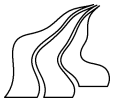
Learning Utility Functions by Imputing



Group E1-119

The Faculty of Engineering and Science
Aalborg University

AALBORG UNIVERSITET

Department of Computer Science

Title:
Learning Utility Functions by Imputing

Period:
February 1^{st} – June 10^{th}
2005

Project group:
E1-119

Group members:
Anders Hansen
Nicolaj Lock
Peter Poulsen

Supervisor:
Thomas D. Nielsen

No. of copies: 7

No. of pages: 76

Appendix pages: 33

Total no. of pages: 113

Abstract:

In this project two methods, called Utility Iteration and Imputing by Comparison, are developed. These methods learn the utilities of an observed agent, such that its preferences can be modeled in an influence diagram. The utilities are learned from the behavior of the agent, by creating constraints based on the observed decisions made by it. The two methods are designed to handle agents that change behavior, using different policies to handle conflicting behavior. The methods have much in common with *FLUF*, developed in Hansen et al. (2004), the main difference being how partially observed strategies are handled. Where *FLUF* relaxes constraints to ensure that the true utility function is not excluded, then Utility Iteration and Imputing by Comparison impute observations to make the strategy fully observed, thereby removing the need to relax constraints. Experiments are conducted with the two new methods and with *FLUF*. Three different kinds of changing behavior are defined, and experiments are conducted with respect to each kind. Both in the experiments with changing behavior and the experiment with static behavior, the new methods achieved better results than *FLUF*, both with regard to accuracy and learning speed. It is concluded that, under the assumption made about the domain in this project, imputing observations will yield a higher accuracy than relaxing constraints.

Preface

This project is done by group E1-119 at Aalborg University department of Computer Science. We would like to thank Hugin Expert A/S for allowing us to use a full version of their program Hugin Researcher. Also, we would like to thank Thomas D. Nielsen for helping with the entire project.

Aalborg June 10^{th}. 2005

Anders Hansen

Nicolaj Lock

Peter Poulsen

CONTENTS

CHAPTER

ONE

Introduction

This project focuses on learning utility functions in influence diagrams. The utility function expresses an agent's preferences for different scenarios. It might be essential to know these preferences in certain decision processes, for instance when marketing a new product it is important to know the preferences of the consumer, otherwise the marketing campaign may fail to reach the intended recipients. It can also improve the artificial intelligence in computer games, such that the game can learn the player's strategy thus making the game more challenging.

There are other areas in which knowing the utility function can aid the decision maker, therefore a method for learning the utility function is needed. Such methods are called *utility learning methods*. Hansen et al. (2004) developed a utility learning method for agents with or without changing behavior. The method developed by Hansen et al. (2004) is called *FLUF (FLUF Learning Utility Function)* and the general idea is that *FLUF* learns the utility function by observing the behavior of an agent. The agent is assumed to be a *rational agent*, that is an agent that always tries to maximize the expected benefit of its decision.

Basically *FLUF* derives some boundaries, called *constraints*, from the observations made. These constraints are inequalities that exclude a set of utility values that cannot explain the observations. *FLUF* is designed to handle *partially observed strategies*, that is situations where the decisions are not observed for all configurations of the influence diagram. In general *FLUF*'s way of handling these unobserved configuration is to relax the already mentioned constraints for the utility values. These relaxations are done such that it is assured that any decision in the unobserved configuration could be explained with some utility values within the allowed boundaries. To expand the area of application for *FLUF*, it is also designed to handle changing behavior, where the utilities can change over time which can cause *conflicting* or *inconsistent* observations.

The experiments from Hansen et al. (2004) showed that predictions potentially could be done more accurately than done by *FLUF*. Some of the inaccuracy that *FLUF* has when it comes to predicting decisions will most likely be derived from the relaxation of the constraints. If the agent's strategy is *fully observed*, i.e. the decisions are observed in all configurations, it will not be necessary to relax the constraints.

Many of the ideas used by Hansen et al. (2004) will be incorporated in this project, such as creating constraints for the utility values based on the observations. One of the more important differences between Hansen et al. (2004) and this project is in the way partially observed

strategies are handled. This project investigates the idea of *imputing* to approximate the unobserved parts of an agent's strategy. A naive way of imputing would be to randomly select decisions for the unobserved strategies. This method would likely lead to wrong imputations, therefore another method for imputing should be employed. One such way could be to use the observations already made, in that the preferences in these observations should provide a hint on what to expect with regard to future observations. This project investigates different imputing methods that impute based on the observations already made.

1.1 Prerequisites

This section describes the assumptions made. Assumptions are made about the prior knowledge available and the agent being observed. The assumptions made for the methods are similar to the prerequisites for *FLUF*, however the observations can contain *noise* in the new methods, that is part of the observation may be corrupted, i.e. by faulty hardware or bad network transmissions.

1.1.1 The Agent

The utilities of the agent are not known from the beginning, and these utilities have to be estimated, through observing the behavior of the agent. It is assumed that the agent tries to maximize the expected utility, i.e. the agent is behaving rationally.

The agent is not restricted from changing its utilities, so utility learning methods should handle changing behavior. However, it is assumed that the observed agent is not using a similar learning algorithm since this may lead to infinite cycles of mutual predictions (Chajewska et al., 2001). The possibility that the observed agent can change its preferences, enables it to perform an action given some configuration at one time, but then later, given the same configuration, perform a different action. Two such observations can only be explained if the expected utility of the two actions are equal or if the agent has changed behavior.

1.1.2 Prior Knowledge

As this project only focuses on the learning of an agent's utilities, it is assumed that the agent's perception of the variables and their causalities in the environment are known, that is the probabilities are known. In effect it assumed that, with the exception of the utility function, it is possible to create an influence diagram for the agent's decision scenario.

1.2 Goals of this Project

The goal of this project is, based on the ideas from Hansen et al. (2004), to create new utility learning methods that use imputing to handle partially observed strategies. Imputing methods should impute the unobserved decisions based on what is already observed. Moreover the methods should be able to handle conflicts since these may occur due to the imputing, and since the agent is allowed to be change behavior. Experiments should be conducted using the imputing method, to determine its *speed* and *accuracy*, such that it can be compared to similar experiments with *FLUF*. Speed being measured in number of training cases, and accuracy begin measured by comparison of the expected utility of the strategy predicted by the utility learning method and the strategy used by the agent.

1.3 The Structure of the Report

The rest of this report is organized in the following chapters:

Chapter Two: A presentation of *FLUF* is given as many of the ideas used to develop the new imputing method is based on concepts developed in *FLUF*. Furthermore potential limitations of *FLUF* are analyzed.

