \rightarrow L^*$

capabilities are strongly limited. With the same preference of execution contexts related to the performer attribute options as in the case of smartphone devices, i.e. W, D and J, contexts related to the availability attribute are limited to just an external access to the worker's organization and without any tools or a keyboard available, i.e. $E1..2$ contexts (see Table 3.2). Because only a cellular network may be accessed by the cellphone device, its processor may be of less power (including battery limitations), and the device may have a SD card installed or not, its preferences of contexts related to the performance attribute would be $M2$ and $M1$ (see Table 3.3). Finally, interaction reliability shall be the lowest, i.e. $L1$. Based on the above analysis policy rules for the cellphone execution device may be like the ones listed in Table 3.11.

Table 3.11: Cellphone policy

Attribute	**Cellphone**	
	not connected (x51)	*connected (x52)*
Performer	$\{W^*, D^*, J^*\}$	$\{W^*, D^*, J^*\}$
Availability	$\{W^*, D^*, J^*\} \rightarrow S2..1$	$\{W^*, D^*, J^*\} \rightarrow E2..1$
Performance	$S2..1 \rightarrow U2..1$	$E2..1 \rightarrow M2..1$
Security	$U1 \rightarrow P^*$	$M2..1 \rightarrow \{T^*, P^*\}$
Reliability	$P^* \rightarrow L1$	$\{T^*, P^*\} \rightarrow L1$

3.2.2 Document policies

Objectives of document policies are in general different from the objectives of execution device policies. It may be seen from the analysis presented above that devices concentrate mainly on *what* minimal resources they can engage to satisfy documents asking for support when performing a given activity. They have no global information on the process in which each

document is involved – in many cases they may be even forbidden to get that information. Moreover devices often have to optimize usage of their resources when supporting many documents at the same time. On the other hand a document is interested in *how* well its activity could be completed in the context of the entire process it is involved in. For that reason a document may characterize its objectives with different preferences of attribute values that the device it arrives to and without any attempt to interfere with its internal policies – a common feature of open multi-agent systems [2]. In the Thesis three types of documents have been considered to define the document policy rules:

(S) documents with a *static* content; they are passive and unable to perform any operation on their own. A static content may be for example a plain text file, an image, a PDF or Word file without any macros embedded in it. The respective activity of their workflow can be performed only by the worker, who may read instructions what to do with the document right from its text.

(R) documents with a *reactive* content; they can perform specific operations in response to some stimulus, using their embedded code. A reactive content may be for example an interactive form that can check lexical correctness of input typed by the worker in its respective fields or input data automatically from the local worker's files in reaction to pressing by the latter the 'browse' activity button, a PowerPoint presentation with embedded animations, or a more sophisticated piece of code capable of doing various operations on opening, like self-extraction of the zipped content, even a virus infected PDF document [14]. The respective activity of their workflow can be performed only by the worker or jointly by the worker and the document.

(P) documents with a *proactive* content; they can initiate and perform various operations on their own, and are capable of controlling interaction with human users or systems. A proactive document may be for example a Flash ad popping up on a Web page, a Web crawler collecting data [48], or a document-agent capable of recognizing the face of its recipient [61]. Depending of the semantics of the respective activity of its workflow it may be performed only by the worker, jointly by the worker and the document, or automatically by the document alone.

Results of the analysis of various combinations of options specified above are listed in Table 3.12. It may be seen that policies of reactive documents are a proper subset of the proactive document policies, and respectively policies

of static documents are a proper subset of the reactive document policies. Indeed, document-agents may be designed by their originators to play various roles in the business process they implement. For example if a specifics service required by the proactive document to process its content in not available at the current execution device it may provide its content for manual editing by the worker. Therefore, not like in the case of device policy rules listed in Tables 3.7-3.11 we do not assume any specific ordering of options defined before in Tables 3.1-3.5 in the document policy rules. We denote that they may be ordered arbitrary by symbol '|'. That order may change dynamically for the same document from device to device with regard to the dynamics of the process they implement. For example, due to some internal deadline the document may choose to complete its activity on the currently available device that prefers the W context instead of to refuse to cooperate with the device currently used by the worker and wait until he/she will be using another device, possibly preferring the J context.

3.3 Bargaining over option trees

Below we combine development of Chapter 1 with the concept of types of execution devices and documents and their respective policy rules introduced in this chapter. Note that option trees of two players determine a set of all offers they may be bargained over. Formally, if C_i and C_{-i} denote respective sets of offers in which P_i and its opponent P_{-i} are interested, bargaining set $C = C_i \cap C_{-i}$. Agents agree to share such a set prior to each encounter by performing the operation $C = \tau_{P_1} \cap \tau_{P_2}$. In order to find all bargaining sets of interest in the system of MIND documents each respective set of offers in the option tree built with policy rules for the each execution device type should be combined with each respective set of offers in the option tree built with policy rules for each respective document type. Moreover, the order of offers in the option tree implicitly specifies utility of each respective offer to the respective party. Of course the order of offers in each player's option tree is not known to its opponent, but may be deduced by observing counteroffers of the latter returned during negotiations, as proposed in Chapter 1.

Let us consider a simplified SBG between a connected laptop (x22) and the proactive document, which is encoded and light, i.e. not imposing significant load on the execution device, with a bargaining set consisting of just

Table 3.12: Document types and policies

Type	*Policy*				
	Performer	**Availability**	**Performance**	**Security**	**Reliability**
Static	W^*	$W^* \rightarrow \{S^*\|E^*\|I^*\}$	$\{S^*\|E^*\|I^*\} \rightarrow \{U^*\|M^*\|A^*\|R^*\|N^*\}$	$\{U^*\|M^*\|A^*\|R^*\|N^*\} \rightarrow \{P^*\|K^*T^*\|C^*\}$	$\{P^*\|K^*T^*\|C^*\} \rightarrow \{L^*\|B^*\|F^*\|H^*\}$
Reactive	$\{J^*\|W^*\}$	$\{J^*\|W^*\} \rightarrow \{S^*\|E^*\|I^*\}$	$\{S^*\|E^*\|I^*\} \rightarrow \{U^*\|M^*\|A^*\|R^*\|N^*\}$	$\{U^*\|M^*\|A^*\|R^*\|N^*\} \rightarrow \{P^*\|K^*T^*\|C^*\}$	$\{P^*\|K^*T^*\|C^*\} \rightarrow \{L^*\|B^*\|F^*\|H^*\}$
Proactive	$\{D^*\|J^*\|W^*\}$	$\{D^*\|J^*\|W^*\} \rightarrow \{S^*\|E^*\|I^*\}$	$\{S^*\|E^*\|I^*\} \rightarrow \{U^*\|M^*\|A^*\|R^*\|N^*\}$	$\{U^*\|M^*\|A^*\|R^*\|N^*\} \rightarrow \{P^*\|K^*T^*\|C^*\}$	$\{P^*\|K^*T^*\|C^*\} \rightarrow \{L^*\|B^*\|F^*\|H^*\}$

five offers:

$$
\begin{aligned}
o_1 &= < D3, E1, A4, T4, H3 >, \\
o_2 &= < D3, E1, M4, T4, H3 >, \\
o_3 &= < D3, E1, R2, K4, F1 >, \\
o_4 &= < D3, E1, R4, C4, H1 >, \\
o_5 &= < D3, I1, N4, T4, H4 > .
\end{aligned}
$$

Nodes of option trees in Figures 3.2 and 3.3 are labeled with the attribute value labels listed in the respective Tables 3.1-3.5, while utility of each individual attribute value to the player is specified in square brackets. Utilities of the respective offers of the bargaining set are listed at the bottom of the tree in a normalized form, according to Formula 1.6.

Figures 3.2 and 3.3 indicate conflicting preferences of the execution device: the active document willing to do everything on its own, most preferably from outside of its home organization network, whereas the device prefers using its company network.

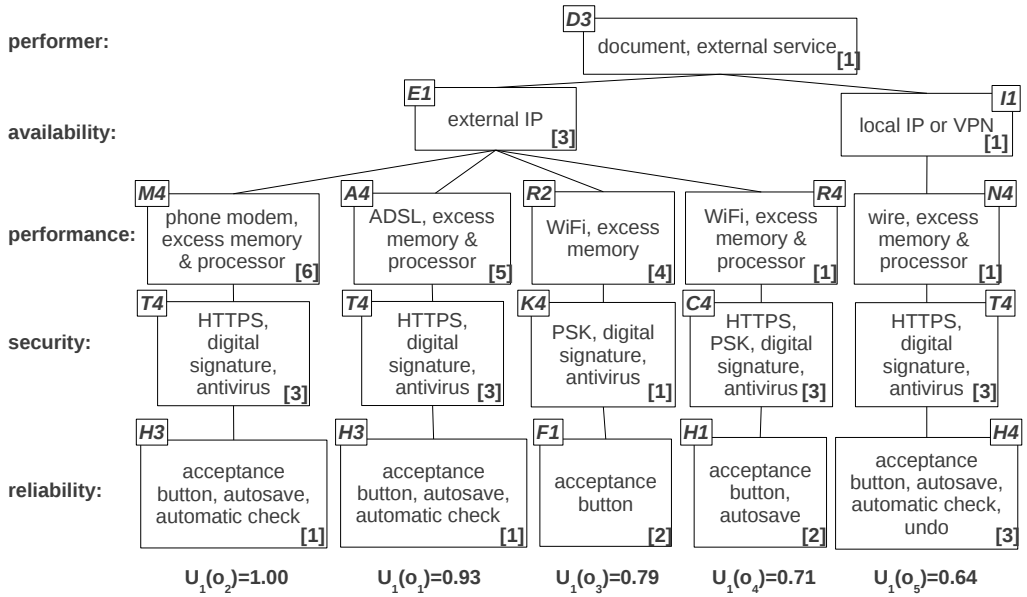


Figure 3.2: Example option tree of a proactive document

It can be seen that negotiations over multi-item offers, is a non-zero game, where each party can win some wealth, even when starting from conflicting offers. A SBG scheme defined in Chapter 1 can resolve such a conflict and enable finding by the document and the execution device an offer in the

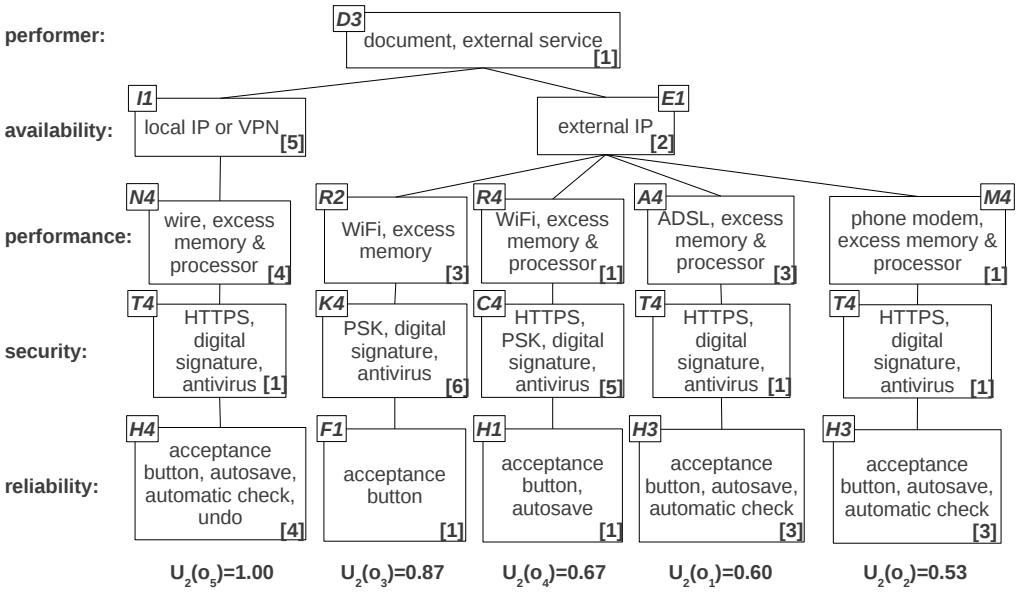


Figure 3.3: Example option tree of a connected laptop

Table 3.13: Bargaining over example option trees

Stage	Player	Offer o_i	$U_i(o_i)$	$U_i(o_i)$
1	P_1	o_2	1.00	0.53
	P_2	o_5	1.00	0.64
2	P_1	o_1	0.93	0.60
	P_2	o_3	0.87	0.79
3	P_1	o_3	0.79	agreed

bargaining set that either party accepts as the one of as much of utility as it can get in the current execution context. It can be seen in Table 3.13 that the document (player P_1) started negotiations by offering to the execution device (player P_2) its most valuable option o_2. Then after several offers and counteroffers option o_3 offered by the device was accepted by the document.

In the above example we have used simplified option trees to keep them small. In real applications involving MIND documents the upper bound for the maximum size of a single option tree could be as high as the product of the numbers of attribute value labels used in Tables 3.1-3.5, which is well over 5000, whereas the upper bound for the number of possible negotiation histories is a product of their respective permutations, in the order of magnitude of 10^{13}. Two questions we would like to answer in the next chapter are:

1. whether machine learning techniques proposed in Chapter 2 can effectively shorten the negotiation process, i.e. document agents can agree contracts with execution devices in a reasonably short negotiation process, and

2. when traveling over a distributed system of execution devices, document agents would not be required to carry their negotiation histories with them, or alternatively would not require access to the network any time they need to consult some external service providing access to such histories.

Chapter 4

Negotiation protocols

At the beginning of this Chapter we would like to recall our problem and methods we have been using to resolve it so far: documents travel among network nodes and are opened (executed) on various devices. In order to do that and take advantage of the device resources, the document as well as the device are equipped with option trees which describe their capabilities. The option trees are then negotiated between the document and the device in order to choose the best option.

In Chapter 1 we have introduced the concept of a simple bargaining game (SBG) and defined its rules. We have also introduced two algorithms: Algorithm 1.2 and 1.4, thanks to which SBG players can find a solution to Equation 1.7. The desired feature of SBG is that its underlying protocol would prefer agreements that could be reached faster. One of the methods that can speed up the negotiation process is to enable players to learn their opponents' preferences. They could do this if they can meet more than once. Then, when the player meets its opponent again, it should be able to guess the contract faster.

In Chapter 2 we have investigated several artificial intelligence methods to use them in the classification process. We have chosen artificial neural networks as the best candidate to reach the above mentioned goal. Below we take advantage of neural networks in teaching the document agents to recognize preferences of execution devices by policies defined in Chapter 3.

In Chapter 2 we have also introduced the concept of coding the training sets, used to train the neural network. Next, in Chapter 3 we have defined the concept of a sequence of offers, which we will call briefly a *sequence* (see Definition 11). By combining that we define training sets to contain sequences, originating from a player which builds its offers using one of the policies, defined in Chapter 3. Sequences have to be transformed into a form required by the given classification task, as described in Chapter 2. There

are two kinds of tasks we are interested in: *grouping* of data and *recognizing* the sequences.

4.1 Bargaining sets, sequences and training data

Our investigations will be focused on four domains of the application of machine learning techniques in order to improve negotiation processes:

1. *occurrence vector clustering* enables grouping devices that belong to the same device classes based on appearances of the given symbols in offers,

2. *sequence clustering* enables grouping of devices basing on ordering of sequences. If the document already knows the class of the device it negotiates with, the next two approaches may be applied:

3. *sequence prediction*, which relies on guessing of ordering of sequences,

4. *contract prediction*, which relies on guessing the contract based on the guessed sequence order.

In the current Chapter we will introduce algorithms that use AI. Before doing that we have to describe how to keep offers in bargaining sets and sequences; we will also explain how to prepare data for the learning algorithms.

We have mentioned in Chapter 2 that prior to training the network, the historical bargaining sets have to be first transformed to the occurrence vectors. We defined Algorithm 4.1 for doing that. In order to explain it, let us first consider data presented in Table 4.1, which are names of attribute values and their encoded equivalents. Policies, first presented in Chapter 3, determine which symbols are allowed to occur in each offer. Understanding of encoding of offers is crucial to understand the work of Algorithm 4.1.

4.1.1 Attribute values encoding

Recall Formula 2.10 which is the neuron equation. The equation determines inputs of neurons as numeric values. Thus, symbols cannot be introduced directly into the input of a neural network but they first need to be converted into the numeric form. The conversion is specified by Formulas 2.16 and 2.17.

A bargaining set contains offers, each one consisting of elements which are attribute values. According to Chapter 3, the offer consists of five elements, each one corresponding to one attribute.

Let us define Function *enc* that encodes labels of attribute values using values from Table 4.1. The table includes all symbols used in the examples here (the current MIND implementation)

$$enc : A_k \to \mathbb{Z}_{\geq 0} \tag{4.1}$$

where A_k is the set of the k-th attribute values.
Since $A_k = \{a_k^1, \ldots, a_k^{|A_k|}\}$, then $o = \langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ where n is the number of attributes. So for $k \in 1 \ldots n$ we have $m_k \in 1, \ldots, |A_k|$. Thus $m_k = enc(a_k^{m_k})$.

Table 4.1: Encoding of attribute values

Attribute	*Encoding*
Performer	$D1 = 0$; $D2 = 1$; $D3 = 2$; $J1 = 3$; $J2 = 4$; $J3 = 5$; $J4 = 6$; $W1 = 7$; $W2 = 8$
Availability	$E1 = 0$; $E2 = 1$; $E3 = 2$; $E4 = 3$; $E5 = 4$; $E6 = 5$; $E7 = 6$; $I1 = 7$; $I2 = 8$ $I3 = 9$; $I4 = 10$; $I5 = 11$; $I6 = 12$; $I7 = 13$; $S1 = 14$; $S2 = 15$; $S3 = 16$ $S4 = 17$; $S5 = 18$; $S6 = 19$; $S7 = 20$
Performance	$A1 = 0$; $A2 = 1$; $A3 = 2$; $A4 = 3$; $M1 = 4$; $M2 = 5$; $M3 = 6$; $M4 = 7$; $N1 = 8$; $N2 = 9$; $N3 = 10$; $N4 = 11$; $R1 = 12$; $R2 = 13$; $R3 = 14$; $R4 = 15$; $U1 = 16$; $U2 = 17$; $U3 = 18$; $U4 = 19$
Security	$C1 = 0$; $C2 = 1$; $C3 = 2$; $C4 = 3$; $K1 = 4$; $K2 = 5$; $K3 = 6$; $K4 = 7$; $P1 = 8$ $P2 = 9$; $P3 = 10$; $P4 = 11$; $T1 = 12$; $T2 = 13$; $T3 = 14$; $T4 = 15$
Reliability	$B1 = 0$; $B2 = 1$; $B3 = 2$; $B4 = 3$; $F1 = 4$; $F2 = 5$; $F3 = 6$; $F4 = 7$; $H1 = 8$; $H2 = 9$; $H3 = 10$; $H4 = 11$; $L1 = 12$; $L2 = 13$; $L3 = 14$; $L4 = 15$

Function given by Equation 4.1 encodes values of the given attributes with numbers like the ones from Table 4.1. The table presents names of attribute values and their encoded equivalents. For example, when substituting symbols in offer $o_1 = \langle W1, S7, A1, C1, L1 \rangle$, by numeric values from Table 4.1 we get $enc(o_1) = \langle 7, 20, 0, 0, 12 \rangle$.

Let us introduce function

$$\Upsilon : A_k \to \mathbb{Z}_{\geq 0} \tag{4.2}$$

which takes the attribute value and returns its index in the occurrence vector defined in Chapter 2. According to Equation 1.2 attributes defined in Chapter 3 will be indexed by v as shown in Table 4.2.

The rightmost column specifies sizes of their respective sets of attribute values specified in Tables 3.1 – 3.5.

The attributes presented in Table 4.2 can have following values: $A_1 = \{a_1^1, a_1^2, \ldots, a_1^9\}$, $A_2 = \{a_2^1, a_2^2, \ldots, a_2^{21}\}$ etc..

Given Table 4.2 function Υ can be calculated as:

$$\Upsilon(a_k) = \begin{cases} enc(a_k) & \text{if } k = 1 \\ \Upsilon(a_k) + \sum_{i=1}^{k} |A_i| & \text{if } k > 1 \end{cases} \tag{4.3}$$

Table 4.2: Attribute value sets

Index k	*Attribute name*	*Set* A_k	*Size* $\lvert A_k \rvert$
1	Performer	A_1	9
2	Availability	A_2	21
3	Performance	A_3	20
4	Security	A_4	16
5	Reliability	A_5	16

4.1.2 Creation of the occurrence vector

Now, when we know how an offer is constructed, we are able to explain how to transform it into an occurrence vector. All steps necessary to transform training data into the required form are included in Algorithm 4.1. Procedure `CreateOccurrenceVector()` in Algorithm 4.1 takes a collection of offers as well as the number of all attribute values and returns the occurrence vector.

The first step of Algorithm 4.1 is a creation of the empty occurrence vector which size is the number of attribute values and then initializing its elements by assigning them 0 value.

Algorithm 4.1: createOccurrenceVector()

```
1  Data: input bargaining set BargainingSet,
2  number of all possible attribute values attributeValueNumber
3  Result: the returned occurrence vector
4  occurrenceVector ←new occurrenceVector(attributeValueNumber);
5  for i ← 1 to length(occurrenceVector) do
6      occurrenceVector [i]← 0;
7  foreach offer in BargainingSet do
8      // attribute values are encoded
9      foreach attributeValue in offer do
10         // procedure getAttributeValueIndex() is
11         // the implementation of function ϒ()
12         index ← getAttributeValueIndex(attributeValue);
13         occurrenceVector [index ]← 1;
14 return occurrenceVector;
```

For each offer in the input collection, each offer element is taken. The element is an attribute value, encoded in the way as explained before. Procedure `getAttributeValueIndex()`, which implements Formula 4.3 and returns an index in the `occurrenceVector`, is called then with the attribute value.

Next value 1 is assigned to the element of the occurrence vector indicated by the returned index. This value indicates that the given attribute value occurs in the offer. After examining all offers in the bargaining set the complete occurrence vector is returned.

The following example demonstrates how the algorithm works. Consider the bargaining set presented in Table 4.3. The attribute values in the respective offers $o_1, \ldots, o_8$ have been encoded using values from Table 4.1.

In order to transform the bargaining set in Table 4.3 to an occurrence vector, procedure `createOccurrenceVector()` is called. The sum of sizes of all attribute values in the current MIND implementation is 82 (see Table 4.2), so the length of the output occurrence vector will be 82 (see Algorithm 4.1).

Table 4.3: An example of the encoded bargaining set

Bargaining set	
Offer	*Encoded offer*
o_1	$\langle 1, 16, 18, 10, 7\rangle$
o_2	$\langle 4, 19, 18, 10, 1\rangle$
o_3	$\langle 3, 15, 17, 10, 2\rangle$
o_4	$\langle 0, 14, 17, 9, 7\rangle$
o_5	$\langle 6, 17, 17, 9, 13\rangle$
o_6	$\langle 8, 16, 18, 9, 12\rangle$
o_7	$\langle 7, 14, 17, 11, 12\rangle$
o_8	$\langle 4, 18, 17, 8, 2\rangle$

After creation of occurrence vector β, the algorithm takes the first offer which can be also written as $o_1 = \langle a_1^1, a_2^{16}, a_3^{18}, a_4^{10}, a_5^7\rangle$. The first attribute value in the offer is 1. The attribute value is then sent as the argument of procedure `getAttributeValueIndex()`. The result is $\Upsilon(a_1^1) = 1$. So, the element in the occurrence vector indexed by 1 receives value 1.

The next element of the offer is 16, thus, the element which has index $\Upsilon(a_2^{16}) = 25$ receives value 1. The step is repeated on each element of the offer. Next, offer o_2 is taken. The first element of o_2 is 4. So, the element which has index $\Upsilon(a_1^4) = 4$ in the occurrence vector receives value 1. Then, the next element of the offer is taken, which is 19. Since $\Upsilon(a_2^{19}) = 28$, then bit b_{28} obtains value 1. Then the next attribute value is taken and so on, until all elements of all offers are examined.

4.1.3 Occurrence vector training

The term 'clustering' is generally reserved for unsupervised learning. However in this Thesis we use it for recognizing devices without distinguishing the method.