Chapter Three: Based on the analysis from chapter two, general ideas on imputing are presented, and two imputing algorithms are designed and presented. Finally their potentials and limitations are analyzed.

Chapter Four: Changing behavior of the observed agent is analyzed and presented together with possible methods for handling this.

Chapter Five: The algorithms designed in chapter three, are tested in a series of experiments. These experiments and the results are presented in this chapter.

Chapter Six: Summary of the most significant considerations and conclusions from the previous chapters.

CHAPTER

TWO

The Method FLUF

In this chapter the method *FLUF*, and the policies for handling conflicting observations, are presented. *FLUF* is based on work by Chajewska et al. (2001) which estimates the utilities in a decision tree by establishing a feasible space in which the utility values are to be found. One main difference between *FLUF* and the work by Chajewska et al. (2001) is that *FLUF* is defined in terms of an influence diagram instead of a decision tree, this difference will be discussed in Section 2.6.

The overall idea in *FLUF* and Chajewska et al. (2001) is that, based on the fact that the agent is rational, it can be assumed that the expected utility of an observed decision is higher than the expected utility of its alternatives. This is used to reduce the number of possible utility functions, by generating inequalities that express the relationship observed. Among the remaining utility functions one is chosen by *FLUF*. This is then used as an estimate of the agent's utility function.

First influence diagrams and some notation used throughout this report is presented. Then the assumptions for *FLUF* will be presented and then the method for estimating the utility values if the strategy of the observed agent is fully observed will be presented. This will be followed by a method for handling situations where the strategy is partially observed, which is the method actually used by *FLUF*.

2.1 Influence Diagrams

Decision scenarios are often represented as influence diagrams. Influence diagram can encode preferences and utilities of a decision maker in the decision scenario. The influence diagram can then be used by the decision maker to determine what decisions that would yield the highest expected utility. In this section the syntax and semantics of these will be presented, as influence diagrams are used by *FLUF* and new methods developed in this project.

Syntax

An influence diagram consists of a directed acyclic graph over chance nodes, decision nodes and utility nodes, with the following qualitative properties, (Jensen, 2001):

- there is a directed path comprising all decision nodes
- the utility nodes have no children

For the quantitative specification, it is required that:

- the decision nodes and the chance nodes have a finite set of mutually exclusive and exhaustive states
- a conditional probability table $P(A|pa(A))$ is attached to each chance node A
- a real-valued function over $pa(M)$ is attached to each utility node M, called a local utility function

where $pa(N)$ is the parents of the node N.

Semantics

The utility nodes represent some gain or loss for the decision maker, where each utility node's contribution is determined by its utility function. The chance nodes represent elements that may, directly or indirectly, influence the gain or loss of the decision maker, or provide information for the decision maker. The decision nodes represent the choices that have to be made by the decision maker.

The decisions are ordered relatively to each other with respect to when they are made, this is called *the temporal order*. Graphically, information precedence is represented as *information links*; there is an information link from a chance node A to a decision node D_i if the chance node is observed before decision D_i but after decision D_{i-1}. The temporal order of the decision nodes is represented graphically by links such that if decision D_1 is made before decision D_2 then there exist a directed path from D_1 to D_2. The temporal order also orders the chance nodes according to when they are observed. Any link that is not an information link is termed a *relation link*.

A general assumption when dealing with influence diagrams is *no-forgetting*, which means that for some decision node the decision maker knows all the choices for decision nodes prior to the current decision node and the states of the observed chance nodes prior to any of the decision nodes earlier in the temporal order or the current decision node.

I_0 denotes the set of chance nodes that are observed before any decision is taken and I_1 denotes the set of chance nodes that are observed after the first decision and before the second. If there are n decision nodes, then I_n denote the set of chance nodes that are observed after the last decision or not observed at all. This establishes the following temporal order: $I_0 < D_1 < I_1 < \ldots < D_n < I_n$. The ordering of the nodes can be deduced from the links and the no-forgetting assumption.

2.1.1 Solving Influence Diagrams

When using influence diagrams to determine what the best decisions are, it called *solving the influence diagram*. The method for solving influence diagrams rely on the chain rule for influence diagram, which is as follows (Jensen (2001)):

Theorem 2.1 *Let ID be an influence diagram with the universe $W_C \cup W_D$. Then*

$$P(W_C|W_D) = \prod_{X \in W_C} P(X|pa(X))$$

where W_C is the set of chance nodes and W_D the set of decision nodes.

Let ID be an influence diagram over $W = W_C \cup W_D$. W_C is the set of all chance nodes in ID while W_D is the set of all decision nodes. Let the temporal order of the variables be described as $I_0 < D_1 < I_1 < \ldots < D_n < I_n$ and let $V(pa(U)) = \sum_i V_i(pa(U_i))$ where V_i is the local utility function for utility node U_i. Then the maximum expected utility is:

$$MEU(ID) = \sum_{I_0} \max_{D_1} \sum_{I_1} \max_{D_2} \ldots \max_{D_n} \sum_{I_n} P(W_C|W_D)V(pa(U)) \tag{2.1}$$

Equation 2.1 is only a principle solution since the size of $P(W_C|W_D)$ grows exponentially. It is possible to avoid this problem by using the distributive law to eliminating the variables one by one, to reduce the size of the largest probability table. Sum-marginalization (marginalization of chance variables) and max-marginalization (marginalization of decision nodes) cannot interchange and the marginalization is therefore restricted by the temporal order, this is called a *strong marginalization.*

Definition 2.1 *A* policy *for a decision node D_i is a mapping σ_i, which for any configuration of the past of D_i yields a decision for D_i such that:*

$$\sigma_i(I_0, D_1, I_1, \ldots, D_{i-1}, I_{i-1}) \in sp(D_i)$$

where $sp(D_i)$ is the state space of D_i. A strategy *consists of a set of policies one for each decision in the influence diagram. A* solution *to an influence diagram is the strategy that maximizes the expected utility.*

If the decision maker acts rationally for decision D_i and all future decisions, i.e. makes the decision that maximizes the expected utility, then the solution is the strategy that comprises all optimal policies (Definition 2.1).

Using the concept of policies, an operational algorithm for calculating the maximum expected utility can be described, this is done in Lemma 2.1. This is operational as the joint probability for all chance nodes is never calculated, but rather only the joint probability for a subset of the chance nodes is calculated at each step.