In Chapter 3 we have distinguished ten device types. The device policies have been presented in Tables 3.7 – 3.11. Because policies of the not connected workstation ($x11$) and not connected laptop ($x21$) make use of the same symbols, we may consider them for a while to be in the same device class; in consequence only nine device classes have to be considered further. Each device class obtains a label. Table 2.10 shows how the labels look like in case of three hypothetical classes of devices indicated in Table 2.1.

Occurrence vector based learning is illustrated in Figure 4.1.

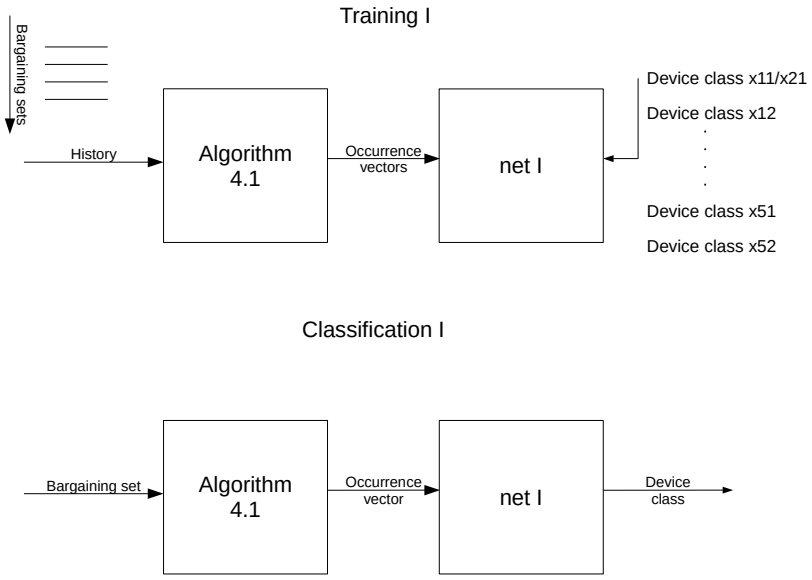


Figure 4.1: Training and classification of occurrence vectors

Despite the fact of distinguishing ten device types (see Table 3.6), the occurrence vector learning discovers only nine device classes. It is caused by the fact that $x11$ and $x21$ device classes utilize the same symbols, thus they are not distinguishable by this method. Therefore we have decided to join these two classes temporarily and so $x11/x21$ device class has been created. Later, $x11$ and $x21$ device classes will be distinguished using another approach (see Section 4.1.4).

Before the appropriate training process can start, bargaining sets have to be converted into occurrence vectors by Algorithm 4.1. Next, neural network $netI$ with 9-element output layer is created. Then, in the training process,

occurrence vectors are associated with the proper device classes by binding them with the proper label of the device class. The labels are the positional representation of integer numbers. The positional notation of the integer numbers relies on substituting the number k by n-element bit-word in which the bit on the k-th position has value 1 and the rest of bit fields have value 0 and where $k, n \in \mathbb{N}$ and $k \leq n$.

So, when $x11/x21$ device class has the label which is equal 1, then its positional representation is 9-th element bit word (9 is the number of the device classes), which field indexed by 1 has value 1 and the rest of fields has value 0. Further, $x12$ device class has the index 2. Thus the 2-nd field of its positional representation has the value 1„ and the rest 0, and so on. The training process relies on binding occurrence vectors with the target outputs which are the labels of device classes.

In the experiments reported further in Chapter 5, the satisfactory results have been reached when $netI$ consisted of two hidden layers, built respectively of 20 and 10 neurons and the output layer built of 10 neurons. As shown in Chapter 5, the training with occurrence vectors built of 10 bargaining sets was sufficient to produce the network capable of recognizing bargaining sets without any error.

4.1.4 Sequence recognition

There are two goals of recognizing of sequences:

1. the goal of sequence clustering is the recognition of device which the document-agent negotiate with by matching up a sequence created by incoming offers to one of the guessed sequences assigned to the specified device class,

2. the goal of sequence prediction is the prediction of the opponent's move by guessing its sequence.

Chronologically p.1 precedes p.2, since before sequence learning opponent's device has to be recognized. However, in the logical order p.2 precedes p.1, because a sequence before matching has to be guessed. So we start the description from p.2.

The rules of SBG presented on page 15 and the definition of sequence on page 55 determine that the maximum length of sequence is $\lceil |C|/2 \rceil$. Nevertheless, let us assume that for simplicity in the current Chapter we assume that sequence length equals bargaining set size. In Chapter 5 we will test sequences which lengths are shorter than bargaining set sizes as well as sequences which lengths equal the bargaining set size.

Classification of offers using neural networks

From now we will distinguish two concepts: *policy rules* and the *policy implementation*. The *policy implementation* is a policy with assigned utilities of attribute values which conform to the requirements of the policy. The learning process is based on training data which are the recorded histories of games. The data may come from two possible sources:

1. They may come from the existing MIND agent system (the *a posteriori approach*).

2. If there is no existing MIND system yet, training data may be generated with any values of utilities that satisfy the predefined policies (the *a priori approach*).

Case 1 is straightforward – if the agent system already works, MIND agents leave behind them the historical data, which can be collected and used for training, e.g. in the agency when they rest.

In Case 2 we have to use data generated from the training example policy implementations. Two example policy implementations are presented in Tables 4.4 and 4.5. However, in order to use them in the learning process, we have to remember that two approaches to learning are possible. The first one may take advantage of that some attribute values occur in a given policy, while others do not. The second one may exploit the order of offers in sequences to recognize policies they belong to as well as to guess the contract of a higher utility. In the first case training with generated examples has the same value as training with examples from an existing MIND system.

However, the order of offers in a sequence can differ from the order of offers in another sequence if they were generated from different policy implementations, even if they were implementations of the same policy.

Consider the alternative *x11* policy implementation to the one shown in Table 4.4 and denote the two implementations respectively by Ξ'_{x11} and Ξ''_{x11}. In Ξ''_{x11} utilities of attribute *Performance* have the following values: $U4 = 0.2$, $U3 = 0.19$, $U2 = 0.18$ and $U1 = 0.17$ and the rest of attributes have the same utilities as in Ξ'_{x11}. Now consider two offers: $o_i = \langle J4, S2, U3, P1, H1\rangle$ and $o_j = \langle J3, S2, U4, P1, H1\rangle$. In the case of Ξ'_{x11} policy implementation $U_{\Xi'_{x11}}(o_i) = 0.2 + 0.15 + 0.15 + 0.05 + 0.1625 = 0.7125$ and $U_{\Xi'_{x11}}(o_j) = 0.175 + 0.15 + 0.2 + 0.05 + 0.1625 = 0.7375$. In Ξ''_{x11} utilities $U_{\Xi''_{x11}}(o_i) = 0.2 + 0.15 + 0.19 + 0.05 + 0.1625 = 0.7525$ and $U_{\Xi''_{x11}}(o_j) = 0.175 + 0.15 + 0.2 + 0.05 + 0.1625 = 0.7375$. Thus, despite the fact that Ξ'_{x11} and Ξ''_{x11} are implementations of the same $x11$ policy, $U_{\Xi'_{x11}}(o_i) < U_{\Xi'_{x11}}(o_j)$ but $U_{\Xi''_{x11}}(o_i) > U_{\Xi''_{x11}}(o_j)$.

Table 4.4: Example policy implementation of the $x11$ device class

Attribute	*Utilities*
Performer	$J4 = 0.2$; $J3 = 0.175$; $J2 = 0.15$; $J1 = 0.125$; $D3 = 0.1$; $D2 = 0.075$; $D1 = 0.05$; $W2 = 0.025$; $W1 = 0.0125$
Availability	$S7 = 0.2$; $S3 = 0.175$; $S2 = 0.15$; $S1 = 0.125$; $S6 = 0.1$; $S4 = 0.075$; $S5 = 0.05$
Performance	$U4 = 0.2$; $U3 = 0.15$; $U2 = 0.1$; $U1 = 0.05$
Security	$P4 = 0.2$; $P3 = 0.15$; $P2 = 0.1$; $P1 = 0.05$
Reliability	$H4 = 0.2$; $H3 = 0.1875$; $H2 = 0.175$; $H1 = 0.1625$; $F4 = 0.15$; $F3 = 0.1375$; $F2 = 0.125$; $F1 = 0.1125$; $B4 = 0.1$; $B3 = 0.0875$; $B2 = 0.075$; $B1 = 0.0625$ $L4 = 0.05$; $L3 = 0.0375$; $L2 = 0.025$; $L1 = 0.0125$

The above example indicates that assuming the opponent's policy is not enough to predict its sequence since the same policy can provide different partial orderings of offers belonging to the same bargaining set. The policy implementation decides on ordering of the sequence and the exploration of the policy implementation is the unattainable goal of the sequence prediction.

Table 4.5: Example policy implementation of the $x21$ device class

Attribute	*Encoding*
Performer	$W2 = 0.2$; $W1 = 0.175$; $J4 = 0.15$; $J3 = 0.125$; $J2 = 0.1$; $J1 = 0.075$; $D3 = 0.05$; $D2 = 0.025$; $D1 = 0.0125$
Availability	$S7 = 0.2$; $S6 = 0.175$; $S5 = 0.15$; $S4 = 0.125$; $S3 = 0.1$; $S2 = 0.075$; $S1 = 0.05$
Performance	$U4 = 0.2$; $U3 = 0.15$; $U2 = 0.1$; $U1 = 0.05$
Security	$P4 = 0.2$; $P3 = 0.15$; $P2 = 0.1$; $P1 = 0.05$
Reliability	$H4 = 0.2$; $H3 = 0.1875$; $H2 = 0.175$; $H1 = 0.1625$; $F4 = 0.15$; $F3 = 0.1375$; $F2 = 0.125$; $F1 = 0.1125$; $B4 = 0.1$; $B3 = 0.0875$; $B2 = 0.075$; $B1 = 0.0625$ $L4 = 0.05$; $L3 = 0.0375$; $L2 = 0.025$; $L1 = 0.0125$

Table 4.5 provides an example of $x21$ policy implementation. The values will be utilized in our further considerations in the current chapter when the sequences are compared.

Sequence prediction

The idea of sequence prediction is illustrated in Figure 4.2. Sequence recognition differs from the occurrence vector clustering in that, for the latter so many networks are created as many device classes are available. The upper part of the figure shows how the network is trained: the sequences are converted to pairs and relations. The pairs are the input of the neural network, the relations are its target. The lower part shows the classification: the bargaining set is converted to pairs, next the relations are recreated using the

neural network. At last the pairs and the relations are converted to sequences which are utilized by the negotiating algorithm.

Table 4.6: An example of the encoded sequence

Sequence		
Offer	*Encoded offer*	*Utility priority*
o_1	$\langle 1, 16, 18, 10, 7\rangle$	8
o_2	$\langle 4, 19, 18, 10, 1\rangle$	7
o_3	$\langle 3, 15, 17, 10, 2\rangle$	6
o_4	$\langle 0, 14, 17, 9, 7\rangle$	5
o_5	$\langle 6, 17, 17, 9, 13\rangle$	4
o_6	$\langle 8, 16, 18, 9, 12\rangle$	3
o_7	$\langle 7, 14, 17, 11, 12\rangle$	2
o_8	$\langle 4, 18, 17, 8, 2\rangle$	1

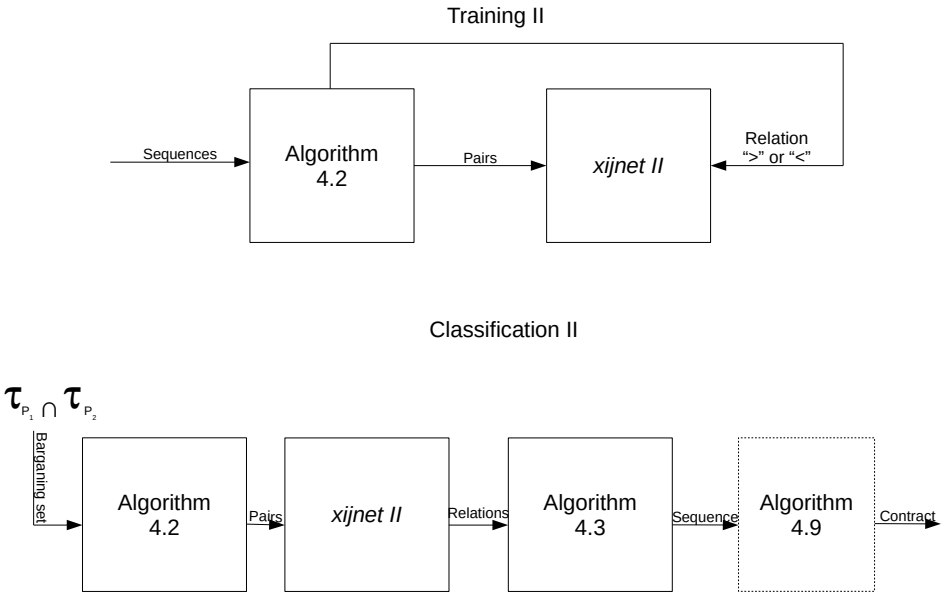


Figure 4.2: Classification of sequences

Sequence to relation

Recall the definition of sequence (Definition 11). Sequence is the order of submitted offers. The offers are submitted by the player which uses Algo-

rithm 1.2 or 1.4. If we analyze the algorithms, we will see that the order of offers is determined by their utilities. First the highest offer is submitted, then the next one. Nevertheless, the rule has an exception. If the player accepts its opponent's offer, then its last submitted offer is a repeat of the opponent's offer and it may get out of the rule (see the rules of *SBG*). However, since it is not difficult to find the acceptance offers and remove them from the sequences, we omit the exception from our further investigations and assume that $\forall_{o_a,o_b \in \varsigma_i} a < b \Rightarrow U_i(o_a) \geq U_i(o_b)$, where $a, b \in \mathbb{N}$ are indexes of offers in sequence ς.

Now consider Table 4.6, which shows how sequences are stored in files. One file stores the information shown in column *Encoded offer*. Column *Utility priority* stores the information about the priority of offer. Utility priority function $\Gamma : o \rightarrow \mathbb{N}$ reflects the order of offers in a sequence (the function assigns the offer to a number): the higher the priority the higher its position in the sequence.

$$\forall_{o_1,o_2 \in \varsigma_i} U_i(o_1) > U_i(o_2) \Rightarrow \Gamma(o_1) > \Gamma(o_2) \tag{4.4}$$

The difference between utility and priority underlies the fact that the first is hidden for the player's opponent while the second is public.

The training of the network is performed as follows: first sequences assigned to the given device are taken from the stored histories. Then, each sequence is converted to the relational form using Algorithm 4.2. Next, the obtained pairs are associated with appropriate relations (relation '>' or '<') according to the result returned by Algorithm 4.2 by binding the pairs with the relations in network *xijnetII* inputs and outputs, where *xij* denotes the respective device class, specified in Table 3.6.

Algorithm 4.2 can be utilized to convert sequences into pairs and relations as well as to convert bargaining sets into pairs. It works in the following way: a sequence is stored in two data structures: one the collection of n offers, and another the n element array of priorities of the respective offers in the sequence. The higher is the offer the higher the number in the array – if the algorithm is used to convert the n-element bargaining set, the input is the collection of n offers in the bargaining set and n element array of integers from 1 to n. In general the steps are:

1. two new empty collections are created for storing the pairs of offers and relations associated with them,

2. the priority of each offer in the input sequence is compared with the given priority in the priorities array of all other offers,

Algorithm 4.2: sequence2Relations()

```
1  Data: input collection offers,
2  priorities of offers in sequence priorities,
3  Result: the collection of pairs pairs,
4  the collection of relations relations
5  relations ← new relations ();
6  pairs ← new pairs ();
7  for i ← 0 to length(offers)−1 do
8      for j ← 0 to length(offers)−1 do
9          if j ≠ i then
10             if priorities[i] ≠ priorities[j] then
11                 pair ← concatenate(offers[i], offers[j]);
12                 pairs.add (pair);
13                 relations.add (1);
14                 pair ← concatenate(offers[j], offers[i]);
15                 pairs.add (pair);
16                 relations.add (0);
17 return (pairs, relations);
```

3. the new pair is created by concatenating the offer with the compared offer,

4. if the priority of the first offer is higher than the second one, the new pair is associated with relation '>' which will be stored as 1,

5. if the priority of the first offer is smaller than the second one, the new pair is associated with relation '<' which will be stored as 0.

The output of Algorithm 4.2 is a two element tuple of the collection of pairs and the collection of associated relations. If Algorithm 4.2 is utilized to convert a bargaining set, we are interested only in the pairs.

The training relies on binding a pair of offers, which is written as a concatenation of two offers, with a target output, which is relation 0 or 1.

The following example can illustrate that:

Example 4. *Consider offers* $o_1 = \{1, 16, 18, 10, 7\}$ *and* $o_2 = \{4, 19, 18, 10, 1\}$ *belonging to the sequence from Table 4.6. They are concatenated and the pair is created* $\zeta_1 = \langle 1, 16, 18, 10, 7, 4, 19, 18, 10, 1\rangle$. *Since* $o_1 > o_2$, *the pair is labeled by* 1. *Another pair* $\zeta_2 = \langle 4, 19, 18, 10, 1, 1, 16, 18, 10, 7\rangle$ *can also*

be created for offers o_1 *and* o_2 *. Relation '<' between the offers results in creating label* 0*. The neural network is trained using pairs* ζ_1*,* ζ_2 *and labels* 1 *and* 0 *as the targets.*

Relation to sequence

If we want to recreate the sequence, we need the bargaining set and the trained network. The bargaining set has to be converted to the pairs of offers using Algorithm 4.2. The result is the collection of pairs and the collection of relations. The collection of relations may be omitted, since the appropriate relations will be reproduced using the neural network. So the relations are the output of the network, which input are pairs. The process is outlined in Figure 4.2.

In the upper part of the figure, which illustrates the training of the network, we can see sequences isolated from the negotiation history, then converted into pairs and relations by using Algorithm 4.2.

The lower part of the figure shows the process of guessing the opponent's sequence in order to boost the negotiation process. The first step is creating the bargaining set by intersection of option trees of the document and the device. We denote it in Figure 4.2 symbolically as $\tau_{P_1} \cap \tau_{P_2}$, where τ_{P_1} and τ_{P_2} are option trees. The algorithm for doing that was described in [35]. Then, the bargaining set is converted into pairs using Algorithm 4.2. Next, the respective network is used to recognize the relations between the pairs. Finally, the pairs and the obtained relations are converted into the sequence form by Algorithm 4.3.

Algorithm 4.3 is the new version of the algorithm outlined in Figure 2.4. The algorithm presented in Chapter 2 was based on inserting offers to the list. Algorithm 4.3 omits problems connected with inserting values into the middle of the collection.

The procedure `relations2Sequence()` implementing Algorithm 4.3 takes a collection of pairs and relations associated with them. The first step is to create an array containing all offers that occur in pairs. Next the array which holds priorities of offers is created. After that it has to be counted how many times each offer occurs at the first place in the pair associated with relation 1. Obtained numbers mean the priorities of offers. However, we have respected only relations '1'. To use relations '0' we have to count zeros in the same way as ones. Then the results have to be reversed in order to transform the numbers of occurrences of '0' in the numbers of occurrences of '1'. Next both obtained arrays have to be merged. The procedure returns the sequence the array of offers and the array of priorities.

Algorithm 4.3: relations2Sequence()

```
1  Data: input relations relations,
2  input more or less signs represented by 0 and 1 signs,
3  Result: the collection of offers offers,
4  the array of priorities of offers priorities
5  offers ← getAllOffers (relations);
6  n ←length(offers);
7  // a collection which stores numbers of occurrence
8  // of respective offers
9  prioritiesOne ← new prioritiesOne[n];
10 for i ← 1 to length(pairs) do
11     pair ← pairs[i];
12     if relations[i] = 1 then
13         // if the first clause of the pair is found among
14         // offers, an associated value is incremented
15         index ← findIndex(getOffer(1, pair), offers);
16         if index > −1 then
17             prioritiesOne[index] ← prioritiesOne[index] + 1;
18 // Do the same with relation 0
19 prioritiesZero ← new prioritiesZero[n];
20 for i ← 1 to length(pairs) do
21     pair ← pairs[i];
22     if relations[i] = 0 then
23         // if the second clause of the pair is found among
24         // offers, an associated value is incremented
25         index ← findIndex(getOffer(2, pair), offers);
26         if index > −1 then
27             prioritiesZero[index] ← prioritiesZero[index] + 1;
28 m ← max(prioritiesZero);
29 // We reverse the occurrences of zeros in order to obtain
30 // occurrences of ones.
31 foreach priority in prioritiesZero do
32     priority ← m − priority;
33 // Priorities retrieved from ones and zeros are merged
34 for i ← 0 to n − 1 do
35     prioritiesOne[i] ← prioritiesOne[i] + prioritiesZero[i];
36 return (offers, prioritiesOne);
```

Sequence clustering

Let us assume now that *netI* has recognized a sequence as belonging to $x11/x21$ device class. Distinguishing between $x11$ and $x21$ device class requires more refinement.

Table 4.7 shows two sequences which are built of the same bargaining set. The orderings of these sequences are based on implementations of policies of $x11$ and $x21$ device classes. The ordering of $x11$ sequence is different, when compared to the $x21$ sequence because of the fact they have been created according to different policy implementations, e.g. like the example ones specified in Tables 4.4 and 4.5.

Table 4.7: Two sequences built of one bargaining set

sequence ordered	
by x11 policy	*by x21 policy*
$\langle 4, 19, 18, 10, 1\rangle$	$\langle 1, 16, 18, 10, 7\rangle$
$\langle 1, 16, 18, 10, 7\rangle$	$\langle 4, 19, 18, 10, 1\rangle$
$\langle 8, 16, 18, 9, 12\rangle$	$\langle 3, 15, 17, 10, 2\rangle$
$\langle 7, 14, 17, 11, 12\rangle$	$\langle 0, 14, 17, 9, 7\rangle$
$\langle 6, 17, 17, 9, 13\rangle$	$\langle 6, 17, 17, 9, 13\rangle$
$\langle 4, 18, 17, 8, 2\rangle$	$\langle 8, 16, 18, 9, 12\rangle$
$\langle 3, 15, 17, 10, 2\rangle$	$\langle 7, 14, 17, 11, 12\rangle$
$\langle 0, 14, 17, 9, 7\rangle$	$\langle 4, 18, 17, 8, 2\rangle$

The difference between sequences can be seen in the first element: a sequence from $x11$ starts with offer $\langle 4, 19, 18, 10, 1\rangle$, whereas the one from $x11$ starts with offer $\langle 1, 16, 18, 10, 7\rangle$. Thus, the sequences can be recognized by observing how consecutive offers in a sequence are submitted. Therefore the only thing one needs to know is the ordering of the sequences.

So, if the document cannot discriminate between $x11$ and $x21$ device class before the negotiation process starts (using the occurrence vector learning described above), it should wait for submitting an offer by the device. Table 4.7 shows that if the device belongs to the $x11$ device class, it would start the negotiation by submitting offer $o_2 = \langle 4, 19, 18, 10, 1\rangle$; however if the device belongs to the $x21$ device class, it would start the negotiation by submitting offer $o_2 = \langle 1, 16, 18, 10, 7\rangle$. So, to correctly guess the device class, the document has to know the ordering of offers of the given policy implementation of $x11$ and $x21$. To get that knowledge we use a neural network.

Now recall how to predict a sequence using neural network, as described in Section 4.1.4. With that in mind, the problem of distinguishing $x11$ and $x21$ can be resolved as follows:

1. All offers from the bargaining set are converted into pairs using Algorithm 4.2.
2. The pairs are introduced to the inputs of $x11netII$ and $x21netII$.
3. Two obtained sets of relations are converted into two sequences: $x11$ and $x21$, using Algorithm 4.3.

The obtained sequences are the ordering we have been looking for.