Lemma 2.1 *Let σ_i denote the policy for the decision node D_i. The maximum expected utility for the node D_i is denoted by ρ_{D_i}. Let n be the number of decision nodes, then the maximum expected utility for the decision node D_i, where $i \leq n$ is found by*

$$\rho_{D_i}(past(D_i)) = \begin{cases} \max\limits_{D_i} \sum_{I_i} P(I_i|past(D_i), D_i) \cdot \rho_{D_{i+1}}(past(D_{i+1}), D_{i+1}) & i \neq n \\ \max\limits_{D_i} \sum_{I_i} P(I_i|past(D_i), D_i) \cdot V(pa(U)) & i = n \end{cases} \tag{2.2}$$

where $past(D_i)$ denotes a configuration of the past of decision node D_i.

Relevant Past

In Lemma 2.1 it is necessary to consider the entire past of the decision node, because the expected utility is being calculated, and the global utility function may depend on nodes that are not in the *relevant past* of the decision node. However, if the expected utility is not needed, but the optimal decision must be found, it is enough to examine the relevant past of a decision node. Using only the relevant past of the decision nodes rather than the entire past when the optimal decision must be found, reduces the number of configurations of the past of the decision nodes that have to be investigated.

Definition 2.2 (Relevant Past) *A decision or chance node X is in the relevant past of a decision node D, if there exists a configuration of the past of D (denoted $\overline{y}$) and two instantiations of X (x and x') where X is in the past of D, such that the decision made in node D is different for the two instantiations of X, i.e.: $\delta_D(\overline{y}, x) \neq \delta_D(\overline{y}, x')$*

An analysis of relevant past is given in Shachter (1999) and an algorithm for finding the relevant past is found in Shachter (1998).

2.2 Assumptions

This section describes the assumptions needed for *FLUF*. The general idea in *FLUF* aswell as in Chajewska et al. (2001), is to determine the relationship between the observed decision given some past, and the alternative decisions. When a decision is observed it can be assumed that the expected utility of that decision is greater than for any of the alternatives, as the agent is assumed to be rational, i.e. always make decisions that maximize the expected utility.

FLUF uses an influence diagram when trying to estimate the utility function of the agent, so one such must be given. The local utility functions are assumed to be unknown.

In Figure 2.1 an example of an influcence diagram is given, where both D and C are binary, thus the probability table for C has four entries, shown in Table 2.1. Similarly the local utility functions can be expressed as tables, shown in Table 2.2. Such tables are called *utility tables*. Each value returned by the utility functions could also be considered as a single utility value (denoted v_i) which is shown in Table 2.3. Furthermore, the utility values are assumed to be normalized, therefore the space spanning the utilities is bounded by $0 \leq v_1 \leq 1, \cdots, 0 \leq v_m \leq 1$ if there are m utility values, which means that there are m different configurations of $pa(U)$, this region is called the *normalized region*. Within this normalized region, constraints, describing the relationship between the expected utility of the observed decision and the alternative decisions (elaborated later in Section 2.3 and Section 2.4), are added. The space spanned by these constraints is called the *feasible space* or the *utility space*.

For practical reasons, which will be described later, it is an advantage to include each utility in every cell of all utility tables. This is done by multiplying each utility with a coefficient that is either zero or one. An example of this is shown in Table 2.4 where the entry $V_1(c_1)$ is described by the coefficients $(1, 0, 0, 0)$ to stipulate that v_1 is the only relevant utility.

In general, each cell in all utility tables is given a cell number, such that no two cells have the same number. So the local utility function for utility node U_j can be described as:

$$V_j(pa(U_j)) = \sum_{i=1}^{m} \alpha_{j,i,pa(U_j)} v_i$$

where i denote cell numbers. $\alpha_{j,i,pa(U_j)}$ is 1 if i is the number assigned to the cell corresponding to the parent configuration $pa(U_j)$, and 0 for all others.

C	d_1	d_2
c_1	p	q
c_2	1 - p	1 - q

Table 2.1: $P(C|D)$ from Figure 2.1

	c_1	c_2
U_1	$V_1(c_1)$	$V_1(c_2)$

	c_1	c_2
U_2	$V_2(c_1)$	$V_2(c_2)$

Table 2.2: $U_1(C)$ and $U_2(C)$, from Figure 2.1, as utility functions

	c_1	c_2
U_1	v_1	v_2

	c_1	c_2
U_2	v_3	v_4

Table 2.3: $U_1(C)$ and $U_2(C)$, from Figure 2.1, as values

U	c_1	c_2
U_1	$1v_1 + 0v_2 + 0v_3 + 0v_4$	$0v_1 + 1v_2 + 0v_3 + 0v_4$

U	c_1	c_2
U_2	$0v_1 + 0v_2 + 1v_3 + 0v_4$	$0v_1 + 0v_2 + 0v_3 + 1v_4$

Table 2.4: Unique utility values from Figure 2.1

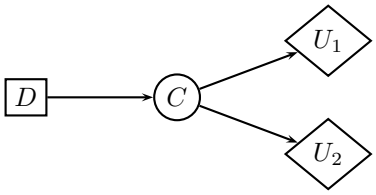


Figure 2.1: Example influence diagram

When using *FLUF* it is assumed that a series of observations have been made. These observations describe a sequence of variables observed and decisions made by the agent. Each observation contains an instantiation of $I_0, I_1 \ldots I_{n-1}$ and $D_1, D_2 \ldots D_n$. The reason I_n is not included is that the variables in I_n are observed after the last decision, if at all. These series of observations fall into two categories: fully observed strategies and partially observed strategies.

2.3 Fully Observed Strategies

Assuming that the observed strategy is fully observed, then the following method can be used to determine boundaries for the utilities.

Given some instantiation of D_k for some $past(D_k$, where $past(D_k)$ is the relevant past of decision node D_k, the observed choice in D_k for that particular configuration of the relevant past is denoted $\delta_{D_k}(past(D_k))$. The entire past of D_k, i.e. all nodes prior to D_k in the temporal order, is denoted $epast(D_k)$. If D_k is the last decision node, the expected utility of the observed choice can be calculated by Equation 2.3.

$$\rho_{D_k}\big(\delta_{D_k}(past(D_k)), epast(D_k)\big) = \sum_{I_k} P\big(I_k | epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot V\big(pa(U)\big)$$
$$= \sum_{I_k \cap pa(U)} P\big(I_k \cap pa(U) | epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot V\big(pa(U)\big) \tag{2.3}$$

It is not necessary to maximize D_k as the observed choice is known. ρ_{D_k} denotes expected utility of the choice and entire past received as arguments.

Now, by working backward from the last decision node, D_n, it is possible to determine a set of constraints $\mathcal{C}$. These are constraints on the utility values, and given these constraints the utilities can be estimated. The constraints span a utility space over the utilities. Any combination of utilities in this space could explain the observed strategy.