Consider the bargaining set built of offers from Table 4.3. The offers are converted into the pairs using Algorithm 4.2. So created collection of pairs becomes the input of $x11netII$ and $x21netII$ networks. After converting the pairs and relations returned by both networks, two sequences are returned; they are identical to sequences presented in Table 4.7.

The device starts the negotiation by submitting its most preferred offer, which is $o_1 = \langle 1, 16, 18, 10, 7 \rangle$. Because the sequence returned by $x21netII$ starts with the same offer (see Table 4.7), the document concludes that its opponent belongs to the $x21$ device class.

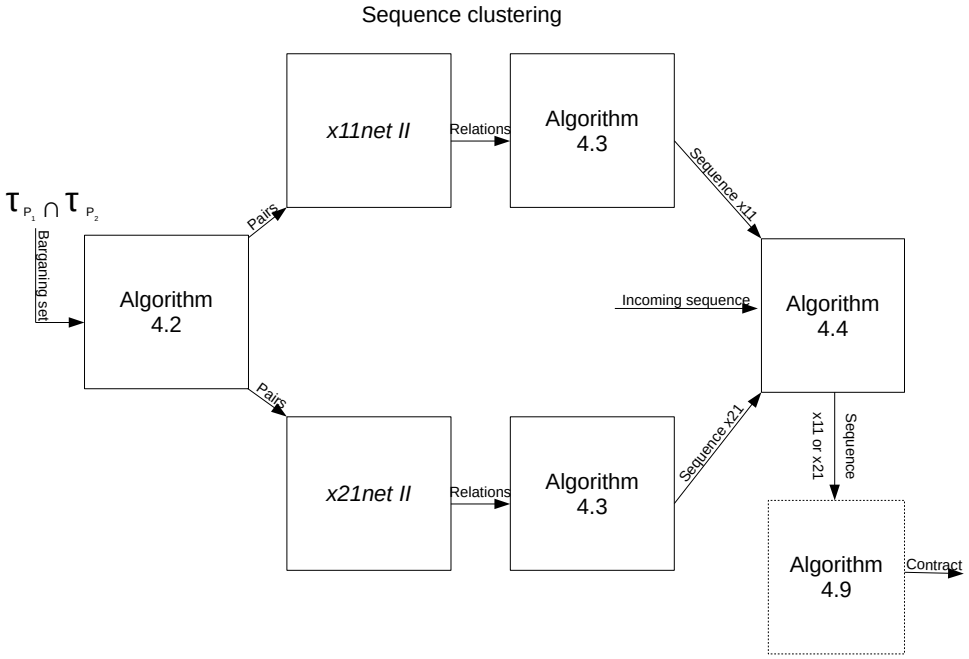


Figure 4.3: Sequence clustering

The recognition process is outlined in Figure 4.3. There are two neural networks. The first one is trained with pairs and relations built of

the sequence belonging to the $x11$ device class, the second one with the ones belonging to $x21$. The input of both networks is the bargaining set shown in Table 4.3 converted into pairs using Algorithm 4.2. Procedure `compareSequences()` which implements Algorithm 4.4 is explained below.

Each neural network returns relations which are converted into sequences by Algorithm 4.3. We call them ς_A and ς_B. The sequences are the inputs to Algorithm 4.4. The algorithm has yet another input. This is a sequence formed by offers submitted by the opponent (a device in this case).

So, the incoming sequence ς has to be compared to both sequences returned by the networks. As it was mentioned before, sequence was stored in two data structures: an array of offers and an array of integers which represent priorities. Now consider that sequence is stored in one array indexed in such a way that the offer which has the highest priority has index 1, the offer which has the second highest priority has index 2, and so on.

Algorithm 4.4: compareSequences()

```
1  Data: incoming sequence ς,
2  sequence returned by the 1-st network ς_A,
3  sequence returned by the 2-nd network ς_B
4  Result: recognized device class or −1 if the device was not recognized
5  stage ← 1;
6  while stage < length(ς) do
7      if ς[stage] = ς_A[stage] then
8          if ς[stage] ≠ ς_B[stage] then
9              return (x11,stage);
10         else if ς[stage] = ς_B[stage] then
11             stage++;
12     else if ς[stage] ≠ ς_A[stage] then
13         if ς[stage] = ς_B[stage] then
14             return (x21,stage);
15         else if ς[stage] ≠ ς_B[stage] then
16             return (−1,stage);
17 return (−1, stage);
```

Procedure `compareSequences()` iterates through array ς starting from element with index 1 and ending at the last element of array. The offer indexed by 1 is the highest offer and the offers indexed by the following numbers are the next offers what corresponds with the order of submitting

the offers. So the indexes are consistent with the stages of the game. During each iteration step the following conditions are examined:

1. if the offer belonging to ς equals the offer of sequence ς_A, then:
   (a) if the offer of sequence ς does not equal the offer of sequence ς_B, then the result is device class $x11$ acquired in the stage assigned by the array index,
   (b) otherwise the index is iterated;
2. if the offer belonging to ς does not equal the offer of sequence ς_A, then:
   (a) if the offer of sequence ς equals the offer of sequence ς_B, then the result is device class $x21$ acquired in the stage assigned by the array index,
   (b) otherwise no result is returned in the current stage.

The example below may seem trivial, however it explains well the underlying concept.

Example 5. *Bargaining set presented in Table 4.3 is converted to pairs. The pairs are inputs of two neural networks: one trained by examples from $x11$ device class, the second by examples from $x21$ device class. The outputs are two sets of relations. The first of them creates the sequence returned by $x11$ device class, the second by $x21$. Both of them are presented in Table 4.7. We interpret them in the following way: the first sequence means that if recognized device class was $x11$, then the offers would submit according to $x11$ sequence; the second sequence means that if the recognized device was $x21$, then the offers would submit according to $x21$ sequences. We should recognize which device class the sequence from Table 4.6 belongs to. Let us denote it by ς. Now let us use procedure* `compareSequences()` *to assign to ς an appropriate device class. The first offer of ς is $o_1 = \langle 1, 16, 18, 10, 7\rangle$. It equals the first offer of the sequence ordered by $x21$ policy and does not equal the first offer of the sequence ordered by $x11$ policy what according to p.2a. means that ς belongs to $x21$ device class.*

4.2 Improved bargaining algorithm

In Chapter 1 we have presented the rules of SBG and Algorithm 1.1 to play it. That algorithm assumed that both players utilized the same negotiation algorithm. Now we would like to assume that player P_2 (the document-agent) utilizes AI techniques, while player P_1 (the device-agent) does not.

It is enough to send to procedure `playSBG()` (Algorithm 1.1) yet another procedure as the parameter. Although the procedure does not change, the bargaining process runs now differently: in the first step the bargaining set is created by the intersection of the option trees belonging to both players. Then the algorithm calls the negotiation procedure of the player not using AI and then increments the move counter. The further step is the negotiation procedure of the player which uses machine learning. It passes the name of player's P_1 negotiation procedure to P_2 because the AI negotiation algorithm (Algorithm 4.5) requires it. It controls the conditions of concluding the game and terminates the game if it is necessary.

Let us look at Algorithm 4.5. Its work sums up all machine learning techniques used in the Thesis. In Section 4.1 we have enumerated four related tasks. All of them are called in procedure `submitAIOffer()` which implements Algorithm 4.5.

Algorithm 4.5: submitAIOffer()

```
1  Data: The bargaining set,
2  Received offer,
3  The histories of recent negotiations
4  Result: Sent counter-offer
5  // Guess player's P_1 device class
6  device ← guessDeviceClass();
7  if device ≠ 0 then        /* the device class is recognized */
8  |  // use neural network to predict the opponent's
   |     sequence
9  |  networkResponse ← useTrainedNetwork(C);
10 |  // send offer using the acquired knowledge
11 |  submitInformedOffer(networkResponse);
12 else                          /* not recognized device class */
13 |  // use one of algorithms not using AI
14 |  submitOffer();
```

Let us enumerate the steps of the algorithm:

1. In the first step the procedure tries to guess the device class.
2. If the device class is solved, the incoming sequence is guessed in order to predict the future opponent's moves and find the best contract.
3. If the device class is not guessed, the negotiation procedure which does not use AI is called.

4.2.1 Classification the of device class

In the first step of procedure `submitAIOffer()` (see p.1 of the enumeration on page 101) the algorithm checks which device class the opponents belongs to. The procedure `guessDeviceClass()` which implements Algorithm 4.6 is called to this end.

The procedure runs as follows:

1. In the first step the occurrence vector is created of the bargaining set.
2. Then procedure `resolveDeviceByOccVec()` which implements Algorithm 4.7 is called:
   (a) Nine-element array containing names of device classes is created.
   (b) The occurrence vector is the input of the neural network and the output is an array.
   (c) The index of the maximum element of the output array is found.
   (d) The procedure returns the name of the device class which is the name indicated by the found index in the array containing the names of the device classes.
3. If any device class other than $x11/x21$ is found, then `guessDeviceClass()` can return the appropriate device class.
4. If the found device class is $x11/x21$, then procedure `compareSequences()` which implements Algorithm 4.4 is called. The procedure has been already described on page 100. It returns $x11$ or $x21$ device class.

The next step of Algorithm 4.5 is to check if the device was recognized. If not, one of algorithms not using AI is called (the last line of the procedure) by calling procedure `submitOffer()`. If the device is recognized, the player may use the neural network in order to predict the incoming sequence, i.e. offers that will be submitted by the opponent. We have described methods of the sequence prediction in Section 4.1.4. In the next Section we will show how to take advantage of the knowledge about the incoming sequence in the aim of speeding up the negotiation.

4.3 Utilization of the acquired knowledge

In Chapter 1 we have promised to show how to make use of the *knowledge* acquired during negotiations. By '*knowledge*' we mean here the order in

Algorithm 4.6: guessDeviceClass()

```
1  Data: the bargaining set C,
2  the sequence,
3  neural networks trained by occurrence vectors,
4  neural networks trained by x11 and x21 sequences
5  Result: the recognized device class
6  occurrenceVector ← createOccurrenceVector(C);
7  // Get the device class using occurrence vectors
8  dev ← resolveDeviceByOccVec(occurrenceVector,netl);
9  // If the device belongs to x11 or x21 device class
10 if dev = x11/x21 then        /* use the sequence clustering */
11     dev ← compareSequences(seq,netResponseX11,netResponseX21);
12 return dev;
```

Algorithm 4.7: resolveDeviceByOccVec()

```
1  Data: occurrence vector,
2  trained neural network
3  Result: the recognized device class
4  // An array containing 9 device classes is created.
5  devs ← [x11/x21,x12,x22,x31,x32,x41,x42,x51,x52];
6  // Obtain the neural network response.
7  netResponseArray ← neuralNetwork(occurrenceVector);
8  // Find index of the highest element.
9  index ← getMaximumIndex(netResponseArray);
10 // Get the name of the device class.
11 return netResponseArray[index];
```

which an opponent will submit its offers - in other words the opponent's sequence predicted by the player processing that knowledge.

So let us go back to Algorithm 4.5. In the case of the recognizing of the device class procedure `useTrainedNetwork()` is called. The procedure summarizes the steps described in Section 4.1.4. Thanks to using the procedure we hide all the complexity bound with converting between sequences and relations. Variable `networkResponse` is a sequence created from pairs made from the bargaining set and relations returned by the neural network. It is represented by an array of offers, which are indexed contrary to the utility of the offer: the lower the index the higher the offer.

The variable is then sent to procedure `submitInformedOffer()` which is the implementation of the *Intelligent Algorithm* which will be denoted further by *KAlg* (for its 'knowledge' component). Algorithm 4.8. Algorithm 4.8 makes the following assumptions:

1. Procedure `submitInformedOffer()` which implements the algorithm is used by P_2 (the document agent).

2. Player P_1 (the device) uses procedure `submitOffer()` implementing one of the earlier presented Algorithms 1.2 or 1.4.

3. P_2 knows its opponent's discount factor.

4. Player P_2 does not know exact values of its opponent's utilities, but assumes the minimum and maximum player's P_1 offers are compatible with respective player's P_2 offers.

5. The opponent's utilities are distributed evenly.

Now we can describe the steps of Algorithm 4.8, which is the key algorithm for this Thesis. Thanks to it the knowledge gained in the sequence recognition process can be utilized to speed up the negotiation process.

1. In its first steps the algorithm initializes the values of the parameters sent to it and then it updates the values of the sets of received and submitted offers. Then the value of the incoming offer is updated to the value of the maximum element of the received offers.

2. Structure P_2 is initialized in order to run the simulation of SBG without using AI. Structure P_1 is sent to the algorithm as the parameter.

3. The order of the opponent's offers which has been sent to procedure in variable `networkResponse`, is not enough to predict the result of the game. In order to do it the exact values of the opponent's offers are necessary. It is done by procedure `genOpponentsOffers()`:

Algorithm 4.8: submitInformedOffer()

Data: set of all offers C,
set of received offers C_R,
set of submitted offers C_S,
structure representing players P_1, with sets of sent $P_1.C_S$, received offers $P_1.C_R$,discounting δ_1,
received opponent's offer o (if the function is called the first time the value is empty) ,
current move number k
procedure playSBG()
procedure submitOffer()
Result: The chosen offer. If the opponent's offer is accepted, the incoming opponent's offer is returned, otherwise the counter-offer is returned, which is the highest predicted offer.

```
// here the player takes advantage of the
// knowledge acquired during negotiations
```
$C_N \leftarrow C - (C_R \cup C_S)$;
$C_N \leftarrow C_N - \{o\}$;
$C_R \leftarrow C_R \cup \{o\}$;
$o \leftarrow$ `max(`C_R`)`;
```
// Initialization of structure P_2 in order to use it in a
   simulation
```
$P_2 \leftarrow P_2$.`initialize(`C, C_S, C_R, δ_2`)`;
P_1.offers $\leftarrow$ `genOpponentsOffers(`networkResponse`)`;
$U_{max} \leftarrow 0$;
foreach $o_{current}$ *in* C_N **do**
    `// the simulation of SBG without AI`
    $U_{temp} \leftarrow$ `playSBG(`$o_{current}$,P_1,P_2,$k+1$,`submitOffer)`;
    if $U_{temp} > U_{max}$ **then**
        $U_{max} \leftarrow U_{temp}$;
        $c \leftarrow o_{current}$;
if $U_{max} > U_2(o)$ **then**
    $C_N \leftarrow C_N$ - $\{c\}$;
    return c;
else
    `// Accept opponent's offer`
    $C_N \leftarrow C_N$ - $\{o\}$;
    return o;

(a) according to p.4 of presumptions of Algorithm 4.8 the minimum and the maximum utilities are equal to the minimum and the maximum player's P_2 utilities;

(b) the remaining $n-2$ values (where n is the length of sequence `networkResponse`) are generated by incrementing the minimum value. The incrementing step is calculated as $step = \frac{U_{max}-U_{min}}{n-1}$, where max and min are player's P_2 maximum and minimum utility.

4. Having estimated all necessary player's P_1 data, player P_2 can predict the result of the negotiation. The prediction may be done by Algorithm 1.1. Procedure `playSBG()`, which implements the algorithm, requires also another parameter: the negotiation algorithm. Algorithm 1.2 or Algorithm 1.4 may be used to implement the procedure `submitOffer()`.

5. Procedure `playSBG()` is called for each possible value of the player's P_2 offer. The offer for which procedure `playSBG()` returns the highest value is compared to the opponent's offer. The higher of the two values is returned.

4.3.1 Experiments with the negotiation algorithm using knowledge

In Chapter 1 we have defined two negotiation algorithms. Algorithm 1.4 finds the solution of *SBG* by calculating a Nash equilibrium in its consecutive round. We will refer to this algorithm further as the *Estimated Algorithm* and it will be denoted by (*EAlg*). Algorithm 1.2 finds solution of the game by comparing incoming offers with the best player's offer. We will refer to this algorithm further as the *Simple Algorithm* for exchanging offers (*SAlg*).

We have generated 6400 bargaining sets; all of them have been included in the attached CD and its content has been listed in Appendix A. The bargaining sets were generated in series consisting of ten of them and they have been used to test Algorithm 4.8 (*KAlg*) for all possible classes of documents and devices. Tables 4.8 – 4.11 include both players' utilities. We show only the values of utilities, because from the point of view of *KAlg* only their values matter.

KAlg assumes opponent to use either Algorithm 1.4 or 1.2 introduced before. In other words, the document agent takes advantage of machine learning when negotiating with devices, whereas devices do not. This assumption is rational, as a single device may negotiate with many documents

at the same time, use more sophisticated resource management techniques than neural networks, and have considerably less training data to learn about documents, that dynamically change their preferences.

KAlg is asymmetrical with regard to the information about the player's opponent. The document agent player knows its opponent's sequence of offers thanks to the knowledge obtained from the learning process.

We assume that *KAlg* is used by P_2. In the experiments values from Tables 4.8 – 4.11 discount factors of both players were set to 0.8.

The main difference between *KAlg*, *SAlg* and *EAlg* is that (*KAlg*) tends to submit offers from a set of offers which can be accepted by its opponent rather than the highest offers.

The bargaining sets used in the experiments to evaluate *KAlg*, in contrast to *SAlg* and *EAlg* were selected from the total of 6400 bargaining sets mentioned before. Utilities of their elements by each player are listed in Tables 4.8 – 4.11.

Table 4.8: Utilities of offers in bargaining set C_1

Offer	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
U_1	0.95	0.975	0.30	0.25	0.70	0.213	0.55	0.925
U_2	0.65	0.363	0.175	0.763	0.338	0.488	0.688	0.413
Offer	o_9	o_{10}	o_{11}	o_{12}	o_{13}	o_{14}	o_{15}	o_{16}
U_1	0.80	0.50	0.425	0.625	0.663	0.888	0.375	0.238
U_2	0.70	0.438	1.00	0.875	0.45	0.15	0.988	0.425

Table 4.9: Utilities of offers in bargaining set C_2

Offer	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
U_1	0.200	0.800	0.900	0.575	0.225	0.998	0.400	0.363
U_2	0.788	0.138	0.175	0.688	0.625	0.563	0.713	0.963

Table 4.10: Utilities of offers in bargaining set C_3

Offer	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
U_1	0.388	0.725	0.600	0.463	0.188	0.763	0.400	0.613
U_2	0.175	0.200	0.213	0.800	0.913	0.688	0.238	0.700
Offer	o_9	o_{10}	o_{11}	o_{12}	o_{13}	o_{14}	o_{15}	o_{16}
U_1	0.700	0.213	0.225	0.538	0.488	0.938	0.688	0.638
U_2	0.975	0.225	0.613	0.375	0.600	0.663	0.538	0.638

For all bargaining sets,$C_1, \ldots, C_4$ the utilities of both players were sorted in the reverse order, i.e. the exchange of offers could last longer. Tables 4.12

Table 4.11: Utilities of offers in bargaining set C_4

Offer	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}
U_1	0.350	0.900	0.725	0.600	0.288	0.300	0.188	0.850	0.675	0.563
U_2	0.400	0.275	0.663	0.963	0.988	0.338	0.450	0.188	0.975	0.438

and 4.13 show results of negotiations using various algorithms: *EAlg* has chosen offer o_1 in move α_1^6, *SAlg* has chosen offer o_{12} also in move α_1^6, whereas *KAlg* has chosen offer o_9 in move α_1^2. Thanks to that the contract reached by *KAlg* was the highest one (because in each player's move utilities are discounted, thus the contract agreed in the earlier stage is likely to have the higher payoff). Note that *KAlg* almost always gave the best payoffs except for bargaining set C_3, where reached the same payoff as *SAlg*. It is interesting to see that for bargaining set C_3 the *EAlg* algorithm got the agreement already in the second move, i.e. faster than the other two. However, *KAlg* works to optimize the payoff not the move number. Since, as we can see in Table 4.12, player P_2 receives the higher payoff in move α_1^2 than in move α_2^1, it chooses rationally to continue the game. There is a similar situation in the case of bargaining set C_3. Player P_2 preferred continuing the game rather than accepting the offer in move α_1^6, since it knew that the next move would bring it the higher payoff.

The tests were intended to check if data from the learning process are able to improve the bargaining process. Results of the negotiation processes between P_1 and P_2 concerning bargaining sets $C_1, \ldots, C_4$, using *EAlg*, *SAlg* and *KAlg* are compared in Tables 4.12 and 4.13. In the former table P_1 used *EAlg*, while P_2 used *KAlg*. In the latter table P_1 used of *EAlg* while P_2 used *KAlg*.

Both tables indicate the number of moves after which the agreement has been reached and the contract negotiated.

Table 4.12: Results of *EAlg* vs. *KAlg*

Bargaining set	C_1				C_2			
Player	Π_1	Π_2	*move*	*offer*	Π_1	Π_2	*move*	*offer*
EAlg	0.33	0.27	α_1^6	o_1	0.14	0.27	α_1^6	o_7
KAlg	0.44	0.56	α_1^2	o_9	0.12	0.39	α_2^7	o_8
Bargaining set	C_1				C_2			
Player	Π_1	Π_2	*move*	*offer*	Π_1	Π_2	*move*	*offer*
EAlg	0.94	0.53	α_2^1	o_{14}	0.33	0.50	α_1^4	o_9
KAlg	0.49	0.62	α_1^2	o_9	0.47	0.62	α_1^2	o_9

From the document agent's (player P_2) point of view the improvements are meaningful. If P_2 used *KAlg* (i.e. took advantage of machine learning) the

contracts were reached twice faster and payoffs were twice higher than in the case machine learning was not used. In one case the agreement was obtained four times faster than when the machine learning was not used. Generally, the more moves in the game variant without using machine learning, the greater improvement in the case of taking advantage of the machine learning techniques as the number of steps to negotiate a contract using AI tends to be constant and small. We will provide detailed characteristics for this observation in Chapter 5.

Table 4.13: Results of *SAlg* vs. *KAlg*

Bargaining set	C_1				C_2			
Player	Π_1	Π_2	*move*	*offer*	Π_1	Π_2	*move*	*offer*
SAlg	0.21	0.36	α_1^6	o_{12}	0.20	0.28	α_2^7	o_4
KAlg	0.56	0.45	α_1^2	o_2	0.99	0.45	α_2^1	o_6
Bargaining set	C_1				C_2			
Player	Π_1	Π_2	*move*	*offer*	Π_1	Π_2	*move*	*offer*
SAlg	0.49	0.62	α_1^2	o_9	0.33	0.50	α_1^4	o_9
KAlg	0.49	0.62	α_1^2	o_9	0.47	0.62	α_1^2	o_9

Results of the experiments reported in Tables 4.12 and 4.13 clearly indicate that knowledge on the ordering of offers, learned from negotiation histories, as explained in p. 4.1.1 and 4.1.4, can significantly reduce the number of steps required to reach the agreement as well as increase the contract value.

In Chapter 5 we will examine what is the minimum number of training data to make a learning process effective, what could be the minimum length of the sequence, what is the impact of errors in learning on the agreed contracts, and how many sequences are necessary to train the document agent on occurrence vectors.

4.3.2 Estimating the opponent's discount factor

In this chapter we have defined Algorithm 4.8, which takes advantage of the acquired (learned) knowledge. Algorithms 1.4 and 4.8 have one common feature, namely they both require a knowledge about the opponent's discount factor. It should be explained that estimation of discount factors is not the primary goal of this Thesis as we would rather concentrate on predicting sequences. But it affects results of negotiation processes and cannot be neglected at all. We will return to this issue further in this Section.

If players have similar preferences they can reach their agreement faster, and the value of the discount factor does not matter so much, especially when

they are equal. However, if the preferences of players differs significantly, the negotiation process will last longer. Then the value of the discount factor should matter. That is why we have selected to the experiments such option trees in which players' preferences are set in the reverse order one to another. Then tuning of the value of the discount factor is the main element that can speed up or slow down the negotiation process. Consider Figure 4.4, which shows how discount factor can affect the move number in which the agreement is obtained.