The overall method is to look at the last decision node and, based on the observations, determine what choice was made for each of its possible configuration of the relevant past. It is possible to determine the choice for each past configuration because it is a fully observed strategy. For each of the observed decisions, the expected utility of the observed choice must be greater than or equal to the expected utility of the alternative choices, as the agent is rational.

To establish constraints, let $\mathbf{O}$ denote the set of all the observations. o is a single observation in $\mathbf{O}$ on the form: $i_0, d_1, i_1, \ldots, i_{n-1}, d_n$. o_k is then the observations of $i_0, d_1, \ldots, i_{k-1}, d_{k-1}$ in o.

First let $k = n$, then for each observation $o \in \mathbf{O}$ add the following constraints to $\mathcal{C}$:

$$\forall_{d \in D_k \setminus \delta_{D_k}(o_k)} : \rho_{D_k}(\delta_{D_k}(o_k), o_k) > \rho_{D_k}(d, o_k) \tag{2.4}$$

After having added the constraints for all the observations, then replace the last decision node (D_k) with a chance node where the probability for the observed choice ($\delta_{D_k}(o_k)$) is one and the probability for the alternative choices, given the same relevant past configuration, are zero. This means that the chance node replacing D_k must have all nodes in the relevant past of D_k as parents. Then apply Equation 2.4 for $k-1$ and continue until $k = 0$.

The constraints are described as strict inequalities, since any strategy could have been explained by the trivial utility function, where all utilities from the same utility nodes are equal, if the inequalities had not been strict. This is because the set of points describing the trivial utility function define a part of the utility space, called *the diagonal*[1], that constraints can at most be tangents to. Appendix A presents a proof that constraints created based on fully observed strategies will always be tangents to this diagonal.

2.4 Partially Observed Strategies

This section introduces the calculations needed for generating constraints when the strategy is only partially observed. When the observed strategy is only partially observed it is not possible to use the same method as for fully observed strategies. If one of the decision nodes, say D_k, is not observed for a particular configuration of the relevant past, which is the case by definition when the strategy is only partially observed, then to calculate the expected utility for the decision node prior to D_k, the following equation would be used:

[1] This set of points is called the diagonal because it includes the line from $(0, 0, \ldots, 0)$ to $(1, 1, 1, \ldots, 1)$

$$\rho_{D_{k-1}}(\delta_{D_{k-1}}(past(D_{k-1})), epast(D_{k-1})) = \sum_{I_{k-1}} P\big(I_{k-1}|epast(D_{k-1}), \delta_{D_{k-1}}(past(D_{k-1}))\big) \cdot \sum_{d \in D_k} P_{\delta_{D_k}}(d|past(D_k)) \sum_{I_k} P\big(I_k|d, epast(D_k)\big) \cdot V(pa(U))$$

where $P_{\delta_{D_k}}$ is the probability function that replaces decision node D_k with a chance node where the probability for the observed choice is one and all others are zero. Unfortunately $P_{\delta_{D_k}}(d|past(D_k))$ is not known as decision node D_k is not observed given the past $past(D_k)$.

Alternatively the method could be based on the assumption that the choice in the decision node D_k is the one that maximizes the expected utility, which is reasonable as the agent is rational. Under this assumption Equation 2.5 calculates maximum expected utility for decision node D_k given its past.

$$\rho_{D_k}(epast(D_k)) = \max_{D_k} \sum_{I_k} P(I_k|epast(D_k), D_k) \cdot \rho_{D_{k+1}}(epast(D_{k+1})) \tag{2.5}$$

However, Equation 2.5 is not linear, because of the maximization, which makes it infeasible for determining the constraints.

The problem of making ρ linear can be solved by relaxing the constraints. For each constraint an upper bound and a lower bound is created like done by Chajewska et al. (2001). These bounds are constructed so that the upper bound is always larger than the expected utility for the node the constraint is derived from, and the lower bound is always less than the expected utility. The upper bound is created so that the for each utility value the decision is assumed to be made so that it maximize this utility value. This means that the probabilities that the utility value have to multiplied with (see Lemma 2.1) are as large as possible. As this is done individually for the utility value the coefficients for each utility value will always be as large or larger than the corresponding coefficient when using Lemma 2.1. The opposite holds when calculating lower bound.

Formulae for Partially Observed Strategies

In order to be able to describe the method for developing constraints these upper and lower bounds, a series of equations are necessary.

In the following $\overline{\rho}$ denotes the calculation of the upper bound and $\underline{\rho}$ the lower bound. These bounds are calculated for each configuration of the relevant past. Whether the decision node is observed for that relevant past, determines which equation is used.

If the decision node is observed in the configuration of its relevant past in $epast(D_k)$ and is the last node in the temporal order, the equation is:

$$\begin{aligned}\overline{\rho}_{D_k}(epast(D_k), \delta_{D_k}(past(D_k)) &= \underline{\rho}_{D_k}(epast(D_k), \delta_{D_k}(past(D_k)) \\ &= \sum_{I_k} P\big(I_k|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot V(pa(U)) \\ &= \sum_{I_k \cap pa(U)} P\big(I_k \cap pa(U)|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot V(pa(U))\end{aligned} \tag{2.6}$$

In Equation 2.6 it is not necessary to maximize the expected utility for the decision node D_k as it has been observed. Note that Equation 2.6 is the same as Lemma 2.1 where $k = n$, meaning that $\overline{\rho} = \underline{\rho} = \rho$ when the decision node is the last decision node in the temporal order.

If the decision node is the last node in the temporal order and is unobserved given the relevant past in $epast(D_k)$ and there is l utility nodes in the domain, the bounds are calculated as:

$$\overline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m}\left(\max_{D_k}\left(\sum_{I_k \cap pa(U)} P(I_k \cap pa(U)|epast(D_k), D_k) \cdot \sum_{j=1}^{l}(\alpha_{j,i,pa(U_j)})\right) \cdot v_i\right) \quad (2.7)$$

$$\underline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m}\left(\min_{D_k}\left(\sum_{I_k \cap pa(U)} P(I_k \cap pa(U)|epast(D_k), D_k) \cdot \sum_{j=1}^{l}(\alpha_{j,i,pa(U_j)})\right) \cdot v_i\right) \quad (2.8)$$

An important aspect of Equation 2.7 and 2.8 is that the components under the maximation are probabilities and coefficients, which both are constants. The utility variables (v_i), the only variables in the equations, are outside the maximation, so the equations are linear in the utility values.