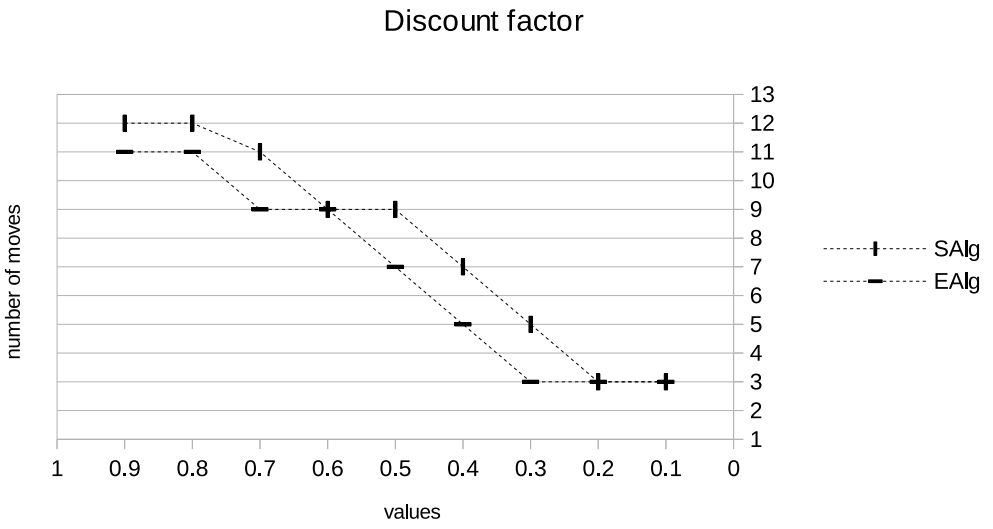


Figure 4.4: Number of moves and discount factors

In the experiment player's P_2 (the document) parameters were not changed. Its discount factor was $\delta_2 = 0.8$. The value of its opponent's discount factor was changed, starting from 0.9 to 0.1. Values of discount factors are displayed on the X axis. The bargaining set consisted of 12 elements. Numbers of moves are displayed on the Y axis. The first move is α_1^0. The last possible move is α_1^{12}. In the case of *EAlg* it was assumed that each player was able to guess correctly its opponent's discount factor.

Based on the example we may propose a general method for estimating the opponent's discount factor by the player not knowing it beforehand. We have to find first in the recorded history all the games in which the document agent (player P_1) took part. Utilities of players that took part in the experiment, were ordered in the opposite order one to another. We assume that if the recorded history of player's P_1 games is sufficiently long, there is at least one recorded sequence for which respective utilities of players were ordered

reversely. It must be the longest recorded sequence. Then it is enough to compare the sequence with results of the experiment presented in Figure 4.4 or another similar one. In the experiment the negotiation between players that used *SAlg* ended in the 9-th move (move α_1^8) if player's P_1 discount factor was 0.5 or 0.6. The size of the bargaining set was 12.

Now assume that the size of the bargaining set is n and the bargaining process has ended in the move number $\frac{3}{4} \cdot n$. Then, assuming that player's P_2 discount factor is 0.8 (like player's P_2 discount factor in the experiment), it may be estimated that player's P_1 discount factor is in between 0.5 and 0.6.

Figure 4.4 shows also that *EAlg* can reach the contracts faster than *SAlg*. However, in the experiment we assumed that players assessed the opponents' discount factor correctly. If the guess was wrong, *SAlg* could reach better results.

Table 4.14: Discount factor in relation to Λ_{SAlg} and Λ_{EAlg}

Discount factor	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Λ_{SAlg}	0.92	0.92	0.85	0.69	0.69	0.54	0.38	0.23	0.23
Λ_{EAlg}	0.85	0.85	0.69	0.69	0.54	0.38	0.23	0.23	0.23

The value of the opponent's discount factor was passed to `submitInformedOffer()` as a parameter. Let us assume that before starting the bargaining process we have run another procedure, `discountFactorEstimation()`, which was utilized to retrieve the value of the discount factor from the history of the games. Before presenting it consider data presented in Table 4.14. It presents the result of the same experiment as shown by the diagram in Figure 4.4. However, now discount factors are bound with function $\Lambda = (k+1)/(|C|+1)$ (where k is the move number in α_i^k) which is the relation of the move number in which the agreement was reached to the maximum number of moves in the given negotiation game.

Procedure `discountFactorEstimation()` will be presented in detail in Appendix B; its steps are the following:

1. Run as many negotiation experiments as possible on generated data for various values of discount factor. The experimental data should be stored in the map data structure where the value of discount factor can be assigned to the given coefficient Λ_{EAlg}. The data should be collected in that way that for the given player (device) the highest value of Λ_{EAlg} coefficient associated with the given discount factor is selected. In result for each device the dictionary built of discount factor and the associated Λ_{EAlg} coefficient is obtained, similar to the dictionary shown in Table 4.14 (naturally without Λ_{SAlg} coefficient).

Table 4.15: Sizes of bargaining sets

Execution context	*Maximum size*	*Realistic size*
x11	16128	30
x12	129024	40
x21	16128	30
x22	258048	50
x31	6912	20
x32	69120	30
x41	576	12
x42	6336	20
x51	432	12
x52	864	12

2. We analyze the negotiation history for each device looking for the highest Λ. In result for each device the value of Λ is obtained.

3. We examine Λ_{EAlg} coefficients obtained in step 2 and check the discount factor in the appropriate dictionary entry obtained in step 1.

In the next Chapter we will examine tests based on much bigger bargaining sets as in examples presented in this Chapter. Based on the experiments with bargaining sets of various sizes, their realistic sizes were determined, i.e. sufficient to observe improvement in the number of moves required to reach the contract when using intelligent bargaining compared to simple or estimated bargaining algorithm. They are summarized in Table 4.15.

Chapter 5

Protocol validation and evaluation

In Chapter 4 we have presented the methods enabling documents to negotiate with devices on the technical parameters of their execution, and the machine learning methods which can help to speed up the negotiation process. Below we present various characteristics that allow us to assess techniques proposed in the Thesis and determine minimal requirements for training data.

5.1 Test plan

The goal of testing is evaluation of negotiation methods and learning approaches and especially an impact of learning on the negotiation capacity.

5.1.1 The subject of testing

Evaluation of the following learning approaches has been assumed:

1. occurrence vector clustering, i.e. distinguishing device classes based on the appearance of specific symbols in offers,

2. sequence clustering, i.e. distinguishing device classes based on ordering of sequences of offers,

3. sequence prediction, i.e. guessing of ordering of sequences of the opponent's offers,

4. utilization of learning in negotiation, i.e. contract prediction.

Sequence clustering precedes sequence prediction. However, since the clustering approach is based on sequence prediction methods, sequence prediction methods has to be examined before sequence clustering. If we want to refer further in this Thesis to both methods related to sequences, i.e. sequence clustering as well as sequence prediction, we will term either one *sequence recognition*.

Tests exclusions

The following simplifications have been considered:

- *Algorithm 1.2* – the Thesis proposes three negotiation algorithms that do not use *AI*but only two two of them have been chosen for experiments, namely *SAlg* (Algorithm 1.2) and *EAlg* (Algorithm 1.4). Since Algorithm 1.2 has been introduced for its simplicity (in particular it does not require knowledge about the opponent's discount factor), we have decided to present only the comparison between *EAlg* and *KAlg* (Algorithm 4.8 that uses *AI*), instead of comparing *SAlg* to *KAlg*, since the latter would be very similar to the former. However, the experiments will show the comparison between negotiation agents that can take advantage of *AI* and agents not utilizing *AI*.

- *Intersection of option trees* – since players are negotiating over the same set of offers, their respective option trees must contain exactly the same offers; the only difference is that both trees are sorted individually by each player.We call such trees *compatible*, i.e. intersection of compatible trees have the same set of elements as either one of them. Therefore, in the tests, instead of generating two option trees and obtaining the result of their intersection we generated the bargaining sets at once.

- *Discount factors* – in Chapter 4 we have presented a simple example in Figure 4.4 of discovering the value of the discount factor of the device. Moreover, the example in Section 4.3.2 indicates that the value of the opponent's discount factor may be discovered without using *AI*. However, the primary goal of the tests is verifying whether AI techniques can speed up reaching agreements by recognizing sequences of offers. That is why we skip investigating that issue further in the Thesis. Nevertheless, estimating of the opponent's discount factor shall be a subject of the further research.

5.1.2 The field of testing

Testing experiments concentrated on:

- effectiveness of learning, measured by the *fitness level*, measured as the ratio of correctly guessed cases to all examined cases taken into consideration.

- effectiveness of negotiation, measured by the *payoff* got by negotiating parties (see the definition of payoff function in Section 1.3.1 In particular, we compare payoffs reached using algorithms that do not take advantage of *AI* with payoffs obtained with the help of *AI*. We also compare them with the payoffs that could be calculated directly from Equation 1.7, when both U_i and U_{-i} are known. This is the ideal case, which we call *fair*.

- costs, measured by the time required to train document agents and the step number in which the solution to Equation 1.7 has been obtained.

5.1.3 The testing methods

Testing to measure the above metrics adopted the quantitative approach, tailored to the specificity of the tested objects, listed in p. 5.1.1. For the training phase a number of bargaining sets were generated. For the testing phase a similar number of bargaining sets were also generated. Both groups were generated automatically as uniformly distributed variations of options for each respective device policies defined in Chapter 3, thus a single element of each such group could be considered a statistically valid representative of the group.

- *Occurrence vector clustering*; 200 training bargaining sets were generated, converted into occurrence vectors (Algorithm 4.1), and introduced to the input of the neural network. When the training of the network was completed, the quality of training was examined using test data. Results obtained from the network output (which is the number assigned to the device class) had to be compared with the device class the given bargaining set was assigned to. The initial number of training examples was then gradually decreased and the training processes were repeated for the new training data. If all test results matched, we received 100% fitness.

- *Sequence prediction*; despite the fact of using the same training data source as in the occurrence vector clustering, the number of sequences used to train the network to predict sequences was 10 times less. The reason was that the examples belonging to each device class had to be trained separately, because each device class required to be trained

by the dedicated neural network. The training data, apart from the bargaining sets, consisted of the file with stored priorities of offers (see Section 4.1.4). The priorities determine the order of submitting offers. The first step of preparing data to training is the creation of sequence from the bargaining set and the priorities. Next the sequence has to be converted into pairs and relations using Algorithm 4.2, and then trained according to the description in Section 4.1.4. Nevertheless, contrary to examples shown in Chapter 4, the network was trained with sequences of various length. Similarly to the occurrence vector training, the initial number of training examples was gradually decreased and the training was repeated using new training data. The output of the neural network was an array of relations. In order to obtain the hits rate of the trained network the testing sequence had to be examined. The testing sequence first had to be converted into the testing pairs and the testing relations. The testing pairs were input of the trained network. The received output array compared to the testing relations was the expected *hits rate*.

- *Sequence clustering*; only $x11$ and $x21$ device classes had to be examined, since they were not distinguishable based exclusively on their bargaining sets. The hits rate was measured in the same way as in the sequence prediction. Additionally, Algorithm 4.4 checked if, and in which move, the proper device class could be identified.

- Learning utilization in negotiation; negotiation methods that use and do not use AI were compared by comparing the payoffs. Algorithms 1.4 and 4.8 were executed using the all generated test data. Next, the results were compared to results of Equation 1.7. In general, examining of the learning utilization relies on running the sequence prediction procedures and then checking how the reached fitness level affects the negotiation results. The steps were the following:

  - The test sequences included in the file with offers from a bargaining set and in the ordering file were first converted into the relational form. The set of pairs and associated with them relations was created (Algorithm 4.2).
  - The pairs were introduced to the input of the network. The set of outputs was created.
  - The outputs were compared with the relations (tasks described in the last two items are executed by the following procedures, listed in Chapter 4).

– Next, Algorithm 4.3 was used to convert the pairs and relations obtained from the output of the network to receive a sequence, which was used in Algorithm 4.8. Because the network was trained using a different number of training examples, a number of the training sequences was created, which was equal to the number of the training sets.

– In order to see whether learning can improve the negotiation process we compare payoffs that document agents got when using Algorithm 4.8, which utilizes the knowledge gained during training to payoffs they could get when negotiating without any prior training (Algorithm 1.4).

5.1.4 The testing resources

Generated bargaining sets

The training examples have been generated respectively for each device, in series consisting of ten n-element sets of offers, where n is the number of offers in the set. Each series had a property that all sequences of the n-th element series were disjoint, i.e. any two sequences in the series did not include the same offer from the bargaining set. For the testing purposes the same number of examples has been generated. Then reasonable subset of them have been chosen as their statistical representative.

Training data were stored in two files: one included encoded values of offers, i.e. bargaining sets, while another included priorities containing the information about ordering of offers in a given bargaining set sorted with regard to their utilities. Based on that information it was known how each possible sequence was built, i.e. it was known in what order a given device would be submitting its offers.

In Chapter 4, when we have presented the sequence based learning, we have assumed that the length of the respective sequences are equal to the size of the corresponding bargaining sets. However, if we look at the rules of *SBG* (see Definition 6), the maximum number of moves in *SBG* is $\lceil n_{max}/2 \rceil$, which is also the maximum length of any sequence (see Definition 11). That was why we examined the behavior of the network for shorter sequences in the training processes. So apart from examining sequences which had the same sizes as the bargaining sets, we examined sequences which sizes were equal a half of the bargaining set, or even smaller.

In sequence clustering as well as sequence prediction (see p.2 and 3 on page 113) the size of the bargaining set, as well as the length of the sequence

should matter. So both of them have to be presented in diagrams that illustrate the sequence prediction.

Neural networks used

Neural networks have been utilized for two specific tasks: the clustering of occurrence vector and the sequence recognition. For each task two types of networks have been adopted with one, and respectively, two hidden layers.

Networks $netI1$ and $netI2$ used for occurrence vector clustering are shown in Figures 5.1 and 5.2. Each one has 82 inputs. Network $netI1$, has one hidden layer consisting of 20 neurons with the *hyperbolic tangent sigmoid* transfer function (*tansig*). The 9-neuron output layer also uses the *tansig* transfer function. Network $netI2$ consists of two hidden layers. The first, 20-neuron hidden layer, has the *tansig* transfer function, so does the second 10 neuron hidden layer. The 9-neuron output layer has the linear (*purelin*) transfer function.

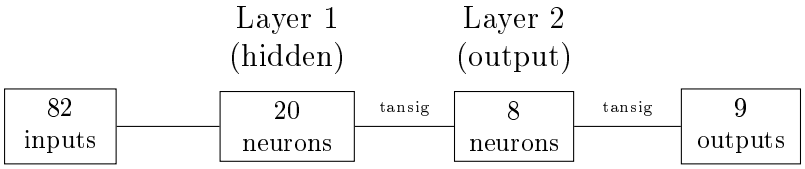


Figure 5.1: Structure of the $netI1$ network

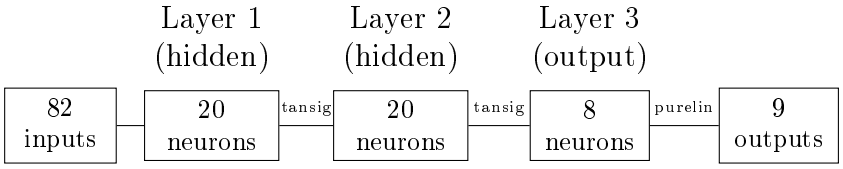


Figure 5.2: Structure of the $netI2$ network

Networks $netII1$ and $netII2$ used for sequence recognition are shown in Figures 5.3 and 5.4. The first of them ($netII1$) is three layer network, whose first hidden layer consists of 20 neurons and the second one consists of 7 neurons. Both hidden layers have the *tansig* transfer function. The output layer, which consists of one neuron has the linear transfer function (*purelin*). The network is shown in Figure 5.3.

The second one ($netII2$) is the 2-layer network, whose hidden layer consists of 10 neurons. The hidden layer and the 1-neuron output layer also uses the *tansig* transfer function. The network has 174 inputs.

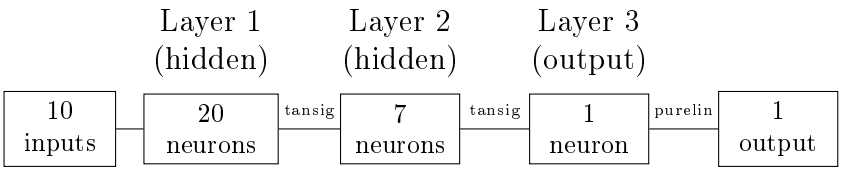


Figure 5.3: Structure of the $netII1$ network

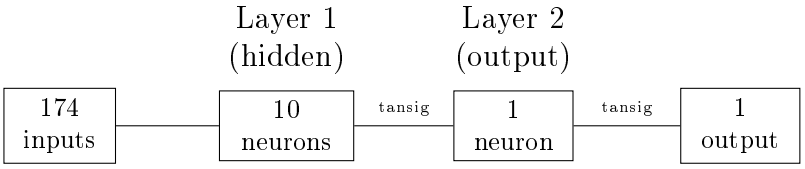


Figure 5.4: Structure of the $netII2$ network

Hardware and software

Further in the chapter several diagrams with results of tests are presented. They illustrate the average time needed to train the networks. In these tests an ordinary laptop with Intel I7 eight-core 1.87 GHz CPU and 16 GB RAM was used. The Matlab 2012b neural network toolbox was used in all the experiments and all major algorithms presented in this Thesis were implemented in Matlab. For completeness, their pseudocode specifications have been included in Appendix B.

5.2 Test design specification

A comprehensive summary of all the tests performed in order to compare and evaluate negotiation algorithms proposed in the Thesis were published in [34]. Below the full account on all experiments performed is given.

5.2.1 Testing scenarios

In the experiments we use the following approach; start training the network with a number of training examples and then gradually decrease their number. Next, the trained networks are tested. The testing data are introduced to the input of the neural network and the output of the network is compared with the expected results which were generated together with the tests.

In the case of the occurrence vector learning the starting number of examples was 2000, and then repeated for 1000, 500, 250, 200 and 100 examples.

In the case of the sequence learning the starting number of examples was 200, and then repeated for 100, 50, 40, 20 and 10 examples.

Furthermore, we assume that the discount factor of each player was 0.8 and both players have discovered that fact before.

Scenario 1 Occurrence vector learning

1. First, each test bargaining set has to be converted into an occurrence vector (Algorithm 4.1).

2. The given occurrence vector has to be examined in which device class it belongs to. The mechanism of choosing the closest class has been described in Chapter 2. It is realized by procedure `strategy_chooserX()`, which source code may be found in the enclosed CD.

3. The given test is passed if the recognized device class (bargaining set) is compatible with the name of the device class included in the filename the bargaining set is stored. The result is the ratio of the number of passed tests to the number of all bargaining sets (the fitness level). It is realized by procedure `strategy_testerX()`. If all tests pass, the fitness level is 100%.

Scenario 2 Sequence clustering

1. The inputs of the sequence clustering are the examined bargaining set and incoming offers that create a sequence $seq0$ (the sequence consisting of offers included in the bargaining set). Moreover, we have two trained networks: $x11netII$ and $x21netII$. The complete scenario 2 is implemented by procedures `relations_train()` or `rel_train_bis()` listed in Appendix B.

2. The input bargaining set and the required ordering of its data, are converted to the relational form (the set of pairs $pset0$ and associated with them relations $rset0$) with Algorithm 4.2. The ordering is stored in file in the test purpose.

3. So created pairs are introduced to the inputs of networks $x11netII$ and $x21netII$.

4. We obtain two sets of pairs and relations associated with them. One set comes from $x11netII$ and another from $x21netII$. We denote them respectively as $pset11$, $rset11$ and $pset21$, $rset21$ (prefix 'p' indicates the set of pairs, prefix 'r' indicates the set of relations).

5. We convert $pset11$, $rset11$ and $pset21$, $rset21$ into sequences $seq11$ and $seq21$, using Algorithm 4.3.

6. Now we have two sequences: $seq11$ which corresponds with the ordering of offers submitted by the device which belongs to $x11$ device class and $seq21$, which corresponds with $x21$ device class. We have only to compare the incoming offers (sequence $seq0$) to both sequences to answer the question, which device class $seq0$ belongs to.

7. We use Algorithm 4.4 to decide if offers from sequence $seq0$ belongs to $x11$ or $x21$ device class. The input parameters of the algorithm are sequences $seq0$, $seq11$ and $seq21$. Its output is the proper sequence (or no solution) and the move number in which the solution has been reached. Knowing the sequence implies knowing the device class.

8. Now we compare $pset0$ and $rset0$ with the adequate pairs and relations, depending on the obtained solution. The hit rate is received, which is the ratio of compatible relations to the number of all relations, i.e. the percent of correct guesses.

9. So, the sequence clustering can give us the following statistics: the number of move in which the solution was obtained in relation to number of training examples and the percentage of correct guesses in relation to number of training examples.

Scenario 3 Sequence prediction

1. The input of the process is a bargaining set. The document agent intends to obtain ordering of the offers in the set in order to create sequence $seqij$ and then compare it with information stored in the ordering file. The ordering file is converted to the relational form and $rset0$ is obtained.

2. The bargaining set is converted into pairs $psetij$.

3. Pairs $psetij$ are introduced to the input of network $xijnetII$. The output of the network is $rsetij$.

4. Comparing $rsetij$ to $rset0$ shows the percent of correct guesses. This result is presented in diagrams in relation to the number of the training examples. The tests end at this point, however to obtain any useful result we have to proceed further.

5. Algorithm 4.3 converts $psetij$ and $rsetij$ into sequence $seqij$ which can be exploited by Algorithm 4.8 in the negotiation process.

Scenario 4 Utilization of learning in negotiation

1. Scenario 3 provided sequences *seqij* and examined the differences between them and sequence *seq*0. In the current scenario we compare the results of negotiation with the device agent whose offers are ordered according to sequence *seqij*. However, the document agent's knowledge about the ordering will differ, depending on the fact, how good its guess was.

2. First the result of Equation 1.7 is calculated. The result is called *fair* because it is calculated from the point of view of someone who knows utilities of both negotiation parties. The result is the primary point of reference, since it is the best result that both players can obtain.

3. We run the negotiation process using Algorithm 1.4. The algorithm does not require any knowledge about the opponents' sequences and therefore the result is our second point of reference. We would reasonably respect that contracts negotiated by the document with the use of AI could yield higher payoffs than for the second point of reference, possibly close to the first point.

4. Sequences provided by Scenario 3 differed depending on the number on training examples used in the experiment and its quality, i.e. the obtained *hits rate* (see the list on page 115). We run the negotiation processes using Algorithm 4.8 using all sequences obtained in Scenario 3.

5. The results are outcomes of the negotiation processes proceeded using sequences created basing on the networks, trained using different number of examples. They are compared to the *fair* results and the results of the negotiation without using *AI*.

5.2.2 Testing criteria

The learning process is applied in two stages: device recognition and contract guessing. Since errors made in the first stage may cause errors in the next one, we have decided to accept only 100% fitness level in occurrence vector clustering and sequence clustering. The remaining methods are assessed in the following way:

- sequence prediction – we will present the relation between number of the training examples and the learning quality (hits rate – the ratio of correctly recognized examples to all examined examples),

- utilization of learning in negotiation – we will show how decreasing of learning quality affects the level of payoffs and compare the obtained results.

5.2.3 Groups of test cases

In general the bigger the bargaining set the better the training material. So we have considered to generate training data for different sizes of the bargaining sets. We divide the data into three groups depending on the size of the bargaining sets that may be generated for policies defined in Chapter 3:

Group 1 Six element bargaining sets as the smallest cases worth to consider. The number was chosen because during our initial experiment it was the border value below which the learning would be unsuccessful. The group was created since we wanted to examine if the results of training could be successful in the case of using bigger number of training examples.

Group 2 The eight element bargaining sets was chosen (based on the same observation as before) that from this size of bargaining sets all learning results should be good.

Group 3 The data collected in the actual target MIND system will rather not have equal sizes. During the tests we identified the sizes of bargaining sets which appeared in the tests most frequently. They are considered realistic and listed in Table 4.15.

5.3 Specification of test cases

For each group listed in the previous subsection 2000 bargaining sets were generated and additional statistically meaningful test bargaining sets were created for testing. The test bargaining sets were generated by the same generating function as the other examples.