If the node is observed given the relevant past in $epast(D_k)$ and it is not the last decision node in the temporal order, the bounds are calculated according to the equations:

$$\overline{\rho}_{D_k}(epast(D_k), \delta_{D_k}(past(D_k)) = \sum_{I_k} P\big(I_k|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot \overline{\rho}_{D_{k+1}}\big(epast(D_{k+1})\big) \quad (2.9)$$

$$\underline{\rho}_{D_k}(epast(D_k), \delta_{D_k}(past(D_k)) = \sum_{I_k} P\big(I_k|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot \underline{\rho}_{D_{k+1}}\big(epast(D_{k+1})\big) \quad (2.10)$$

If the decision node is unobserved in the configurations of its relevant past in $epast(D_k)$, it is still necessary to calculate the bounds, as Equation 2.9 and 2.10 are defined recursively.

For defining the equations that describe how to calculate the bounds for unobserved nodes, the definition for ρ_{D_k} has to be extended. ρ_{D_k} can be described as: $\rho_{D_k}(epast(D_k)) = \rho_{D_k,1}(epast(D_k))v_1 + \rho_{D_k,2}(epast(D_k))v_2 \ldots$ where $\rho_{D_k,i}(epast(D_k))$ is the coefficient of v_i in $\rho_{D_k}(epast(D_k)$. The bounds for unobserved decision nodes given the past $epast(D_k)$ prior to the last node is calculated as follows.

$$\overline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m}\left(\max_{D_k}\left(\sum_{I_k} P(I_k|epast(D_k), D_k) \cdot \overline{\rho}_{D_{k+1},i}(epast(D_{k+1}))\right) \cdot v_i\right) \quad (2.11)$$

$$\underline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m}\left(\min_{D_k}\left(\sum_{I_k} P(I_k|epast(D_k), D_k) \cdot \underline{\rho}_{D_{k+1},i}(epast(D_{k+1}))\right) \cdot v_i\right) \quad (2.12)$$

Note again that the part of the equations that is being maximized, does not contain any utility variables meaning that the equations are linear in the utilities.

With these equations it is possible to find the constraints.

First let $k = n$, then for each observation $o \in \mathbf{O}$: If decision node D_k is observed in o add the constraints:

$$\forall_{d \in D_k \setminus \delta_{D_k}(o_k)} : \overline{\rho}(\delta_{D_k}(o_k), o_k) > \underline{\rho}(d, o_k)$$

Then decrease k by one and iterate through the observations again, until k reaches zero meaning that all decision nodes have been evaluated.

2.5 Choosing Utility Values

This section introduces the method for choosing the utility values to be used in *FLUF*. When the feasible space has been established some utility values have to be chosen. Any point within the feasible space is considered valid. In *FLUF* the selected utility values are defined as the coordinates of the center of the largest possible hypersphere in the feasible space. The overall strategy is to choose a point that is within the constraints so that the radius of the sphere from that center is as large as possible. This is expected to be a fairly good strategy as the constraints span more utility values than those that can explain the observed behavior. The invalid utility values will likely be near the constraints as the constraints are relaxed, meaning that utility values near the constraints might be in the feasible space only as a result of this relaxation. So the further from the constraints, the utility values are chosen, the more observations are expected to be explained, as long they are in the feasible space.

Let the set of constraints be $c_1(\mathbf{v}), c_2(\mathbf{v}), \ldots, c_z(\mathbf{v})$ and each constraint be defined as $c_i(\mathbf{v}) \equiv c_{i,1}v_1 + c_{i,2}v_2 + \cdots + c_{i,m}v_m > 0$ where each $c_{i,t}$ is a constant. A new set of linear inequalities is defined from the constraints as follow:

$$d(\mathbf{p}, c_k(\mathbf{v})) \geq r$$

where r is a new variable expressing the radius of the sphere, and $d(\mathbf{p}, c_k(\mathbf{v}))$ is the distance from the point $\mathbf{p}$ to the hyperplane described by $c_k(\mathbf{v})$. d is calculated as:

$$d(\mathbf{p}, c_k(\mathbf{v})) = \frac{c_{k,1}p_1 + c_{k,2}p_2 + \cdots + c_{k,n}p_n}{\sqrt{c_{k,1}^2 + c_{k,2}^2 + \cdots + c_{k,n}^2}}$$

This means that the linear inequality for each constraint is as follows:

$$\frac{c_{k,1}p_1 + c_{k,2}p_2 + \cdots + c_{k,n}p_n}{\sqrt{c_{k,1}^2 + p_{k,2}^2 + \cdots + c_{k,n}^2}} \geq r$$

$$\Updownarrow$$

$$c_{k,1}p_1 + c_{k,2}p_2 + \cdots + c_{k,n}p_n \geq r \cdot \sqrt{c_{k,1}^2 + c_{k,2}^2 + \cdots + c_{k,n}^2}$$

$$\Updownarrow$$

$$c_{k,1}p_1 + c_{k,2}p_2 + \cdots + c_{k,n}p_n - r \cdot \sqrt{c_{k,1}^2 + c_{k,2}^2 + \cdots + c_{k,n}^2} \geq 0 \tag{2.13}$$

Besides the inequalities for the constraints, inequalities have to be added to ensure that the center of the hypersphere is within the normalized region.

Each of the coordinates of the center must be above 0. So for each dimension t, i.e. $t \in [1; n]$, the following inequality is added:

$$\begin{aligned} & p_t & \geq r \\ \Updownarrow & & \\ & p_t - r & \geq 0 \end{aligned} \tag{2.14}$$

Similarly each of the coordinates of the center must be less than one, for the each dimension in the feasible space.

$$\begin{aligned} & p_t + r & \leq 1 \\ \Updownarrow & & \\ & p_t + r - 1 & \leq 0 \end{aligned} \tag{2.15}$$

Then r is maximized in accordance with the inequalities from Equation 2.13, 2.14 and 2.15. This can be solved as a linear programming task (Fraleigh and Beauregard, 2003).

The centroid $(p_0, p_1, \ldots, p_n)$ is then a point within the feasible space. As each dimension in the feasible space corresponded to a utility, the values of the utilities are set to the corresponding coordinate of the centroid. The point chosen is also called the *utility point*.

2.6 Comparison of *FLUF* and Chajewska et al. (2001)

As mentioned *FLUF* is based on the method proposed by Chajewska et al. (2001), so in this section these two methods will be compared. This will include comparison in both structure and in complexity.

2.6.1 Structure

The immediate structural difference between *FLUF* and the method presented in Chajewska et al. (2001) is that *FLUF* operates on influence diagrams whereas Chajewska et al. (2001) operates on decision trees. As already argued in Hansen et al. (2004) any symmetric decision tree can be described as an influence diagram, and any decision tree can be made symmetric by inserting additional artificial nodes. These artificial nodes must have the same state space as the corresponding nodes in the alternative branches. The reverse process from influence diagram to decision tree is also possible, so the representation of the domain does not affect when either method can be used. As there is no decision scenario where influence diagrams can be used while decision trees cannot, or vice versa, the remainder of this analysis will assume that both are given. If that should not be the case, the described transformation might be necessary which is a non trivial procedure.

Chajewska et al. (2001) assumes that each utility node comprises a series of linear additive subutilities. Each of these subutilities are assumed to contribute to all outcomes by some weight (zero if they do not contribute at all). When comparing decision trees and influence diagrams, each branch of the decision tree equals one instantiation of the entire influence diagram, and vice versa. This means that when comparing the utility nodes of an influence diagram with the utility nodes of a decision tree, all of the utility nodes in the influence diagram contribute to each of the utility nodes in the decision tree. Each of the possible outcomes of each utility node

in the influence diagram can be considered a single subutility in the decision tree. A significant number of the weights for the subutilities will be zero, as for each branch in the decision tree only a single utility value from each utility node in the influence diagram can contribute. This matches how *FLUF* considers the utility nodes in influence diagrams.

In the method presented by Chajewska et al. (2001) the expected utility is calculated for each node in the decision tree including the chance nodes, whereas *FLUF* only calculates it for the decision nodes. *FLUF* instead incorporates the chance nodes in calculation of the expected utility for each decision node. It is fully possible to only calculate the expected utility for the decision nodes in the decision tree as well. Considering the formula for calculating the expected utility for an unobserved decision node in a decision tree. ($\overline{\hat{V}}_n[\mathbf{v}]$ is the upper bound for expected utility in node n in a decision tree, and $S(n)$ is the set of successor nodes of node n. This notation is presented in Chapter 2 and Chapter 3 in Hansen et al. (2004))

$$\overline{\hat{V}}_n[\mathbf{v}] = \sum_{i=1}^{m} \max_{\alpha_{n',i}:n' \in S(n)} (\alpha_{n',i} \cdot v_i) \tag{2.16}$$

The expression being maximized $(\alpha_{n',i} \cdot v_i)$ can be replaced with the formula for calculating the expected utility for chance nodes, shown in Equation 2.17.

$$\overline{\hat{V}}_n[\mathbf{v}] = \sum_{n' \in S(n)} p_{n'} \overline{\hat{V}}_{n'}[\mathbf{v}] \tag{2.17}$$

If the decision node, n, is not followed by chance nodes in some or all of the edges leading out, the corresponding successors could, during calculations, be treated as if there actually was a chance node between them and n, with only one state. This would not increase complexity, as the corresponding $n' \in S(n)$ would only describe one element in the inserted chance node. As Equation 2.17 shows, $\overline{\hat{V}}_n[\mathbf{v}]$ generates a mean of the successors if n is a chance node, so to use Equation 2.17 in the scope of decision nodes, the notation of $\overline{\hat{V}}_n[\mathbf{v}]$ is expanded to $\overline{\hat{V}}_n[\mathbf{v}] = \sum_{i=1}^{m} (\overline{\hat{V}}_{n,i}[\mathbf{v}] \cdot v_i)$, meaning that $\overline{\hat{V}}_{n,i}[\mathbf{v}]$ is the coefficient attached to utility number i in node n. Now the result of merging Equation 2.16 and 2.17 is shown in Equation 2.18.

$$\overline{\hat{V}}_n[\mathbf{v}] = \sum_{i=1}^{m} \left(\max_{n' \in S(n)} \left(\sum_{n'' \in S(n')} p_{n''} \cdot \overline{\hat{V}}_{n'',i}[\mathbf{v}] \right) \cdot v_i \right) \tag{2.18}$$

Comparing Equation 2.18 with the formula used by *FLUF* (Equation 2.19) the similarities between the two methods can be seen. Equation 2.19 uses ρ to denote the expected utility whereas Equation 2.18 uses $\hat{V}$.

$$\overline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m} \left(\max_{D_k} \left(\sum_{I_k} P(I_k | epast(D_k), D_k) \cdot \overline{\rho}_{D_{k+1},i}(epast(D_{k+1})) \right) \cdot v_i \right) \tag{2.19}$$

Note that Equation 2.18 does not explicitly have to take the past into consideration, as it is represented by the node's position in the tree.

When choosing the utility function, *FLUF* and Chajewska et al. (2001) uses completely different methods. *FLUF* finds the utility values that are as far away from the created constraints

as possible. Chajewska et al. (2001) assumes that a probability distribution over utility functions is given. This is then used to find a utility function within the feasible space that has a high likelihood. The exact difference between the utility function chosen by *FLUF* and the one chosen by Chajewska et al. (2001) depends on the supplied distribution over the utility functions, and no general conclusions are drawn. The difference between the two methods for choosing utility functions, do not only affect the utility function, but also makes the prerequisites for using the methods different as *FLUF* does not require a prior knowledge about the utility function.

In conclusion, *FLUF* and Chajewska et al. (2001) creates the same feasible space given that the subutilities in Chajewska et al. (2001) is represented as described here. This is no coincidence as the underlying work for how *FLUF* creates its constraints, is the work presented in Chajewska et al. (2001). The most significant structural difference between the methods is how the utility function is chosen within the feasible space.

2.6.2 Complexity

When solving decision trees and influence diagrams, both representations have a worst case time complexity that is $\mathcal{O}(nodes^{states})$ where *nodes* is the number of decision and chance nodes in the domain for influence diagrams, and the depth of the tree for decision trees. *states* is the largest state space of any node in the domain. However, the worst case time complexities of the methods (Chajewska et al. (2001) and *FLUF*) are $\mathcal{O}(nodes^{states} \cdot utilities)$, where *utilities* is the number of (sub)utilities in the domain. This is because, as Equations 2.18 and 2.19 show, coefficients must be calculated for every (sub)utility. Had the method for decision trees not been rewritten to Equation 2.18, the same calculations would still have to be made, though in a different order.

However, even though the time complexities of the two methods are identical, the complexities of the models are not the same. The number of nodes in a decision tree grows exponentially in the depth of the tree, as each potential future for a node will have to be modeled. More precisely the complexity is $\mathcal{O}(states^{nodes})$. This means that if the decision scenario always involves the same decisions and the same uncertainties in the same order, the tree will be symmetric where each decision and each uncertainty is represented once for each configuration of the past.