The training data and the test data (bargaining sets) were generated in series. One series consisted of ten bargaining sets, each one stored in a separate file. In this way each test could be easily identified by its filename. Bargaining sets consisted of offers used to generate sequences by applying the respective ordering information. So each bargaining set file with offers has one associated file with its orderings.

Table 5.1: Test files name patterns

Training	
Offers	Ordering
traits_mod_0**Xij.txt	classes_mod_0**Xij.txt
Test	
Offers	Ordering
tst_traits_mod_0**Xij.txt	tst_classes_mod_0**Xij.txt

Generated data are stored in files, which are identified by four filename patterns, as shown in Table 5.1. The networks are trained using data containing sets of offers and ordering of these offers, in two separate files named accordingly. The first file specifies offers and the second one their ordering. The test data were stored in files identified by two similar names. The first file specifies offers, while the second one their ordering. The testing examples were stored in the same way as the training examples. The templates of file names are specified in Table 5.1. Wild cards stand for 20 different pairs of digits from '00' to '19'. The filename templates concern all three test groups.

Document types were described in Table 3.12 with regard to policies they may want to follow. For the experiment we have to divide documents into more specific groups, as specified in Figure 5.5.

The static, reactive and proactive policies specified in Table 3.12 characterize the level of *interactivity* that documents may exhibit to the device. Besides that we would like to investigate how their *content* may be accessed by the device, by distinguishing whether the content is *protected* or *open*, and what *load* they may want to put on the execution device, by classifying it as *heavy* or *light*.

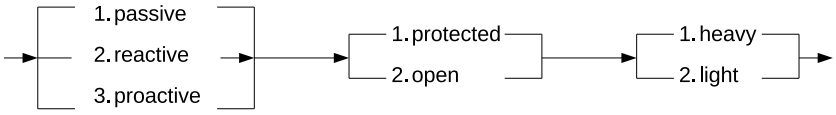


Figure 5.5: Generation scheme for document dictionaries

This detailed classification refines the respective sets of attribute values used in the policy rules listed in Table 3.12 into smaller ones. We call such sets *dictionaries* and list them in Appendix A. Dictionaries for each respective class of documents have been generated using the scheme shown in Figure 5.5.

Files `d11.txt`,...,`d322.txt` with document dictionaries, generated for the experiments, indicated what options should be considered when generating bargaining sets and making option trees compatible. For example `d312.txt` includes options specific to the proactive document with the protected con-

tent and which load is light.

Similarly, dictionaries for the execution devices have been generated as files `X11.txt`,...,`X52.txt`, according to Table 3.6.

5.4 Occurrence vector learning

Figure 5.6 shows results of training $netI2$ (the 3-layer network) with bargaining sets belonging to Group 3. It may be seen that the occurrence vectors made of at least 50 bargaining sets enable the negotiating document to recognize the device with 100% accuracy. When repeating the experiment with $netI1$ (the 2-layer network) the results were of the similar shape, but reaching the top of only 80% of correct hits (we skip the respective chart for brevity). It shall be noted that the time required to train $netI2$, as shown in Figure 5.7 is practically negligible. This observation is quite important, as training was performed using a laptop of a moderate computational power (see specification on page 119). We will come to this issue later in the Chapter.

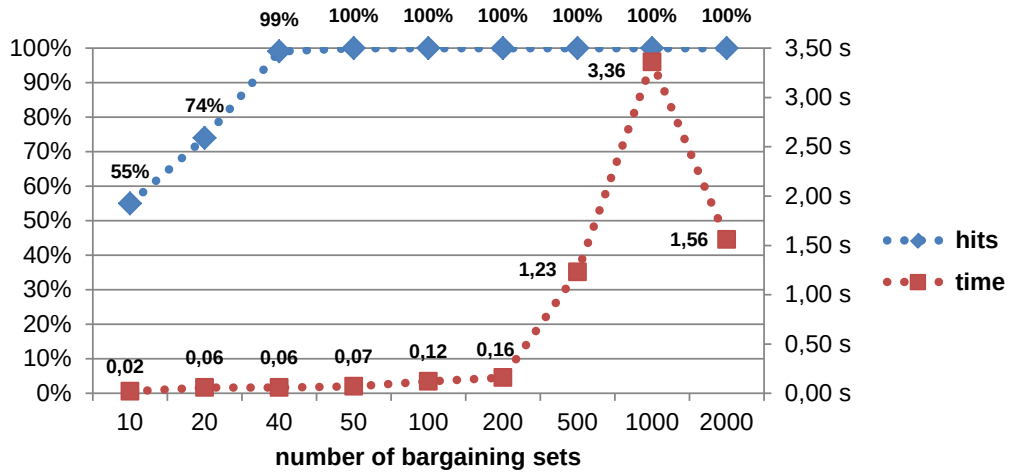


Figure 5.6: Hits and training times for $netI2$ trained with examples of Group 3

Figures 5.7 and 5.8 show results obtained with networks $netI1$ and $netI2$ trained with bargaining sets of Group 2. Network $netI1$ was able to achieve perfect hits when trained with occurrence vectors made of at least 1000 bargaining sets, again in a relatively short time.

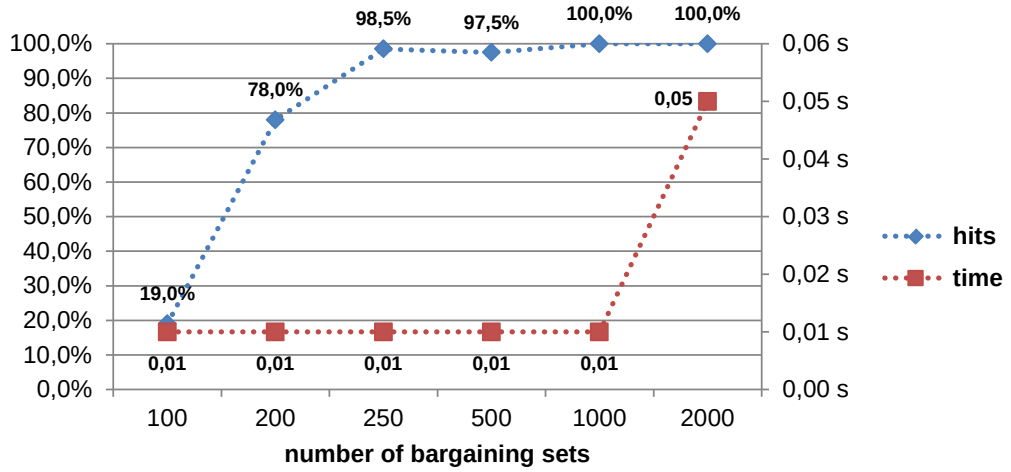


Figure 5.7: Hits and training times for $netI1$ trained with examples of Group 2

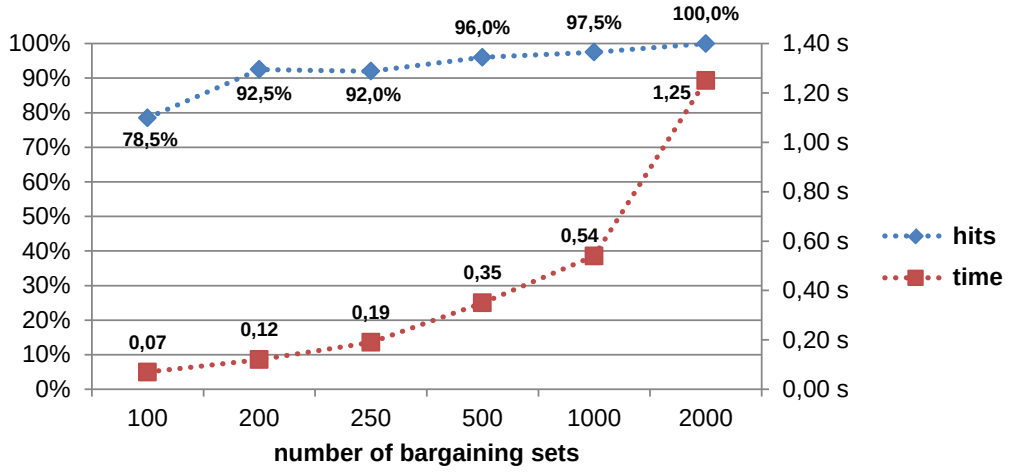


Figure 5.8: Hits and training times for $netI2$ trained with examples of Group 2

In the case of $netI2$ perfect hits have been possible after training the network with occurrence vectors made of 2000 bargaining sets and in a relatively short time as before. The diagram is shown in Figure 5.8.

Figures 5.9 and 5.10 indicate that results of tests with the bargaining

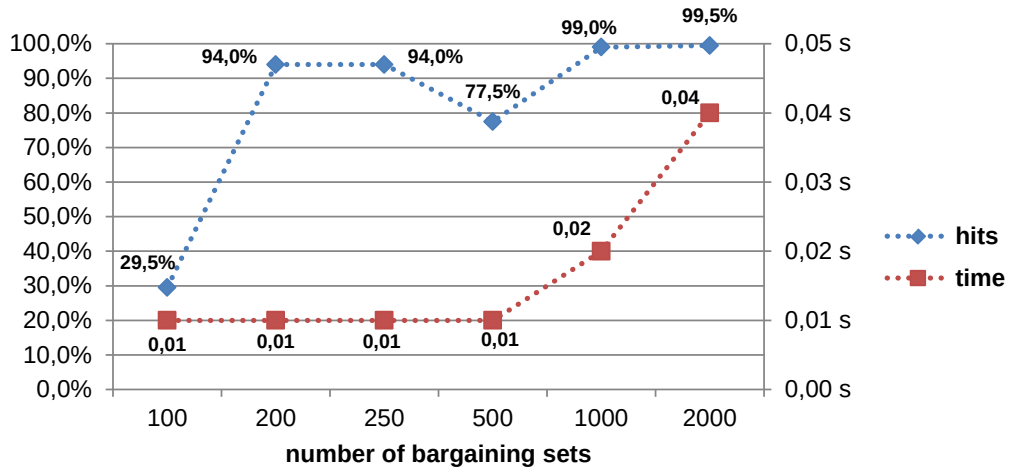


Figure 5.9: Hits and training times for $netI1$ trained with examples of Group 1

sets of Group 1 were not satisfactory, although for both types of networks occurrence vectors made of 2000 bargaining sets were close to the required 100%. It indicates that 6-element bargaining sets are too small to use them in the occurrence vector training. On the other hand increasing of the number of the training examples beyond 2000 seems pointless, so 6-element bargaining have been excluded.

Network $netI2$ gave much better results than network $netI1$ for Group 1, contrary to Group 2 and 3, where the latter worked better (see Figures 5.6 – 5.8).

In Section 5.2.2 we have required fitness level of the trained network to be 100%. This requirement is satisfied by Groups 2 and 3; examples belonging to Group 1 used for training *netI1* and *netI2* – even in the case of 2000 training examples, reached just 99.5% fitness level. On the other hand, examples of Group 3 returned very good results, especially when used to train with *netI2*. The network got 100% fitness level even when trained with only 50 bargaining sets.

Figures 5.6 – 5.10 illustrate the most important result of experiments performed in connection to Scenarios 1 and 2. For brevity we skip presenting detailed results of the remaining experiments related to these scenarios, but provide their synthetic summary in Tables 5.2 and 5.3.

We can see that network $netI1$ is better in the sense that it requires in general less training examples to train the network, as well as requires less

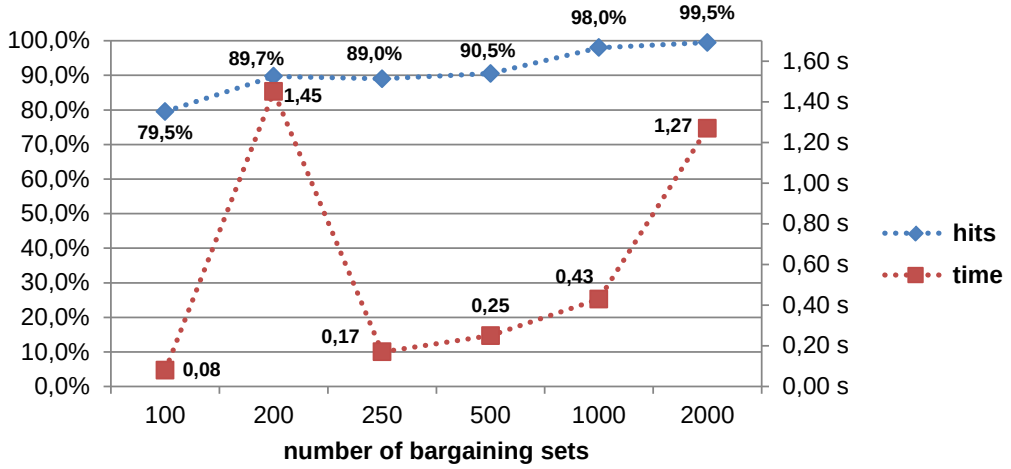


Figure 5.10: Hits and training times for $netI2$ trained with examples of Group 1

Table 5.2: Learning summary

	Minimum number of bargaining sets in the occurrence vectors to reach 100% hits	
Group	$netI1$	$netI2$
1	2000*	2000*
2	1000	2000
3	40	50

* max hits close to 100%

Table 5.3: Time summary

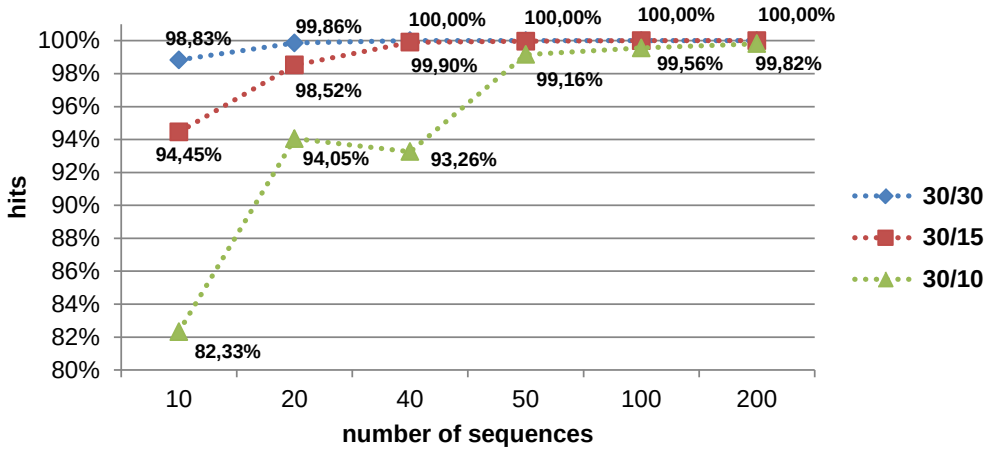
	Maximum time of training to reach 100% hits	
Group	$netI1$	$netI2$
1	$0.04s$*	$1.27s$*
2	$0.05s$	$1.25s$
3	$0.30s$	$3.36s$

* max hits close to 100%

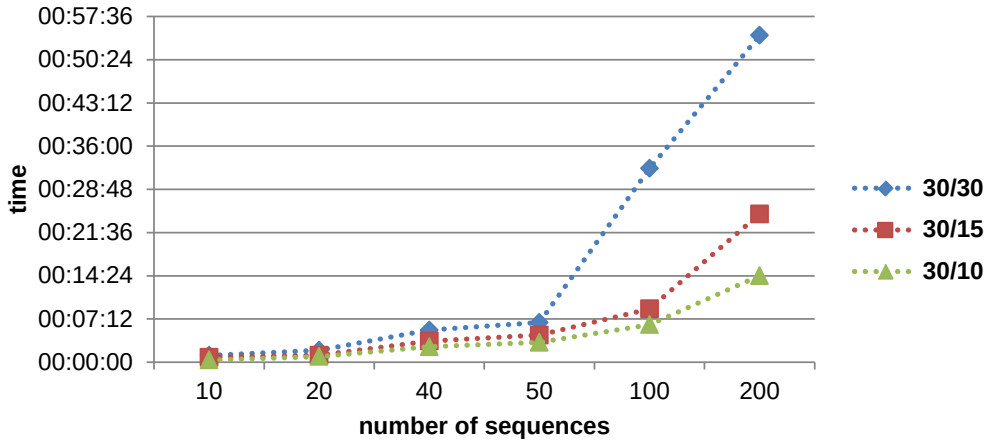
time to complete it.

5.5 Sequence prediction

Logically, sequence clustering precedes sequence prediction. However, since sequence clustering is based on sequence prediction, we need to describe sequence prediction before describing sequence clustering. We will return to the latter in Section 5.7.



(a) Recognition for $x11$ device class (30-element bargaining set)



(b) Learning times for $x11$ device class (30-element bargaining set)

Figure 5.11: Network *netII1* sequence

Now let us assume for a while that the document-agent has recognized the proper device class. In this phase of the experiment the same training data might be used as before. Recall that learning of occurrence vectors covered all training examples at once, and the number of tested bargaining sets was 2000. Now training procedures affect only one device class, because each device class has to be trained separately. Since there are 200 bargaining sets belonging to one device class, ten separate neural networks have to be trained – each one using 200 training examples, or less.

Before presenting diagrams concerning Scenarios 2 – 4 we would like to recall the fact that the length of sequences may differ from the size of the bargaining sets. In the diagrams, which refer to sequence prediction the size of the bargaining set and the length of the sequence are presented as a fraction, in the form X/Y, where X denotes the bargaining set size and Y the sequence length. The values are displayed on the vertical axis. The horizontal axis shows the number of bargaining sets used for training.

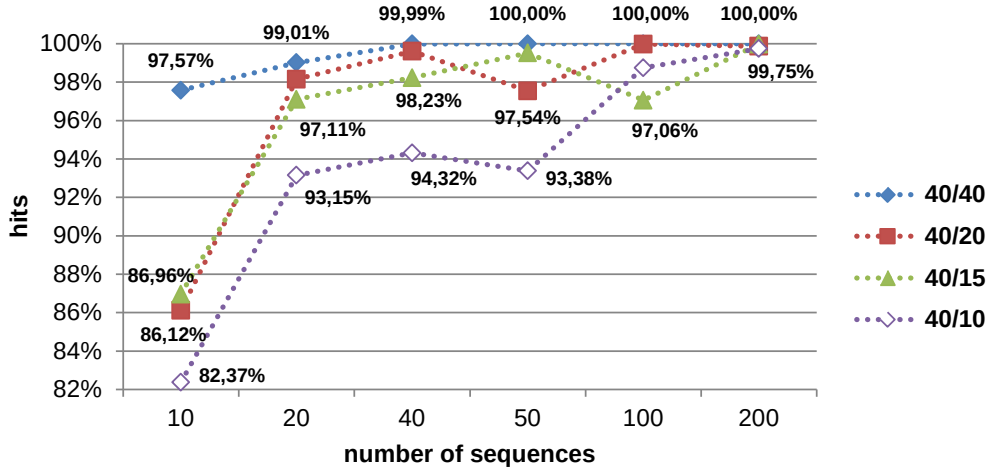
The diagram shown in Figure 5.11(a) presents results of training *netII1* with examples belonging to Group 3, while the times of training sequences are shown in Figure 5.11(b). They are in general longer than in the case of training networks to recognize device classes, but still within a reasonable limit. In particular, sequence learning times for the number of 50 – 200 sequences required to reach perfect or almost perfect hits (see Figure 5.11(a)) could be measured in minutes, which is still a reasonable time given the fact that training was performed on an ordinary laptop (see page 119).

The question what would be the impact of imperfect hits when predicting sequences during negotiating on the contract finally agreed, will be answered in the next section. For the time being it may be said that imperfect hits in Scenario 2 are of lesser importance than in the case of Scenario 1.

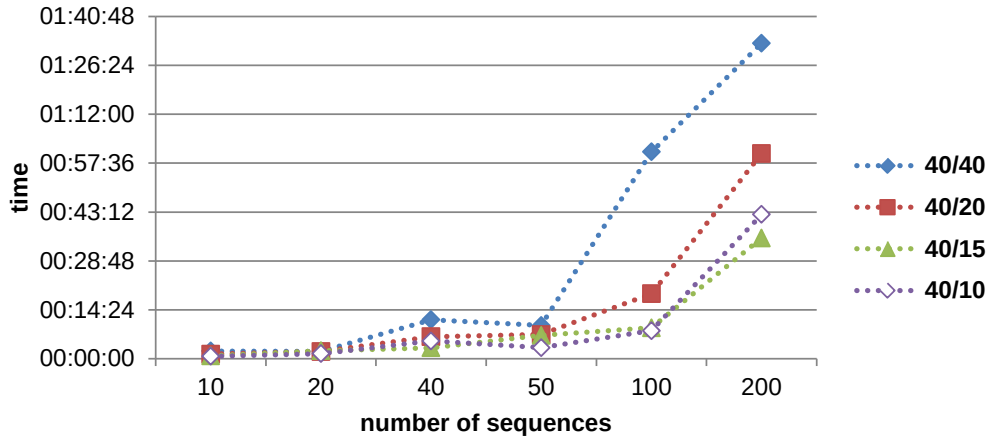
The diagram in Figure 5.12(a) shows the results of training network *netII1* using examples that belong to the $x12$ device class (connected workstation), while Figure 5.13(a) shows the results of training network *netII1* with examples belonging to the $x21$ device class (disconnected laptop).

They illustrate stability of the training method. Note that the network trained with 10 element sequences could reach almost perfect hits only when trained with specific number of training examples. As the length of training sequences grows, quality of the training depends less on the size of the training set. We will come to this issue later in the Chapter.

Figure 5.13(a) shows results of training, which look similar to the previous diagrams, except when training network *net II1* with the ten element sequence (30/10). In spite of training using 200 training examples we can see the worse result than in the case of the other number of examples. The results indicated the need to look for another network which could behave



(a) Recognition for $x12$ device class (40-element bargaining set)

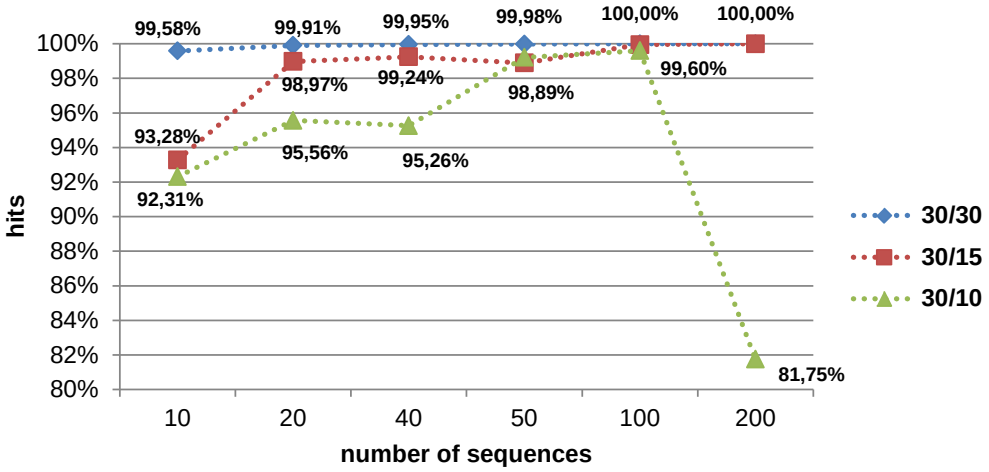


(b) Learning times for $x12$ device class (40-element bargaining set)

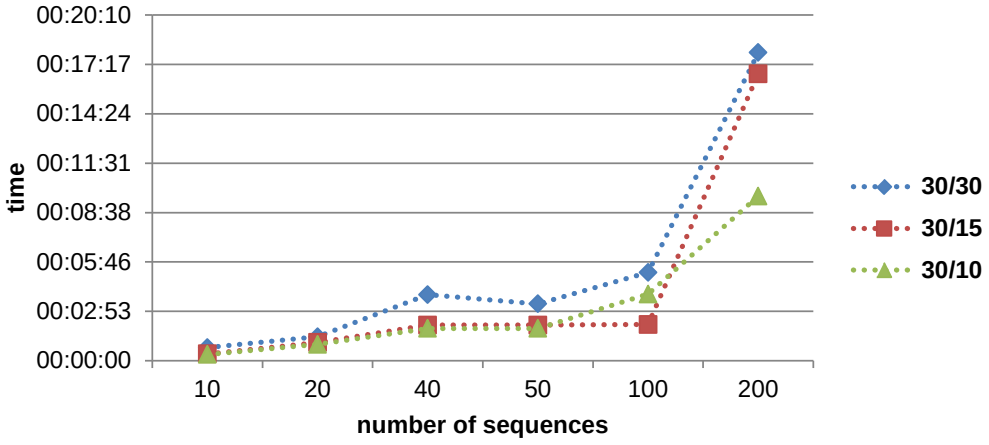
Figure 5.12: Network *netII1* sequence

better in the case of training using shorter sequences. For that we exploited network *netII2*. The times of training network *net II1* are shown in Figure 5.13(b). The times are compatible with the length of the sequence – the longer the sequence the more time of training is necessary.

The diagrams in Figures 5.14(a) and 5.15(a) show results of training using



(a) recognition for $x21$ device class (30-element bargaining set)

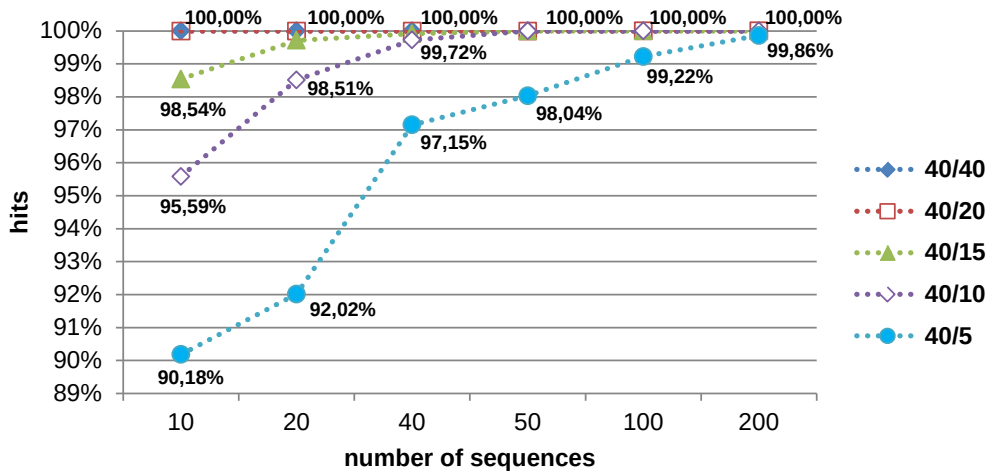


(b) learning times for $x21$ device class (30-element bargaining set)

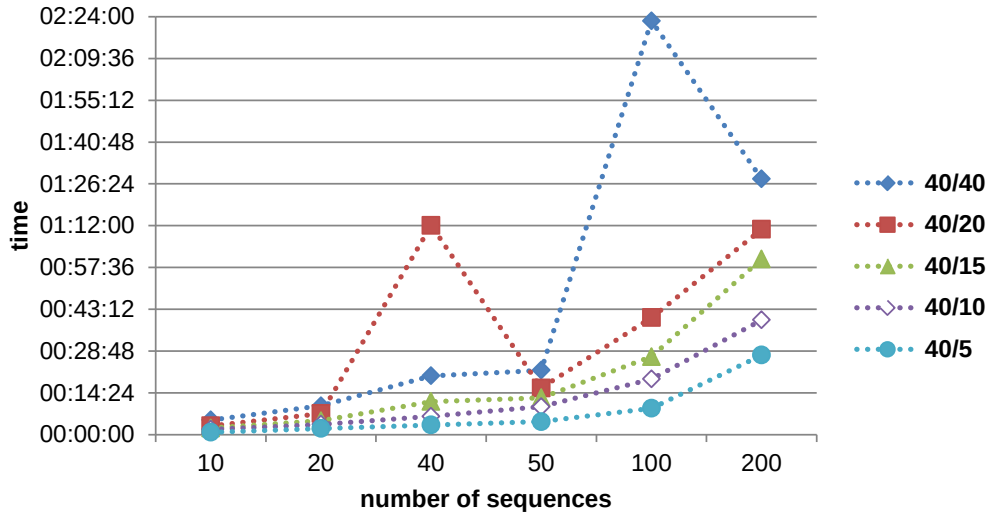
Figure 5.13: Network *netII1* sequence

network *netII2*.

Because we have expected better results of the network, we have decided to examine more sequences. So, the $x12$ device class was examined using 40-, 20-, 15-, 10- and 5-element sequences, while the size of the bargaining set has been 40. The $x22$ device class was examined using 50-, 25-, 20-, 15-, 10- and



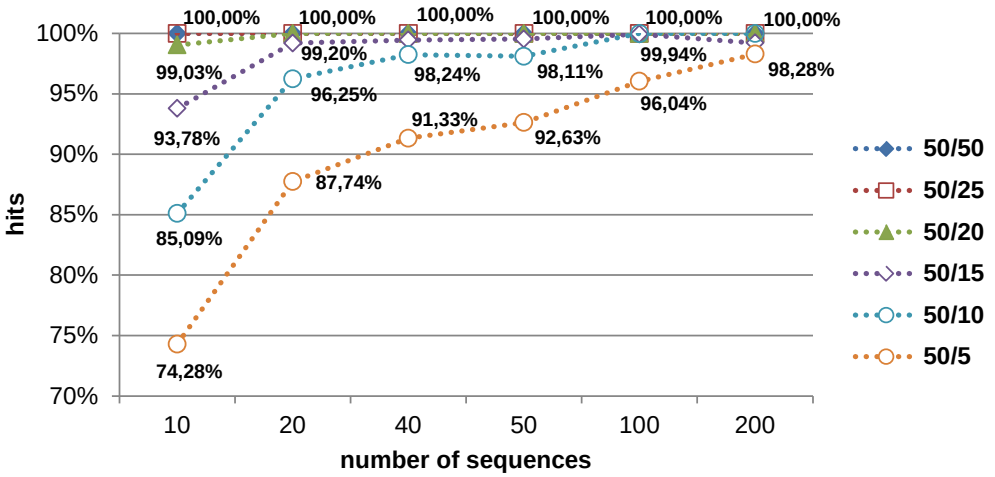
(a) Recognition for $x12$ device class (40-element bargaining set)



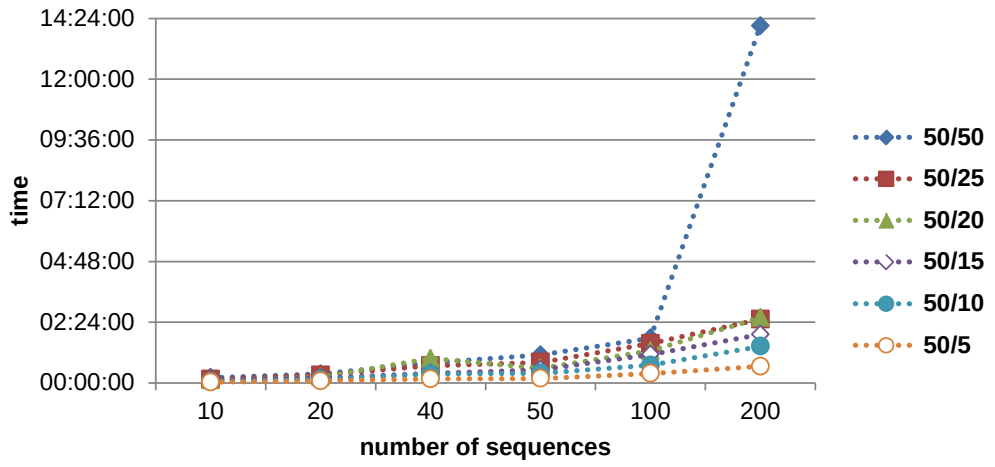
(b) learning times for $x12$ device class (40-element bargaining set)

Figure 5.14: Network *netII2* sequence

5-element sequences, while the size of the bargaining set has been 50. The results are satisfactory: network *netII2* trained using examples belonging to the $x22$ device class obtained 100% result even if it was trained using only ten sequences. The size of the sequence has been 25 whereas the size of the bargaining set has been 50. Moreover, ten 5-element sequences could



(a) Recognition for $x22$ device class (50-element bargaining set)



(b) Learning times for $x22$ device class (50-element bargaining set)

Figure 5.15: Network *netII2* sequence

train network *netII2* reaching 74% fitness level. Similarly the examples from the $x12$ device class could train network *netII2* well. Ten sequences, whose length equals the half of the size of the bargaining sets, which is 40, can train the network to the 100% fitness level. Furthermore, ten 5-element sequences has given 90% result of learning.

When comparing results of training networks *netII1* and *netII2* with examples from Group 3 and the $x12$ device class, it may be seen that sequences 40/40 and 40/20 gave 100% fitness level regardless of the number of the training examples (network *netII2*). Network *netII1* performed worse comparing to Network *netII2*. Times of learning of both networks, presented in Figures 5.12(b) and 5.14(b), have been slightly longer in the case of network *netII2*. Some exceptions may be observed. They relied on the fact that with less training examples the training times were sometimes longer. It could be caused by the fact that network *netII2* had to initialize randomly more weights then network *netII1*. However, Figure 5.14(b) shows that the exceptions occur only in the case of the longest sequences – 40/40 and 40/20. Thus, since in practice the training examples will consist of shorter sequences, the exceptions may have no practical meaning.

When testing with sequences relevant to the bargaining sets of Group 2, results obtained for network *netII2* trained with 4– and 2– element sequences built on the 8-element bargaining sets are shown in Figure 5.16(a).

We can see that if the number of the training examples is 200, 4-element sequences give the perfect hits and 2-element sequences - almost the perfect fitness level.

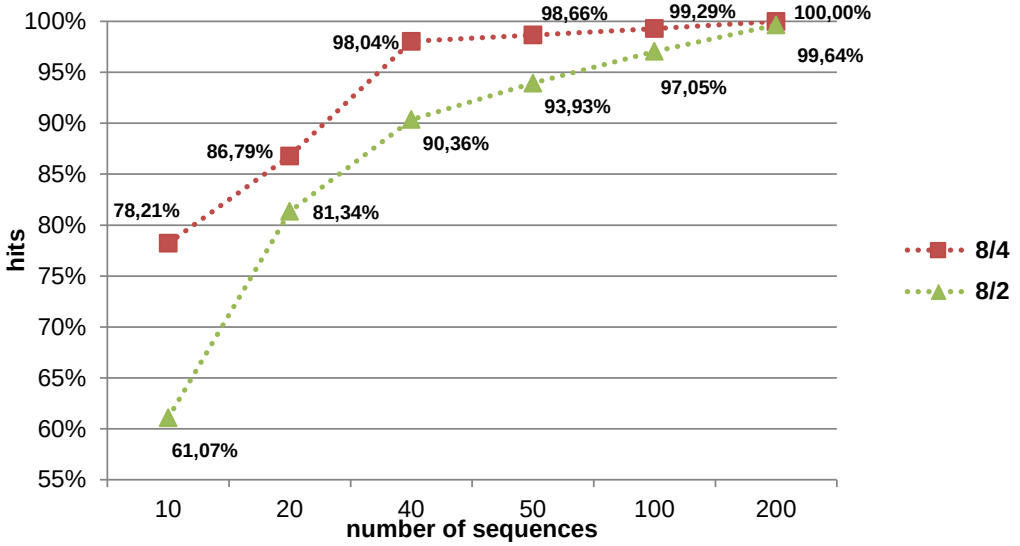
Figure 5.16(b) shows the time needed to train the network, as presented in the previous diagram. It may be seen that it is reasonably low.

Figure 5.17(a) concerns examples from Group 1. It shows that in the case of 3-element sequences built of 6-element bargaining sets even 200 training examples is not enough to obtain 100% results of learning, although they are close to it (99.5%). Figure 5.17(b) shows the time of training of the network, which is also reasonably low.

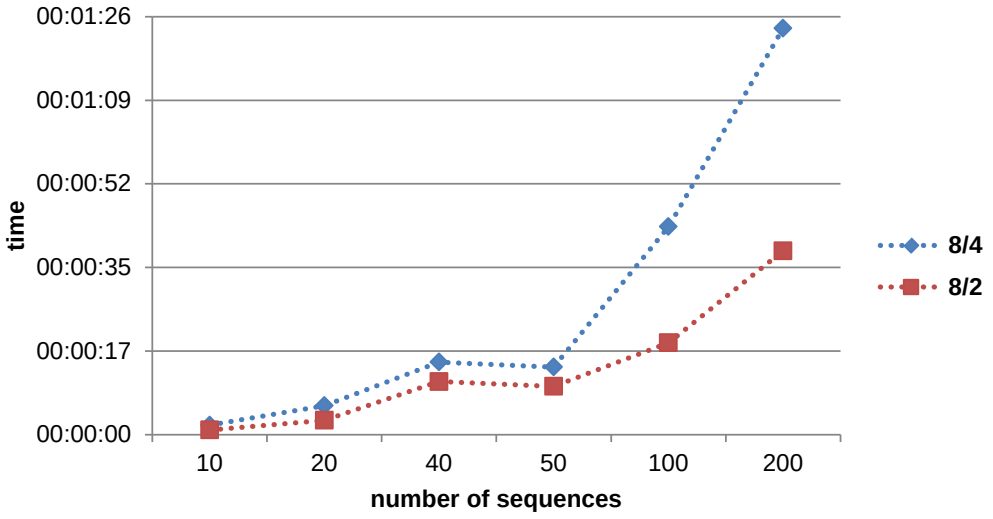
Table 5.4: Perfect hits of $x12$ device class trained with $40/x$ sequences

	Minimum number of bargaining sets in the sequence prediction to reach 100% hits	
Sequence length	*netII1*	*netII2*
40	50	10
20	–	10
15	–	50
10	–	50

To conclude this section we will compare *netII1* and *netII2* neural networks which were used in sequence prediction. We have chosen the examples belonging to connected workstation device class ($x12$) to perform the comparison. This device class is complex enough to be the representative. Thus,



(a) Recognition for $x11$ device class (8-element bargaining set)

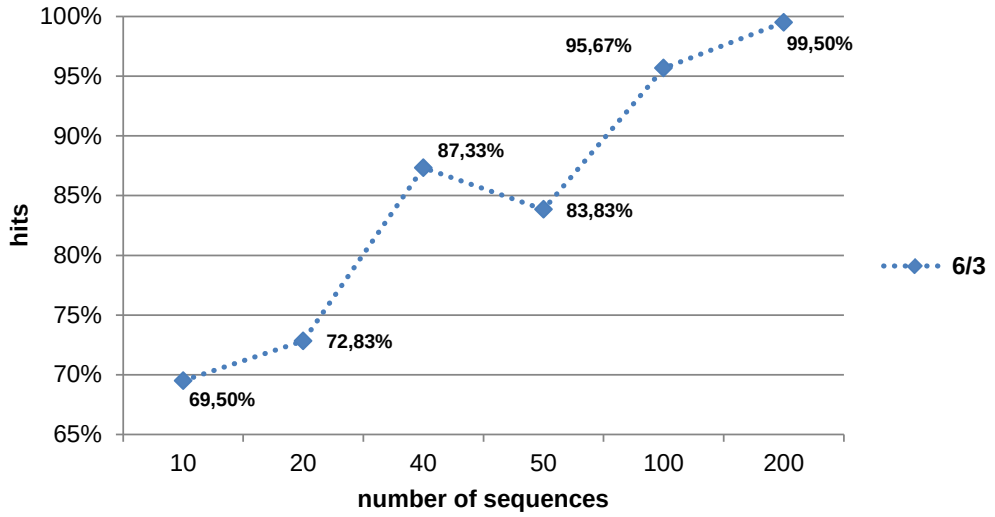


(b) Learning times for $x11$ device class (8-element bargaining set)

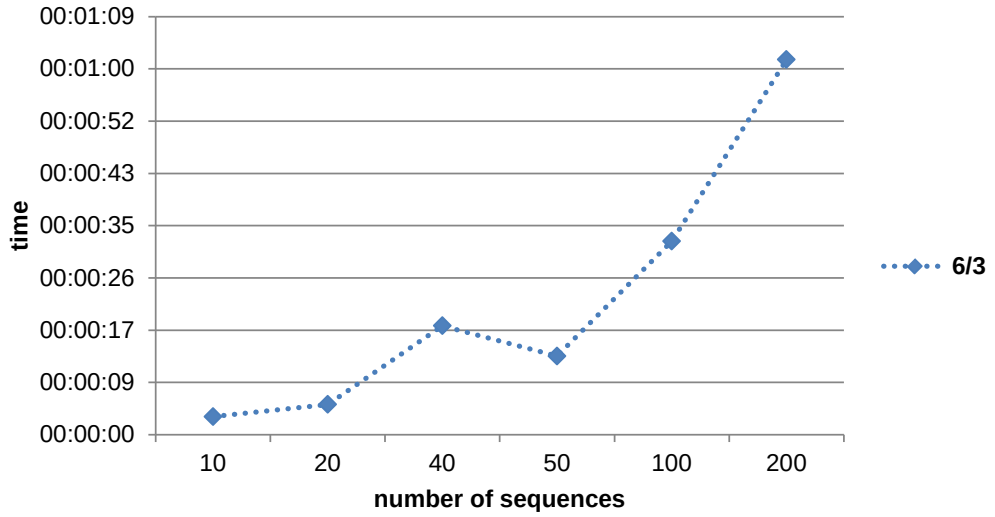
Figure 5.16: Network *netII2* sequence

Table 5.4 shows how many training examples is necessary to obtain 100% hits in sequence prediction. The lengths of the bargaining sets in $x11$ device class are 40.

The number of 40-elements sequences needed to train network *netII1* to reach perfect hits is 50. Network *netII2* needs five times less training



(a) Recognition for $x12$ device class (6-element bargaining set)



(b) Subfigure 2 caption

Figure 5.17: Network *netII2* sequence

examples to receive the same result. Moreover, *netII2* can reach perfect hits when trained with 20-, 15- or even 10-element sequences. Network *netII1* did not reach the perfect hits but for 20−, 15− and 10− element sequences what is indicated by '−'. Table 5.5 shows that *netII2* needed the same time to be trained with 50 sequences consisting of 10 elements as network *netII1*

Table 5.5: Time to reach perfect hits of $x12$ device class trained with $40/x$ sequences

	Time of training the bargaining sets in the sequence prediction to reach 100% hits	
Sequence length	*netII1*	*netII2*
40	$9.48s$	$5.04s$
20	–	$3.03s$
15	–	$12.45s$
10	–	$9.40s$

with 50 sequences consisting of 50 elements. It may be seen that network *netII2* reaches better results than *netII1*. Moreover, it can be trained using sequences which lengths are several times smaller then the sizes of bargaining sets.

5.6 Learning utilization in negotiation

Table 5.6 shows the payoffs of the device (P_1) and the document (P_2) for various fitness levels of the network, indicating how well the network is trained to recognize sequences, the fair result and the payoff got by the document not trained at all (0% fitness level). In each negotiation experiment the device negotiated using Algorithm 1.4, which does not use any AI at all, whereas the document used Algorithm 4.8. The respective payoffs of the negotiation between proactive documents without distinguishing their subclasses, i.e. without considering specific ordering of their trees and the $x11$ device (not connected workstation) are listed in Table 5.6 and depicted in Figure 5.18.

In Section 5.3 we have described how the tests were generated. We have listed there the dictionary files, where the values necessary to decode offers are stored. Having an encoded offer as well as the device and the document dictionary, we can calculate the payoffs as shown in Tables 5.6 and 5.7. Encoding of offers and the way of calculating the payoffs have been described in Chapter 4 in Sections 4.1.1 and 4.1.4.

Table 5.7 shows the payoffs got when trained proactive documents, negotiated with the $x22$ device class (connected laptop). The table shows various fitness levels of the network, from 100% to the completely untrained network (0% fitness level) and two fair results. It is because Equation 1.7 has two fair solutions, i.e. $0.43 \cdot 0.44 = 0.44 \cdot 0.43$ (see the respective fair values for P_1 and P_2 in Table 5.7).

Table 5.6: Payoffs got when negotiating with the $x11$ device class

Fitness level	Fair	100%	99.89%	98.62%	97.93%	97.82%
P_2 *payoff*	0.52	0.47	0.4608	0.4608	0.4608	0.4608
P_1 *payoff*	0.77	0.36	0.384	0.384	0.384	0.384
Fitness level	97.36%	96.09%	93.68%	90.69%	81.49%	0%
P_2 *payoff*	0.4608	0.4608	0.4736	0.2949	0.4736	0.2539
P_1 *payoff*	0.384	0.384	0.36	0.2457	0.36	0.2918

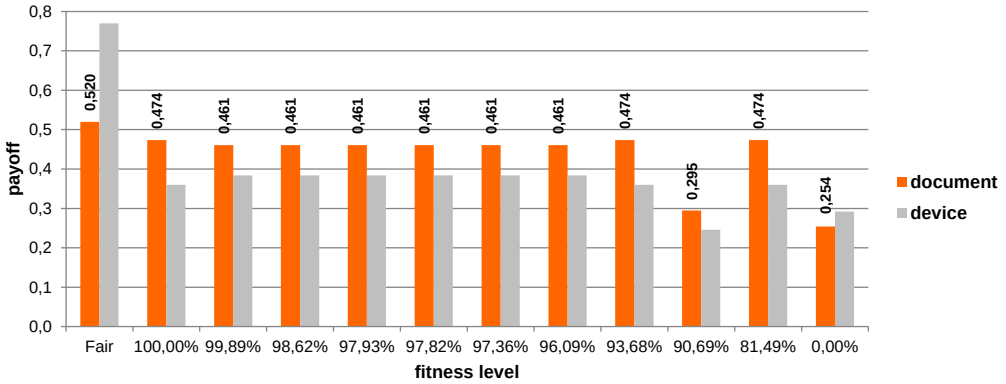


Figure 5.18: Impact of learning on the negotiation process based on $x11$ device class

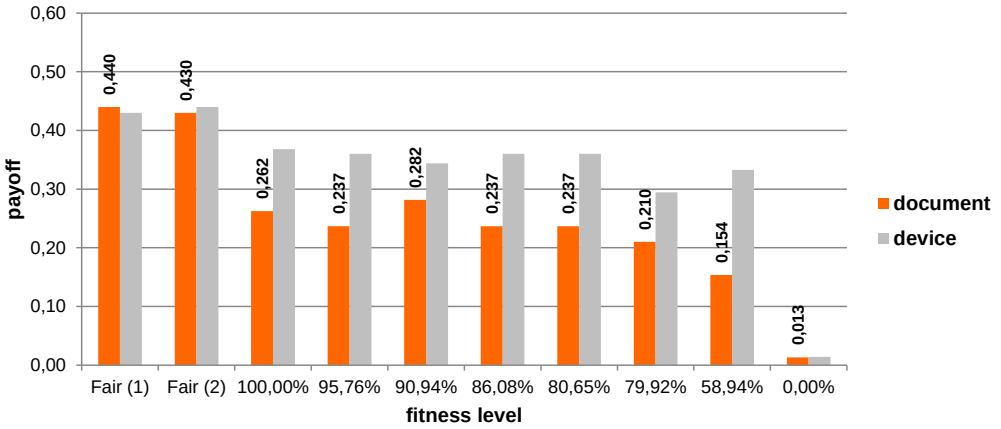


Figure 5.19: Impact of learning on the negotiation between the proactive document class and $x22$ device class

The data presented in Tables 5.6 and 5.7 indicate clearly that payoffs got by trained documents are significantly better than the ones got by documents without any learning capabilities. Moreover, the better learning (higher fit-

Table 5.7: Payoffs got when negotiating with the $x22$ device class

Fitness level	Fair(1)	Fair(2)	100%	95.75%	90.93%
P_2 *payoff*	0.44	0.43	0.2624	0.2368	0.2816
P_1 *payoff*	0.43	0.44	0.368	0.36	0.344
Fitness level	86.08%	80.65%	79.91%	58.93%	0%
P_2 *payoff*	0.2368	0.2368	0.20992	0.1536	0.013
P_1 *payoff*	0.36	0.36	0.2944	0.3328	0.014

ness level) the higher the payoff is.