When the decision scenario always involves the same decisions and uncertainties in the same order, influence diagrams have a lower complexity, with regard to the number of nodes, than decision trees. The reason being that the configuration of the past is not represented in the qualitative part of the influence diagram. The number of nodes is constant, no matter the size of the state spaces. However, the size of the needed probability table is always a fixed size based on the state space, so if the past in the decision scenario can make a decision or uncertainty irrelevant, the influence diagram still have to model all outcomes, even those that will no impact on the utilities.

In conclusion, if the relevant pasts of the decision nodes are large, meaning they contain a large number of nodes, and that there are many configurations of the relevant pasts that have probability zero, decision trees will have a lower complexity with regard to number of nodes. If the relevant pasts of the decisions are small and there are few configurations with zero probability influence diagrams will have a lower complexity.

2.7 Limitations of *FLUF*

In this section an analysis of the learning method used by *FLUF* is presented, highlighting the steps in *FLUF* in which the inaccuracies occur.

FLUF analyzes which utility values in an influence diagram that could describe the observed behavior of an agent. Basically it tries to define a feasible space for the utility values, as more than one set of values could describe the observed behavior. But the complexity of finding the feasible space of utility values that exactly describes the observed behavior makes it impractical. Therefore *FLUF* uses a less complex method that finds a different space, which is a super space of the exact feasible space, and chooses its utility values from this super space. This relaxation decreases the accuracy with which *FLUF* can estimate the utility values of the observed agent. It may result in *FLUF* not being able to determine a utility function that could explain the behavior already observed, even if the agent's utility values never change. A method that defines the constraints as exact as possible will be denoted the *optimal method*. In the following section *FLUF* will be compared to the optimal method in order to understand where the inaccuracies in *FLUF* occur.

2.7.1 The Optimal Method

The feasible space spanned by a constraint created using *FLUF* is larger than the feasible space that could explain the observed behavior. For an observed decision ($D_k(o_k)$), the calculations of ρ (the expected utility) is done as if the subsequent unobserved decisions are in the state that gives the largest possible coefficient for each utility value.

When *FLUF* determines constraints, a lower bound for the expected utility for the decisions that was not selected is also determined. When calculating this, the unobserved decisions are assumed being in the state that give the smallest possible coefficients for determining ρ. This ensures that the utility space describing the observed agents strategy is a sub space of the space spanned by the created constraint.

The most exact expected utility that could be defined for an unobserved decision node, is using the formula for calculating the expected utility:

$$\rho_{D_k}(epast(D_k)) = \begin{cases} \max\limits_{D_k} \sum_{I_k} P(I_k|epast(D_k), D_k) \cdot \rho_{D_{k+1}}(epast(D_{k+1})) & k \neq n \\ \max\limits_{D_k} \sum_{I_k} P(I_k|epast(D_k), D_k) \cdot V(pa(U)) & k = n \end{cases}$$

This formula can be used in conjunction with the following formula, for calculation of ρ when the configuration of the relevant past of D_k in *epast* has been observed to be $\delta_{D_k}(past(D_k))$.

$$\rho_{D_k}(epast(D_k), \delta_{D_k}(past(D_k))) = \begin{cases} \sum_{I_k} P\big(I_k|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot \rho_{D_{k+1}}\big(epast(D_{k+1})\big) & k \neq n \\ \sum_{I_k} P\big(I_k|epast(D_k), \delta_{D_k}(past(D_k))\big) \cdot V(pa(U)) & k = n \end{cases}$$

With those two formulas the constraints for each observation can be added. The constraints for each observed decision are created as:

$$\forall_{d \in D_k \setminus \delta_{D_k}(o_k)} : \rho_{D_k}(\delta_{D_k}(o_k), o_k) \geq \rho_{D_k}(d, o_k)$$

The feasible space spanned by the constraints of the optimal method is denoted the *true feasible space* or the *true utility space*.

For an unobserved decision node D_k, in some configuration of its relevant past, the normalized region can be divided into subregions, one for each possible decision in D_k. For all utility points within each subregion, the corresponding decision of D_k will yield the maximum expected. If all the different relevant pasts of D_k are considered in this way, then each relevant past will divide the normalized region in a set of subregions, each of these corresponding to a specific decision for that relevant past. If a point is chosen in the normalized region, then by examining which subregions it is in, then a complete policy for node D_k can be described.

When creating constraints using the optimal method, for some decision in a node, D_p, that preceeds D_k ($p < k$), then a constraint for each possible policy, conforming with the behavior observed so far, is created. These constraints are calculated as if the policy had been observed, meaning they can be created as for fully observed strategies, Section 2.3. However, each constraint is only valid in the subregion corresponding to the policy used to create it, so the space spanned by the constraint is intersected with this subregion. Now the spaces spanned within each subregion describe the utility points that can explain the observed decision in D_p, when the policy in D_k is the one associated with that subregion.

The total space that can describe the observed D_p is the union of all the spanned spaces in the subregions. The reason all the subregions have to be included is that the subregion that the utilities are actually in is unknown.

Exactly which utility point that is chosen within the true utility space is irrelevant as they would all described the observed behavior. So to choose the utility point as the center of the largest possible hypersphere of the true utility space could still be done. It should be noted that finding this is significantly more complex than for *FLUF*, as the space is no longer spanned by linear constraints.

Now, the only difference between this optimal method and *FLUF* is the way the feasible space is determined, so this will be examined closer. Only unobserved decision nodes are examined, as the calculations in observed nodes are semantically identical. Using a notation that is like the one used to describe *FLUF* the optimal method would use Equation 2.20 (Note that ρ for the optimal method is neither overlined nor underlined).

$$\begin{aligned}\rho_{D_k}(epast(D_k)) &= \max_{D_k}\left(\sum_{I_k} P(I_k|epast(D_k),D_k)\rho_{D_{k+1}}(epast(D_{k+1}))\right)\\ &= \max_{D_k}\left(\sum_{I_k} P(I_k|epast(D_k),D_k)\cdot\sum_{i=1}^{m}\left(\rho_{D_{k+1},i}(epast(D_{k+1}))\cdot v_i\right)\right)\\ &= \max_{D_k}\left(\sum_{i=1}^{m}\left(\sum_{I_k} P(I_k|epast(D_k),D_k)\cdot\rho_{D_{k+1},i}(epast(D_{k+1}))\right)\cdot v_i\right)\end{aligned} \tag{2.20}$$

This calculation is done for each subregion of D_k, such that $\max_{D_k}$ yields a different decision in all these calculations. However, $\rho_{D_{k+1},i}(epast(D_{k+1}))$ represents the expected utility of strategy followed by the decision nodes following D_k in the temporal order. This means that for each possible choice in D_k this calculation has to be done once for each set of policies for subsequent decision nodes, consistent with all observations made and the chosen decision in D_k.