Higher payoff values shown in Figures 5.18 and 5.19 indicate less negotiation stages. For example, the outcome of the negotiation without AI shown in Table 5.7 was agreed in the 31st move (15th stage), thus discounted 15-times. In general, the obtained improvements follow from finding solution in less number of stages.

Figures 5.18 and 5.19 indicate that *capability* of the document to learn, i.e. fitness level $> 0\%$ allows it to get better payoffs than it would not be trained at all.

5.7 Assessment of sequence clustering

In the most cases considered earlier in this Chapter the problem of recognizing the device class could be solved by the occurrence vector clustering. One exception is distinguishing $x11$ and $x21$ device classes, as shown in Sections 4.1.1 and 4.1.4. Then sequence clustering can be used.

Sequence clustering has also different application from occurrence vector clustering. The occurrence vector approach can distinguish device classes before a bargaining process starts whereas the sequence clustering one can do it during the bargaining process – in the most cases in the first stage of a game, however it is possible to notice the difference between device classes in the later stage – look at Table 5.8.

The following experiment involves sequence clustering. We used the training examples of Group 2, since smaller sequences (as in the examples of Group 1), makes learning more difficult, as indicated by Figure 5.17(a). Moreover, examples of Group 1 could not pass the tests of occurrence vector learning, as explained in Section 5.4.

The outcomes of the experiment are listed in Table 5.8. First, networks *x11netII2* and *x21netII2* were trained using sequences built of all elements of the bargaining set (8/8) and using 200 training bargaining sets. It could make the networks to reach 100% fitness. Then the networks were trained using two element sequences (8/2). The number of training examples was

gradually decreased from 200 to 10. With that, fitness levels between 99% and 70% were reached. Next Algorithm 4.4 was used for negotiations. We can see that when the networks were trained using 400 training examples (even with just two element sequences) 100% hits were obtained (the percent of hits is measured in the same way as in the case of the sequence prediction).

The diagram presented in Figure 5.20 shows results of the experiment in a graphical form. The cumulative height of the column indicates a percentage of the correct guesses in each consecutive stage. It can be seen that at most three stages were needed to obtain the solution (except one case where four stages were needed).

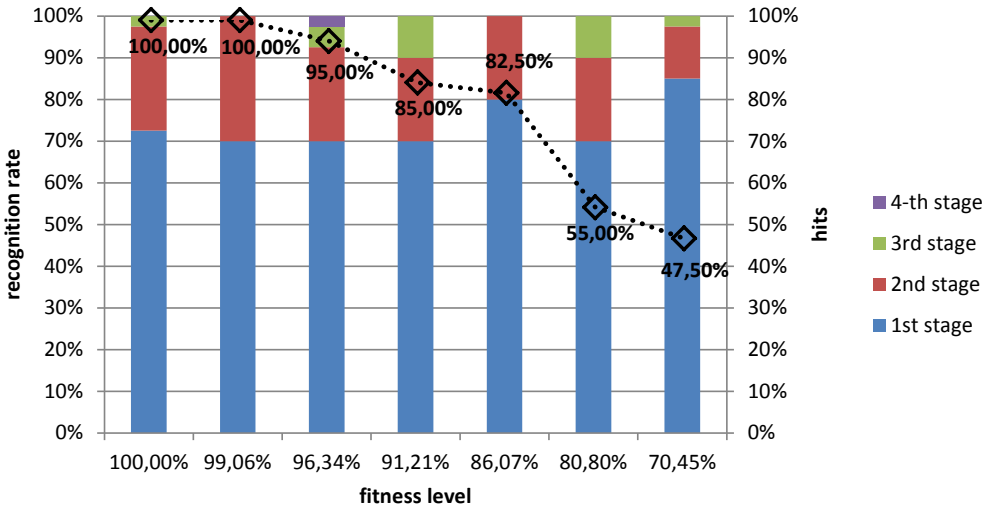


Figure 5.20: Sequence clustering

The first two rows include headers of offers. The upper one indicates the size of the bargaining sets and the lengths of the sequences, the lower one indicates numbers of training examples.

The row labeled 'recognition %' indicates how many percent of the relations, which were created from the examined sequences, were correctly recognized. They were calculated in the same way as in the case of the sequence prediction.

The next three rows, labeled 'hit', 'fail' and 'no solution' denotes results of sequence clustering: *hit* means the device was recognized, *fail* means the device was recognized incorrectly, *no solution* means the device was not recognized.

The last four rows show percent of cases whose solution were found.

The conclusion drawn from their experiment is the following: to obtain

Table 5.8: Sequence clustering

Bargaining set size /sequence length	8/8	8/2	8/2	8/2	8/2	8/2	8/2
Samples	400	400	200	100	80	40	20
recognition %	100%	99.06%	96.34%	91.2%	86.07%	80.8%	70.45%
hit rate	100%	100%	95%	85%	82.5%	55%	47.5%
fail rate	0%	0%	2.5%	2.5%	10%	17.5%	20%
no solution %	0%	0%	2.5%	12.5%	7.5%	27.5%	32.5%
1-st stage %	72.5%	70%	70%	70%	80%	70%	85%
2-nd stage %	25%	30%	22.5%	20%	20%	20%	12.5%
3-rd stage %	2.5%	0%	5%	10%	0%	10%	2.5%
4-th stage %	0%	0%	2.5%	0%	0%	0%	0%

100% accuracy in recognizing devices only 400 training examples are necessary. Moreover, only the first two elements of the 8-element sequence were enough to recognize the correct device class.

Sequence clustering may be also compared to the occurrence vector clustering. The sequence clustering examples presented in the current Chapter concern Group 2. Results presented in Table 5.2 show that in the case of network *netI1* 1000 training examples were necessary to distinguish the correct device class. Thus 1000 examples belonging to all 10 device classes mean 100 examples of one device class. In the case of sequence clustering 400 examples belonging to two device classes means 200 examples of one device class. So we can see that occurrence vector clustering requires relatively less training examples (with regard to one device class).

5.8 Assessment of the intelligent negotiation algorithms

This section presents experiments which summarizes development presented in this Thesis. The negotiations between documents belonging to four types of pro-active documents (documents $d311 \ldots d322$ according to the scheme outlined in Figure 5.5) and ten device classes $x11, \ldots, x52$ (according to Table 3.6) were exercised with 2000 bargaining sets generated according to the policy rules defined in Tables 3.7 – 3.11. Next the respective sets of compatible option trees were generated and 1600 negotiation exercises were performed, where documents were trained to recognize devices and sequences. We generated 20 option trees of each pro-active document class and of each device class. Since all generated option trees of documents and devices were compatible, 80 bargaining sets were created for each pair document – device

(4 documents negotiated with 20 devices). Because there were ten device classes the number of such pairs was 800. Each negotiation utilized two algorithms – with and without AI, so the number of exercises was 1600.

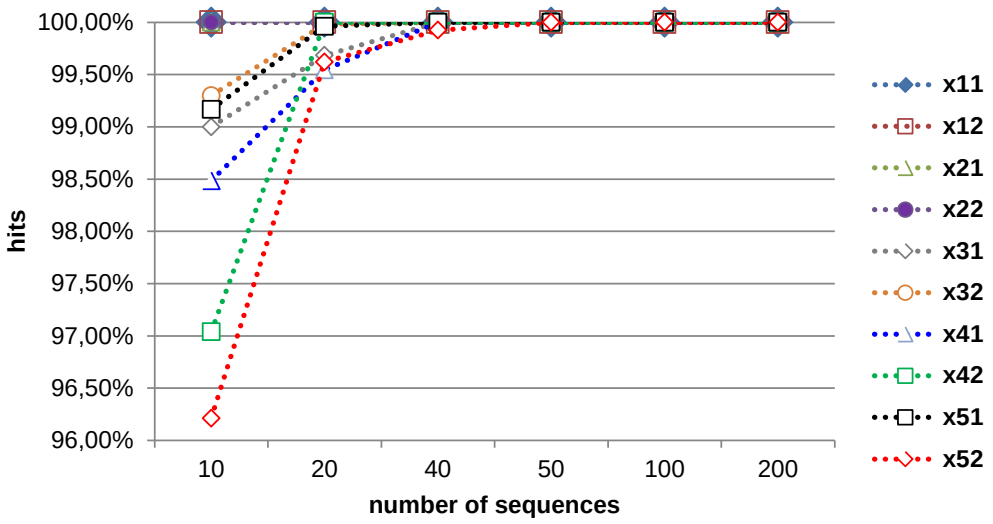


Figure 5.21: Comparing contract prediction for various device classes

The main purpose of these experiments was the comparison of learning utilization in negotiation with each device class. So similar learning conditions for each device class had to be generated. That was why the training examples belonging to Group 3 (the sizes of the training sequences were equal to the lengths of the bargaining sets) were used.

The intermediate result of this comprehensive experiment is presented on the diagram in Figure 5.21. It shows results of the sequence prediction of all device classes with the use for network *netII2*.

The first conclusion is that all results are really good. The worst hit rate (in the case of $x51$ device class) was above 96%. In the case of using 50 or more training examples the hit rate reaches maximum.

Since Group 3 were used in these tests, the sizes of the training sequences differed, depending on the device class. The shorter sequences were $x52$, $x42$ and $x41$, and these device classes gave worse results within the range of 10 – 40 training sequences. Although, even the 'worse' results were, in fact, acceptable. The results of the negotiations between documents and devices are presented in Table 5.9.

As mentioned before (see page 119), the value of the discount factor of both parties was assumed $\delta_1 = \delta_2 = 0.8$. In the negotiations two algorithms were used: Algorithm 1.4 that did not use any AI and Algorithm 4.8 that

Table 5.9: Utilization of learning in negotiation

dev.class	*x11*		*x12*		*x21*		*x22*		*x31*	
AI used	*no*	*yes*	*no*	*yes*	*no*	*yes*	*no*	*yes*	*no*	*yes*
d321	5.74	6.84	1.28	6.67	7.70	7.97	4.68	7.36	7.73	8.24
d322	7.49	7.91	2.88	7.2	5.40	7.05	2.27	6.81	5.40	7.35
d311	9.77	10.08	4.22	6.96	6.73	8.58	0.33	4.72	6.76	9.00
d312	7.64	8.92	2.71	6.35	5.10	8.07	0.09	4.05	8.99	9.71
dev.class	*x32*		*x41*		*x42*		*x51*		*x52*	
d321	5.70	7.11	9.74	9.74	7.33	8.66	10.37	10.39	9.57	9.80
d322	2.19	6.10	8.67	8.83	4.67	7.71	9.41	9.56	8.58	9.00
d311	3.66	5.57	10.28	10.47	5.38	6.98	11.32	11.32	7.85	8.95
d312	4.70	6.32	10.60	10.74	6.29	7.42	11.66	11.42	9.25	9.51

used neural networks. Since 20 negotiations between a document and the device of the same class were run, their payoff were totaled and placed in Table 5.9. The table lists the total values of payoffs agreed by each tested pair of negotiators with documents respectively not trained (Algorithm 1.4) and trained (Algorithm 4.8). It may be seen from Table 5.9 that:

1. The highest gain when using machine learning was reached by documents negotiating with $x12$ (connected workstation) and $x22$ (connected laptop) devices, whereas the lowest one with $x41$ (smart phone) and $x51$ (cellular phone) device classes.

2. In general, higher gains could be reached by documents when negotiating with connected rather than disconnected devices (see Table 3.6).

The reason is that $x12$ and $x22$ device classes exploit the most of the possible attribute values, i.e. their dictionaries consist of more different symbols than the other device classes. Also dictionaries of the connected devices have more symbols, providing a richer material to train documents.

5.9 Implementability considerations

To this point all experiments indicated a significant improvement of the bargaining process when document-agents were properly trained to recognize their opponent device class and sequences specific to it. However, in all 1600 experiments simulation was used instead of real agents – implemented as separate objects of code that have to travel on their own in a distributed system of execution devices. The question arises whether limited CPU and RAM resources of a mobile agent may prevent effective training of the agent to

generalize knowledge on its opponents, as well as make it unable to use that knowledge to recognize opponents and contracts they may prefer. An alternative would be carrying the individual negotiation history by each agent, or keep it at some remote site to consult during negotiation. It however would be unrealistic, as the size of a typical bargaining set may be calculated as the product of the number of symbols provided for each respective attribute value, of the magnitude of 10^3, while the upper bound for the number of possible negotiation histories is a product of their respective permutations, of the magnitude of 10^{12} [32].

Also the negotiation histories kept at some remote site may not always be accessible to the agent during negotiation, as the execution device may not allow the agent to connect to any network.

Therefore our intelligent document-agent should be equipped with the following software:

1. One *netI1* or *netI2* neural network trained to recognize device classes.

2. Ten *netII1* or *netII2* neural networks, each one trained to recognize sequences.

Based on experiments described earlier in the Chapter network *netI2* (shown in Figure 5.2) proved to perform better. Its size in bytes is the sum of all weights and biases times the number of bytes needed to represent a short floating point number, as listed in Table 2.12.

Calculation of the number of floating point numbers needed to represent network *netI2* is the following:

- Layer 1 consists of 20 neurons (see Figure 5.2). Each neuron of the network is stimulated by 82 inputs. It gives 82 weights and 1 bias, so $20 \cdot 83 = 1660$ floating point numbers are required.

- Layer 2 consists also of 20 neurons. Each neuron of the network is stimulated by outputs of the first layer. Each neuron has 20 weights and 1 bias, so $20 \cdot (20 + 1) = 420$ floating point numbers are required.

- Layer 3 consists of 9 neurons. Each neuron is stimulated by 20 outputs from the second layer, so $(20 \cdot 9) + 9 = 189$ floating point numbers are required.

From the above, we get the total of 2269 floating point numbers needed to represent all weights and biases of network *netI2*.

Calculation of the weight of *netII2* network (see Figure 5.4) are the following:

- Layer 1 consists of 10 neurons. Each neuron is stimulated by 174 inputs. It gives 174 weights and 1 bias, so $10 \cdot (174 + 1) = 1750$ floating point numbers are required.

- Layer 2 consists of 1 neuron. The neuron is stimulated by 10 outputs from the first layer. The neuron has 10 weights and 1 bias. It gives so $1 \cdot (10 + 1) = 11$ floating point numbers.

From the above we get for one network the total of 1766 floating point numbers. Since as mentioned above the document agent would need ten networks to recognize all possible sequences. In result we get the total of 17660 floating point numbers needed to represent of all weights and biases of ten networks *netII2*.

Finally, in order to properly recognize the one of ten devices, and then a sequence related to one of them, it needs 11 networks, so it must be able to carry 19929 floating point numbers to use Algorithm 4.8 during negotiation with devices it may arrive to.

For example if 32-bit floating point Java arithmetic is used, the document would need to carry an extra load of $32 \cdot 19929 = 637728$ bits or nearly 80 KB of extra load. This amount is much less than the typical negotiation history that the document might use for intelligent negotiations with the device. Moreover, carrying the latter makes it vulnerable for the analysis by the receiving device, thus would violate our assumptions on privacy of document preferences. Besides, the extra load of slightly less than 80 KB carried by the document-agent is practically nil, given the size of documents measured in MB that are usually sent as email-attachments today. Since the prototype MIND implementation uses email as the transport layer [20], neural networks provide a very attractive mechanism to augment MIND with the intelligent negotiation capability.

Chapter 6

Thesis summary

In the Thesis we have investigated the issue of augmenting proactive documents with the negotiation capability, which can be their significantly new feature, not considered yet in the area of document engineering. However, the concept of negotiating agents has been around for a longer time, and there is an impressive publication record on the research on both: game theoretic models [12] of negotiation and the use of machine learning do improve the overall process [11]. Below we provide a comprehensive survey of the most relevant ideas published in the world literature on the subject. It will allow us to compare and contrast the development of this Thesis to the current state of the art, and evaluate the concept of negotiating documents.

6.1 Related work

The active document that can negotiate technical capabilities of its execution device, as described in the paper, is a novel concept in the context of document engineering [32]. It has roots in the problem discovered a decade ago in connection to implementing the ebXML standard for e-commerce [16]. The standard enables business parties to engage in a transaction by providing means to interchange technical information needed to reach an agreement on terms of the transaction. Expectations of each party are specified in ebXML by the *Collaboration-Protocol Profile (CPP)* document, which is an XML tree composed of a certain number of predefined elements and attributes. When two parties want to conduct business with one another, they have to reconcile their CPPs to form a *Collaboration-Protocol Agreement (CPA)*. The CPA document (also an XML tree) is used to configure the ebXML system to execute the transaction as agreed by the parties.

When parties involved in a business transaction have different expecta-

tions about the required services and their technical parameters, reconciliation of conflicting CPPs would require negotiation. Although automated negotiation has been postulated by OASIS [59] as a method for resolving such conflicts, the specification of the CPA negotiation process has never been made complete [44].

The reason for that has probably been the fact that reconciliation of conflicting CPPs requires automated multi-attribute negotiation strategies, constituting a relatively fresh subject of research [40]. To the best of our knowledge there was only one solution proposed in the literature that addressed the negotiation problem in ebXML [25]; it adopted the concept of modeling the negotiation problem as the Constraint Satisfaction Problem (CSP) [38]. Several attributes of the standard *CollaborationRole* element of CPP, two more elements defining the lifespan of CPA and two attributes of the element *ConversationConstraits* of CPA [16] could be negotiated. One disadvantage of the CSP based negotiation is the relatively high cost of calculating the new offer. Each respective party P_i makes a concession by subtracting from the value of its recent offer some concession value Δ calculated by a function specifically defined for the concession strategy assumed by the party, i.e. formally by calculating $v = U_i(o^{j+1}) = U_i(o^j) - \Delta$. Next P_i must select a set of new instantiations of items of the new offer o^{j+1} by resolving the equation $v = U_i(o^{j+1})$. It requires to assume first the class of the utility functions, for which a solution method is to be used – usually a weighted sum of utilities of the individual items is assumed. This approach, however, limits the capability of execution devices to negotiate with document agents that use only the predefined class of utility functions.

Our option trees introduced in the paper generalize the concept of CPPs, allowing document agents to specify interest in any services and technical parameters of execution devices they may want to execute during the workflow. Moreover, they provide a computationally cost-effective data structure to represent preferences of negotiating parties, without making any specific assumptions on the class of their utility functions. This representation allowed us to model automated bilateral negotiation between document agents and execution devices with a (really) simple bargaining game (SBG) [35]. Negotiation is multiple-issue, no information about preferences of the opponent is required by either party nor any assumption has to be made on the class of functions used to calculate utility of offers – including their discrete values. The only assumption on the negotiation process is that parties start with the offer of their highest utility and make gradual concessions by selecting consecutive offers from their arbitrarily sorted option trees. One advantage of our automated negotiation model is that SBG can be played by document agents and execution devices without any support of external negotiation or

mediation services. It is very important, since in our implementation of the agent system documents and devices are often on their own – whenever no Internet connection is available or is too costly to the owner of the mobile device, if the cellular network is used.

6.1.1 Multi-issue negotiation

Although a generic approach to the problem of multiple issue negotiation with no information about the opponent has been proposed in the literature [40], the formal mathematical proof of the convergence of the monotonic concession strategy, which our SBG implements with option trees, was not provided until [70]. The only property of utility functions of negotiating parties assumed in the latter Reference is that they have to be *concave*, due to the specific geometric interpretation of the space of offers A. The idea of that proof has been to show that the distance between the offers of the two negotiating parties decreases in each round until the contract is agreed. The key notion is the *indifference surface*, defined as the set of all offers of the lowest utility a given party P_i may accept in the current round; all offers of the higher utility lie inside the surface in the m-dimensional space of offers A. Because each party concedes in each consecutive round, their respective indifference surfaces get closer until they finally intersect to form the *zone of agreement*, where the contract may be found – provided of course that after making the concession the next least acceptable utility would not be less that the reservation utility of the party.

This geometrical interpretation may be applied to our SBG game played by document agents and devices negotiating over option trees. Recall the example option trees in Figures 3.2 and 3.3 and the history of bargaining over them listed in Table 3.13. Figure 6.1 illustrates how offers and counteroffers are selected by the negotiating parties – in the manner known in the literature as the *alternating projection protocol* [56]. It may be seen that offers and counteroffers, selected by each party according to the amount of concession either one can accept in the current round are getting closer in the utility space until the contract (offer o_3) is agreed.

The following observations are in order:

1. offers selected by each agent from its option tree are fully bundled, i.e. each offer is complete and includes all items being negotiated in every round;

2. our bargaining set $C \in A$ is discrete and option trees provide strictly monotonic ordering of complete offers for each respective negotiating party;

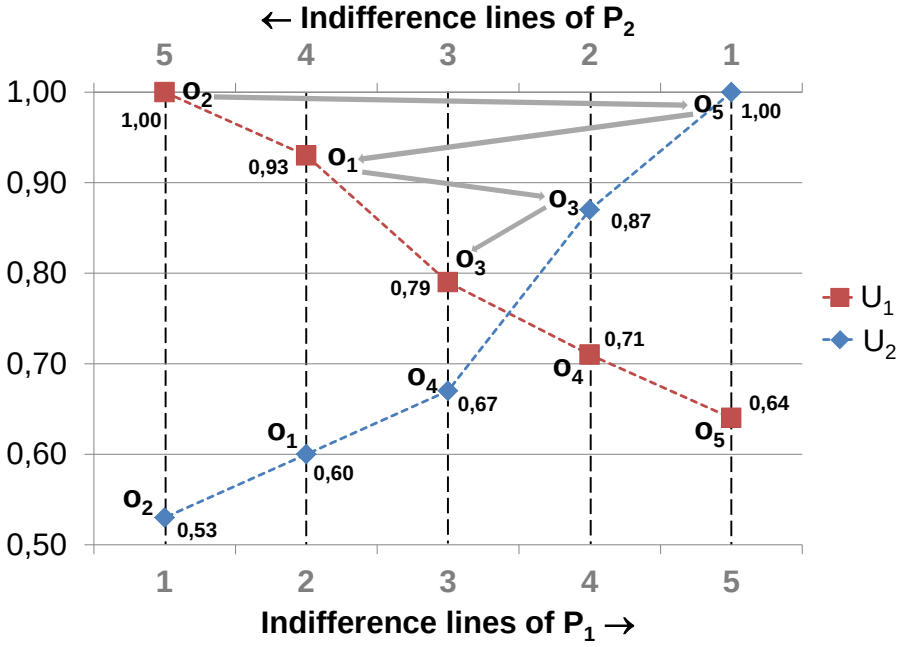


Figure 6.1: The alternating projection protocol in SBG.

3. utility values assigned to each offer in the option tree may be calculated by the each respective party in any way;

4. there is no reservation utility for either party, as the whole range offers specified by the option tree is negotiated.

Based on the existing literature Observation 1 can provide the argument in favor of our approach to negotiate complete offers instead of negotiating them item-by-item. According to [28], agents prefer bargaining over all items simultaneously, rather then separately. Moreover, it has been shown in [5] that when the probability of negotiation breakdown is low, agents always prefer to negotiate complete contracts. Since the cost of computing offers and counteroffers during negotiation oven option trees is negligible, the probability of breakdown in our system of document agents and devices is virtually nil. This is because the realistic deadlines for completing respective document workflow activities are of several orders of magnitude higher that reading offers from the tree or guessing the contract using machine learning. Moreover, because negotiating parties exchange fully-bundled offers the outcome of our SBG game is Pareto-efficient [27].

Observation 2 indicates that no specific assumptions on the properties of

utility functions used by document agents and devices to valuate offers in their respective option trees have to made, except to be injective in order to enable sorting the tree and ensure the monotonicity of preferences. It simplifies the graphical interpretation of the monotonic concession strategy used in SBG (as shown in Figure 6.1), and eludes the assumption on concavity of utility functions – the property that may not be true in many realistic settings [63].

Another important conclusion about our approach based on option trees may be drawn from Observation 3. Note that any utility function may be used to valuate offers in one option tree, even one specific function per each particular offer. Monotonicity of offer preferences in the option tree is in fact the only assumption that the negotiating agent has to make on its opponent. The linear additive utility function defined by Formula 1.5 has been used by us just to simplify generation of option trees for simulation experiments and to generate training sets; in any realistic implementation of our system of (active) document agents, however, option trees of negotiating documents may be designed with any valuation of offers assumed by their originators.

Finally Observation 4 addresses the reservation utility and determines the existence of the agreement zone and success of the negotiation. Since the whole range of offers specified by the option tree of each party is negotiated, the agreement zone always exist in SBG, and both parties (the document and the device) have to concede. It is possible, because of to the embedded functionality of the document-agent, which can adopt execution of the current workflow activity to the current technical capabilities of the execution device. In general the document agent is interested in performing the activity in the best possible way, whereas the execution device is interested in doing that at the lowest possible cost.

6.1.2 Intelligent negotiation

Our document-agent is augmented with learning capability to improve its negotiation competence when reaching agreements with execution devices and to speed-up the negotiation process. In other words, the document can be taught to find the offer of the value satisficing both parties in the number of steps which is less than when just bargaining over the option trees with the device. This capability is important for negotiation scenarios involving mobile execution devices, which computational resources are limited – in particular throughput of the network they can access (if any), or their current battery load.

The major problem to be solved for making negotiation agents intelligent in the sense mentioned above is the lack of detailed knowledge on the

opponent and its tactics, making negotiating behavior of the latter hard to predict. That knowledge must be discovered by analyzing the recorded historical data about the past behavior of the opponent agent, or when no negotiation history is available, by implementing mechanisms enabling dynamic adaptation of the negotiation agent to the opponent's behavior during the actual encounter.

Although the problem of machine learning in the area of game theory has been addressed in the literature for a long time, starting from the works such as [30] or [36], making the negotiation agent capable to predict its opponent's behavior by learning from interactions is a challenge, because of the arbitrary complex behavior patterns the agents may exhibit. The good point to start to study them is the classification of negotiation agents' tactics proposed by [12]: *time dependent*, where offers generated by agents take into account the amount of time remaining to reach the agreement, *resource dependent*, where offers depend on the resources consumed so far, and *behavior* dependent, where the agent imitates the behavior of its opponent. This classification is useful when building models of the opponent's behavior, selecting their parameters and adopting methods to estimate them based on available data. These classes of tactics may be mixed in many ways to model arbitrary complex negotiation behaviors.

The negotiation agent may focus on one or more parameters of its opponent to directly calculate offers that are most likely to be accepted, through modeling the more general characteristics (factors) of the opponent that may drive its specific negotiation behavior, up to attempting to classify the opponent and its preferences.

Learning of the opponent's parameters has been demonstrated in the literature with a variety of methods. For example, it has been shown in [69] that the Bayesian belief network may be utilized by the intelligent agent to learn the opponent's reservation price in multiple-issue negotiations. Non-linear regression approach was successfully applied to predict the approximate value of the opponent's deadline and reservation values [26]. Weights of the linear additive utility function and the reservation price of the opponent can also be learned, e.g. when based on reinforcement learning, or with a genetic algorithm, as demonstrated respectively in [29] and in [9].

Modeling by the negotiation agent of the specific characteristics of its opponent to enable the former to adapt to the behavior of the latter, so that generated offers could lead to reaching the agreement faster, may be represented as the optimization problem. For example, in [67], the best offer in each round is searched in the set of possible offers using the particle swarm optimization (PSO) method [37]. PSO is used to maximize the specially defined objective function, which is parametrized by the *time pressure*

and *eagerness* factors; the latter aggregates history success-or-failure of the last few transactions. Another example of modeling generation of offers as the optimization problem may be found in [3]. Arbitrary complex behaviors may be represented by mixing two characteristics of the opponent: one related to time and indicating to what extent the opponent responds to its own time constraints, and another, called *imitation*, indicating to what extent the opponent responds to the negotiation agents actions. With these two characteristics the negotiation may be modeled as a multi-state control process, so the intelligent agent's task is to determine a sequence of optimal controls, i.e. to predict future offers. One advantage of the optimization approach proposed in [3] over the one proposed in [67] is a significantly reduced number of computations, as the entire solution space does not have to be searched all over again in each round. The negotiation agent may also directly attempt to computationally approximate its opponent utility function, based on the offers received so far. For example, it has been shown that just three regression functions, namely linear, power and quadratic ones, can successfully estimate the opponent's utility values during the encounter, with no prior training data [52].

Machine learning mechanisms may also be directly employed to generating offers by negotiation agents– in particular there exists a quite impressive record of publications on using neural networks for learning from past offers and assist in selecting appropriate tactics [62], [53], [54], [41], [49], [50], [55], [1]. It is generally assumed in the literature that no a-priori knowledge on opponents exists, thus proper off-line training of the network with historical data is not possible, and on-line training during the actual encounter must be performed instead. In consequence major research efforts on the learning capability of negotiation agents have been concentrating on predicting better offers, i.e. offers that may faster lead to the contract, based on the offers submitted by the opponent during some initial series of rounds of the actual encounter. Two problems arise with this approach. One is how many rounds are needed to train the network during the encounter in order to enable it to generate the offer which may successfully conclude the negotiation, and another at what computational cost it can be achieved, i.e. what the minimal size of the network may suffice. Especially the latter is important for any realistic implementation of mobile negotiation agents, which usually lack any significant computational resources. Of course, solutions based on external services may be considered to aid the negotiation agent in that respect, such as NaaS (negotiation-as-a-service) proposed in [1], but it may work only when the document agent has (or is allowed to) access the network from its current execution device. Some hints may be found [53], presenting results of experiments with neural networks of various sizes in a single issue bilateral

negotiation – in encounters of the number of rounds between 100 and 200 the network required just three neurons in the hidden layer and one in the output layer to predict the contract.

Compared to the above our approach also relies on neural networks, as they have been demonstrated to require moderate computational and storage resources, but prefers training of document agents before rather then during the actual encounter with the device. In general, neural networks should perform far better than they are trained off-line, owing to the obviously much reacher set of offers used to train the network.

The following observations are on order:

1. The content of option trees is predetermined by the set of classes of execution devices that the document-agent may encounter during its workflow;

2. Each execution device have some generic preferences imposed by the policy specific to its class.

Based on Observation 1 it may be argued that offers in the bargaining set of each respective class of execution devices combine values of various attributes in the specific way. In consequence, if only the training set includes sufficiently many offers from bargaining sets of each respective devices, the network may be trained to recognize devices based on the offers returned during the actual encounter.

Observation 2 indicates that predicting by the negotiation agent of the exact sequence of offers leading to the agreed contract – the problem addressed before in [3], but from from different premises – may be aimed at recognizing the actual sequence of offers based on the ordering of initial offers during the encounter. It requires the document to be trained off-line to classify sequences according to specific policies of device classes.

6.2 Research contribution

The research statement:

> **Selection of negotiation strategies based on the bargaining model enables effective generation of collaboration agreements between conflicted parties.**

listed in Chapter 1 on page 13 and investigated through the Thesis has been proved. Moreover, it has been shown that the monotonic concession strategy modeled as a Simple Barganing Game may effectively utilize machine learning

mechanisms to speed-up the negotiation process at the relatively low cost. In particular the following novel concepts has been introduced in this Thesis to the area of document engineering:

1. Augmenting active documents with the negotiation capability to resolve conflicts in setting up the execution context with their execution devices [32].

2. Modeling the set of attributes of the execution context using option trees [31], [35].

3. Modeling the negotiation as a simple bargaining game and developing three algorithms to play it (naive, recursive and estimated bargaining) [33], [35].

4. Improving the estimated algorithm with intelligent bargaining utilizing machine learning of the document's opponent preferences based on:

   - the notion of policies providing ordering of attribute preferences [33],
   - neural networks that can recognize device classes and sequences [34].

5. Evaluating estimated and intelligent bargaining approaches in a series of experiments, that proved [33], [34]:

   - both approaches to be realistic in the context of active documents,
   - effective in finding a solution to the conflict,
   - intelligent bargaining based on neural networks performing better in predicting contracts during negotiation then its estimated bargaining counterpart.

6.3 Future work

The a priori approach considered in this Thesis allows learning the agents before the negotiation process. An interesting question is whether agents can acquire newq knowledge during the actual negotiation and bring it to the agency to share with other agents. This is the a posteriori approach just mentioned in this Thesis. The future work should take a closer look at this concept. Certainly, different machine learning mechanisms should be considered., namely reinforcement learning or genetic methods. This prospect is currently investigated by the author.

Appendix A

Supplementary data

All the respective files are included in the enclosed CD for their voluminous size. The included CD content is specified in the README file.

Appendix B

Supplementary algorithms

Algorithm B.1: discount

```
1 Data: discount depth n,
2 player's 1 discount factor d_1,
3 player's 2 discount factor d_2
4 Result: value of the discount factor of the given depth
5 if n = 1 then
6 |   return (1 − d_2);
7 else
8 |   return d_1(1 − discount(n − 1, d_1, d_2));
```

Algorithm B.2: discountFactorsEstimation

```
1 Data: the map of discount factors dfmap,
2 Negotiation history of the given device history
3 Result: value of the discount factor δ
4 Λ ← 0;
5 foreach negotiation in history do
6 |   tmp ← getLambda(negotiation);
7 |   if tmp > Λ then
8 |   |   Λ ← tmp;
9 δ ← getHighestKey(Λ, dfmap);
```

Table B.1: The most important algorithms

Algorithm		Function	Source	Remarks	Programming
ref. no	name	name	code file		language
1.1	`plaSBG()`	`simulate_game2()`	`simulate_game2.m`		Matlab
1.1	`plaSBG()`	`process-bargaining()`	`new-trees.rkt`	handles tree intersection	Racket Scheme (Lisp dialect)
1.2	`submitOffer(SAlg)`	`send_offer()`	`send_offer.m`		Matlab
1.4	`submitOffer(EAlg)`	`send_equilibrium_offer()`	`send_equilibrium_offer.m`		Matlab
2.1	`kMeans()`	`kmeans()`	`hamming.h`		C++
2.2	`updateClusters()`	`update_clusters()`	`hamming.h`		C++
2.3	`chooseCentroid()`	`choose_centroid()`	`hamming.h`		C++
2.4	`updateCentroids()`	`update_centroids()`	`hamming.h`		C++
2.5	`updateCentroid()`	`update_centroid()`	`hamming.h`		C++
4.1	`createOccurrenceVector()`	`list_directory3()`	`Statistics.cpp`		C++
4.2	`sequence2Relations()`	`sequence2Relations2()`	`sequences2relations2.m`		Matlab
4.3	`relations2Sequence()`	`relations2Sequence3()`	`relations2sequences3.m`		Matlab
4.4	`compareSequences()`	`compareSequences()`	`compare_sequences.m`		Matlab
4.8	`submitInformedOffer()`	`send_KAlg_offer()`	`send_KAlg_offer.m`		Matlab

Bibliography

[1] Mohammad Irfan Bala, Sheetal Vij, and Debajyoti Mukhopadhyay. Intelligent agent for prediction in e- negotiation: An approach. In *Proc. Int. Conf. on Cloud Ubiquitous Computing Emerging Technologies*, CUBE 2013, pages 183–187, Nov 2013.

[2] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[3] Jakub Brzostowski and Ryszard Kowalczyk. Predicting partner's behaviour in agent negotiation. In *Proc. 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 355–361, New York, NY, USA, May 2006. ACM.

[4] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. XML dataspaces for mobile agent coordination. In *Proc. 2000 ACM Symposium on Applied computing - Volume 1*, SAC '00, pages 181–188, 2000.

[5] M. Keith Chen. Agendas in multi-issue bargaining: When to sweat the small stuff. Harvard Department of Economics, January 2006.

[6] Paolo Ciancarini, Robert Tolksdorf, and Franco Zambonelli. A survey of coordination middleware for XML-centric applications. *Knowl. Eng. Rev.*, 17:389–405, December 2002.

[7] P. Cichosz. *Systemy uczące się*. Wydawnictwa Naukowo-Techniczne, 2000. in Polish.

[8] Howard Demuth and Mark Beale. MATLAB neural network toolbox user's guide : version 4, 2000.

[9] Deng Dong-mei and Jian Li. An agent negotiation system based on adaptive genetic algorithm. In *Proc. 5th Int. Conf. on Wireless Communications, Networking and Mobile Computing*, WiCom '09, pages 1–4, Los Alamitos, CA, USA, Sept 2009. IEEE.

[10] Paul Dourish, W. Keith Edwards, Anthony LaMarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas B. Terry, and James Thornton. Extending document management systems with user-specific active properties. *ACM Trans. Inf. Syst.*, 18(2):140–170, Apr 2000.

[11] Łukasz Dutka and Jacek Kitowski. Application of component-expert technology for selection of data-handlers in crossgrid. In Dieter Kranzlmüller, Jens Volkert, Peter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2474 of *Lecture Notes in Computer Science*, pages 25–32. Springer Berlin Heidelberg, 2002.

[12] Peyman Faratin. *Automated Service Negotiation Between Autonomous Computational Agents*. PhD thesis, University of London, Queen Mary and Westfield College, 2000.

[13] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[14] Lee Garber. Melissa virus creates a new type of threat. *Computer*, 32(6):16–19, 1999.

[15] David Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[16] Brian Gibb and Suresh Damodaran. *ebXML - Concepts and Application*. Wiley Publishing Inc., 2003.

[17] Robert J. Glushko and Tim McGrath. *Document Engineering – Analyzing and Designing Documents for Business Informatics and Web Services*. MIT Press, 2008.

[18] Magdalena Godlewska. *A model of the open architecture of distributed electronic documents supporting collaborative decision making*. PhD thesis, Gdansk University of Technology, Faculty of ETI, http://pbc.gda.pl/Content/32567/phd_godlewska_magdalena.pdf, 2013. in Polish.

[19] Magdalena Godlewska and Bogdan Wiszniewski. Distributed MIND – a new processing model based on mobile interactive documents. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 6068 of *Lecture Notes in Computer Science*, pages 244–249. Springer, Berlin/Heidelberg, 2010.

[20] Magdalena Godlewska and Bogdan Wiszniewski. Smart email: Almost an agent platform. In Tarek Sobh and Khaled Elleithy, editors, *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, volume 313 of *Lecture Notes in Electrical Engineering*, pages 581–589. Springer International Publishing, 2015.

[21] Google. Responsive design – harnessing the power of media queries. http://googlewebmastercentral.blogspot.jp/2012/04/responsive-design-harnessing-power-of.html, April 30, at 6:07 PM 2012. Webmaster Central Blog.

[22] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Techincal Journal*, 29:147–160, 1950.

[23] D.J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. A Bradford book. MIT Press, 2001.

[24] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Advanced book program: Addison-Wesley. Addison-Wesley, 1991.

[25] Ouassila Hioual and Zizette Boufaida. Using intelligent agents in conjunction with heuristic model for negotiating ebXML CPP. In *Proc. 2nd IEEE Int. Conf. on Information and Communication Technologies: from Theory to Applications*, ICTTA '06, pages 268–273, Los Alamitos, CA, USA, 2006. IEEE.

[26] Chongming Hou. Predicting agents tactics in automated negotiation. In *Proc. IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, IAT 2004, pages 127–133, Los Alamitos, CA, USA, Sept 2004. IEEE.

[27] Younghwan In and Roberto Serrano. Agenda restrictions in multi-issue bargaining (II): unrestricted agendas. *Economics Letters*, 79(3):325 – 331, 2003.

[28] Roman Inderst. Multi-issue bargaining with endogenous agenda. *Games and Economic Behavior*, 30(1):64–82, 2000.

[29] Li Jian. An agent bilateral multi-issue alternate bidding negotiation protocol based on reinforcement learning and its application in e-commerce. In *Proc. 2008 Int. Symp. on Electronic Commerce and Security*, pages 217–220, Los Alamitos, CA, USA, Aug 2008. IEEE.

[30] James S. Jordan. The exponential convergence of bayesian learning in normal form games. *Games and Economic Behavior*, 4(2):202–217, 1992.

[31] Jerzy Kaczorek. Negocjacyjna metoda generowania protokołu uzgodnień na platformie ebXML. In *Mat. I Konf. Studentów i Doktorantów Elektroniki, Telekomunikacji, Informatyki, Automatyki i Robotyki (ICT Young)*, Zeszyty Naukowe WETI PG, Tom 1, pages 279–286. Politechnika Gdańska, 28-29 May 2011, Gdańsk, Poland. in Polish.

[32] Jerzy Kaczorek and Bogdan Wiszniewski. Augmenting digital documents with negotiation capability. In *Proc. 2013 ACM Symposium on Document Engineering*, pages 95–98. ACM, 10-13 Sept. 2013, Florence, Italy.

[33] Jerzy Kaczorek and Bogdan Wiszniewski. Bilateral multi-issue negotiation between active documents and execution devices. In *Proc. 9th Int. Conf. on Digital Society (ICDS 2015)*, pages 52–58. IARIA XPS Press, 22-27 Feb. 2015, Lisbon, Portugal.

[34] Jerzy Kaczorek and Bogdan Wiszniewski. Document agents with the intelligent negotiation capability. In *Proc. Knowledge and Cognitive Science and Technologies (KCST 2015)*, The 19th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2015), 12-15 July 2015, Orlando, Florida, USA. in press.

[35] Jerzy Kaczorek and Bogdan Wiszniewski. A simple model for automated negotiations over collaboration agreements in ebXML. In *Proc. 13th IEEE Conf. on Commerce and Enterprise Computing, CEC 2011*, pages 167–172. IEEE, 5-7 Sept. 2011, Luxembourg, Luxembourg.

[36] Ehud Kalai and Ehud Lehrer. Rational learning leads to Nash equilibrium. *Econometrica*, 81(5):1019–1045, 1993.

[37] James Kennedy and Russel Eberhart. Particle swarm optimization. In *Proc. IEEE Int. Conf. on Neural Networks,*, volume 4, pages 1942–1948, Los Alamitos, CA, USA, Nov 1995. IEEE.

[38] Ryszard Kowalczyk and Van Bui. On Constraint-Based Reasoning in e-Negotiation Agents. In Grzegorz Rozenberg and Frits W. Vaandrager, editors, *Agent-Mediated Electronic Commerce III*, volume 2003 of *Lecture Notes in Computer Science*, pages 31–46. Springer, Berlin Heilderberg, 2001.

[39] Sarit Kraus, Katia Sycara, and Amir Evenchik. Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[40] Guoming Lai and Katia Sycara. A generic framework for automated multi-attribute negotiation. *Group Decision and Negotiation*, 18(2):169–187, 2009.

[41] Ning Liu, DongXia Zheng, and YaoHua Xiong. Multi-agent negotiation model based on RBF neural network learning mechanism. In *Proc. of the Int. Symp. on Intelligent Information Technology Application*, IITAW '08, pages 13–136, Dec 2008.

[42] D.J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[43] MeNaID project home page. Methods and tools of future document engineering. http://www.menaid.org.pl.

[44] Dale Moberg and Sacha Schlegel. OASIS ebXML Collaboration Protocol Profile and Agreement (ebCPPA) v3.0. OASIS Week of ebXML Standards Webinars, June 2007. slide 15.

[45] John Nash. The bargaining problem. *Econometrica*, 18(2):155–162, April 1950.

[46] John Nash. Non-cooperative games. *Annals of Mathematics*, pages pp. 286–295, 1951.

[47] OASIS. *Collaboration Protocol Profile and Agreement Specification Version 2.0*, September 2002.

[48] Christopher Olston and Marc Najork. Web crawling. *Found. Trends Inf. Retr.*, 4(3):175–246, 2010.

[49] Ioannis V. Papaioannou, Ioanna Roussaki, and Miltiades E. Anagnostou. Neural networks against genetic algorithms for negotiating agent behaviour prediction. *Web Intelligence and Agent Systems*, 6(2):217–233, 2008.

[50] Ioannis V. Papaioannou, Ioanna Roussaki, and Miltiades E. Anagnostou. Using neural networks to minimize the duration of automated negotiation threads for hybrid opponents. *Journal of Circuits, Systems, and Computers*, 19(1):59–74, 2010.

[51] Thomas A. Phelps and Robert Wilensky. Toward active, extensible, networked documents: multivalent architecture and applications. In *Digital Libraries*, pages 100–108, 1996.

[52] Fenghui Ren and Minjie Zhang. Prediction of partners' behaviors in agent negotiation under open and dynamic environments. In *Proc. 2007 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology*, pages 379–382, Los Alamitos, CA, USA, Nov 2007. IEEE.

[53] Ioanna Roussaki, Ioannis V. Papaioannou, and Miltiades E. Anagnostou. Employing neural networks to assist negotiating intelligent agents. In *Proc. 2nd IET Int. Conf. on Intelligent Environments*, volume 1 of *IE '06*, pages 101–110, Los Alamitos, CA, USA, July 2006. IEEE.

[54] Ioanna Roussaki, Ioannis V. Papaioannou, and Miltiades E. Anagnostou. Building Automated Negotiation Strategies Enhanced by MLP and GR Neural Networks for Opponent Agent Behaviour Prognosis. In *Proc. 9th Int. Conf. on Artificial Neural Networks*, volume 4507 of *Lecture Notes in Computer Science*, pages 152–161. Springer, 2007.

[55] Ioanna Roussaki, Ioannis V. Papaioannou, and Miltiades E. Anagnostou. Using neural networks for early detection of unsuccessful negotiation threads. *Int. Journal on Artificial Intelligence Tools*, 20(3):457–487, 2011.

[56] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.

[57] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2010.

[58] L. Rutkowski. *Metody i techniki sztucznej inteligencji*. Informatyka - Zastosowania. Wydawnictwo Naukowe PWN, 2009. in Polish.

[59] Martin Sachs, Arvola Chan, Jamie Clark, Chris Ferris, Brian Hayes, Neelakantan Kartha, Kevin Liu, Heiko Ludwig, Pallavi Malu, Dale Moberg, Himagiri Mukkamala, Peter Ogden, Yukinori Saito, Krishna Sankar, and Jean Zheng. Negotiation requirements. OASIS/ebXML CPPA Technical Committee, March 2002.

[60] Ichiro Satoh. Mobile agent-based compound documents. In *Proc. of the 2001 ACM Symposium on Document Engineering*, DocEng '01, pages 76–84, 2001.

[61] Jacek Siciarek, Maciej Smiatacz, and Bogdan Wiszniewski. For your eyes only - biometric protection of PDF documents. In *2013 Int. Conf. on e-Learning. e-Business, Enterprise Information Systems and e-Government (EEE 2013)*, pages 212–217, 22-25 July, 2013, Las Vegas, Nevada, USA.

[62] Von-Wun Soo and Chun-An Hung. On-line incremental learning in bilateral multi-issue negotiation. In *Proc. First Int. Joint Conf. on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 314–315, New York, NY, USA, 2002. ACM.

[63] Katia Sycara and Tinglong Dai. Agent reasoning in negotiation. In Marc D. Kilgour and Colin Eden, editors, *Handbook of Group Decision and Negotiation*, volume 4 of *Advances in Group Decision and Negotiation*, pages 437–451. Springer, 2010.

[64] R. Tadeusiewicz. *Sieci neuronowe*. Problemy Współczesnej Nauki i Techniki: Informatyka. Akademicka Oficyna Wydawnicza, 1993.

[65] Robert Tolksdorf. Workspaces: A Web-based workflow management system. *IEEE Internet Computing*, 6(5):18–26, 2002.

[66] J. Von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.

[67] Zhu Wang and Lingfeng Wang. Adaptive negotiation agent for facilitating bi-directional energy trading between smart building and utility grid. *IEEE Transactions on Smart Grid*, 4(2):702–710, 2013.

[68] J. Watson. *Strategy: An Introduction to Game Theory (Third Edition)*. W. W. Norton, 2013.

[69] Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. *Int. Journal of Human-Computer Studies*, 48(1):125–141, 1998.

[70] Ronghuo Zheng, Nilanjan Chakraborty, Tinglong Dai, Katia Sycara, and Michael Lewis. Automated bilateral multiple-issue negotiation with no information about opponent. In *Proc. 47th Hawaii Int. Conf. on System Sciences*, HICSS 2013, pages 520–527, Los Alamitos, CA, USA, 2013. IEEE.