When *FLUF* maximizes the coefficients for utility values it uses Equation 2.21.

$$\overline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m}\left(\max_{D_k}\left(\sum_{I_k} P(I_k|epast(D_k),D_k)\cdot\overline{\rho}_{D_{k+1},i}(epast(D_{k+1}))\right)\cdot v_i\right) \tag{2.21}$$

Comparing Equation 2.20 and 2.21 it can be seen that the difference is in the order in which the decision is made and when the utility values are added. As *sum* and *max* are not commutative, these are not equivalent. In fact it will always hold that, no matter which decision is chosen for ρ_{D_k}:

$$\overline{\rho}_{D_k} \geq \rho_{D_k} \tag{2.22}$$

The opposite argument can be made for how *FLUF* minimizes the coefficients for the utility values:

$$\underline{\rho}_{D_k}(epast(D_k)) = \sum_{i=1}^{m} \left(\min_{D_k} \left(\sum_{I_k} P(I_k|epast(D_k), D_k) \cdot \underline{\rho}_{D_{k+1},i}(epast(D_{k+1})) \right) \cdot v_i \right) \tag{2.23}$$

Meaning that it will always hold that:

$$\underline{\rho}_{D_k} \leq \rho_{D_k} \tag{2.24}$$

Basically, the constraints created by *FLUF* are of the form (in *FLUF* strict inequalities are used, but that does not affect this discussion):

$$\forall_{d \in D_k} : \overline{\rho}_{D_k=\delta_{D_k}}(epast(D_k)) - \underline{\rho}_{D_k=d}(epast(D_k)) \geq 0 \tag{2.25}$$

Where δ_{D_k} is the observed choice in decision D_k. Using the optimal method the constraints are of the form:

$$\forall_{d \in D_k} : \rho_{D_k=\delta_{D_k}}(epast(D_k)) - \rho_{D_k=d}(epast(D_k)) \geq 0 \tag{2.26}$$

From the established relationships in Equations 2.22 and 2.24, used by *FLUF* and the optimal method, it can be concluded that the feasible space spanned by the constraints in *FLUF* will always be a super space of the corresponding constraints in the optimal method.

Another way of comparing the feasible space of the optimal method and *FLUF*, is to look at how many of the observations the methods would be able to explain after having observed them. In other words, examining whether the constraints created during observations will be descriptive enough, for the method to predict the decisions correctly if one of the observations were repeated. An experiment was conducted using the static domain from the experiments in Hansen et al. (2004). It was determined how many of the already observed training cases could be predicted by the method every time a new observation is added. The optimal method has not been implemented but, theoretically, its chosen utility point would be able to explain all observed decisions. So the number of training cases that *FLUF* is unable to predict correctly, indicates how it performs compared to the optimal method, and thereby expresses how much have been lost in the approximation being done by *FLUF*.

The values shown in Figure 2.2 is the average of training cases predicted correctly (that is both decisions are correct). This average is based on 30 runs with 200 observations. As can be seen in Figure 2.2 *FLUF* is able to predict 80% of the observed training cases correctly, on average, after only one case. The prediction by *FLUF* stabilizes at that level after 20 cases.

As about 20% of the observations are not correctly predicted by *FLUF* it indicates that the utility point is wrong. This means that the feasible space of *FLUF* must be larger than the feasible space of the optimal method, as the optimal method would be able to predict the observations from any utility point in its feasible space.

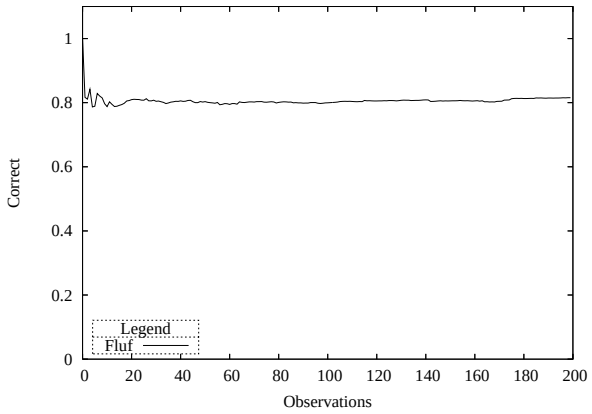


Figure 2.2: Training cases predicted correctly

$P(C_1\|D_1)$	$D_1 = d_1^1$	$D_1 = d_1^2$
c_1^1	0.3	0.75
c_1^2	0.7	0.25

Table 2.5: $P(C_1|D_1)$ for Figure 2.3

$P(C_2\|C_1, D_2)$	$D_2 = d_2^1$	$D_2 = d_2^2$
c_1^1	(0.4, 0.2, 0.4)	(0.1, 0.4, 0.5)
c_1^2	(0.1, 0.7, 0.2)	(0.3, 0.1, 0.6)

Table 2.6: $P(C_2|C_1, D_2)$ for Figure 2.3

Example 2.7.1 In order to illustrate the inaccuracies discussed, an example is given of how calculations would be made. The influence diagram used in this example is shown in Figure 2.3, and the probability tables are shown in Table 2.5 and 2.6. Both decision nodes are binary.

The following observation is made:

$$t = \langle D_1 = d_1^2, C_1 = c_1^1, D_2 = d_2^2 \rangle$$

Using *FLUF* the upper and lower bounds for the decisions are calculated. First for D_2.

$$\begin{aligned}
\overline{\rho}_{D_2}(C_1 = c_1^1, D_2 = d_2^2) &= P(c_2^1|c_1^1, d_2^2)V(c_2^1) + P(c_2^2|c_1^1, d_2^2)V(c_2^2) + P(c_2^3|c_1^1, d_2^2)V(c_2^3) \\
&= 0.1(v_1 + 0v_2 + 0v_3) + 0.4(0v_1 + v_2 + 0v_3) + 0.5(0v_1 + 0v_2 + v_3) \\
&= 0.1v_1 + 0.4v_2 + 0.5v_3 \\
\underline{\rho}_{D_2}(C_1 = c_1^1, D_2 = d_2^1) &= P(c_2^1|c_1^1, d_2^1)V(c_2^1) + P(c_2^2|c_1^1, d_2^1)V(c_2^2) + P(c_2^3|c_1^1, d_2^1)V(c_2^3) \\
&= 0.4(v_1 + 0v_2 + 0v_3) + 0.2(0v_1 + v_2 + 0v_3) + 0.4(0v_1 + 0v_2 + v_3) \\
&= 0.4v_1 + 0.2v_2 + 0.4v_3
\end{aligned}$$

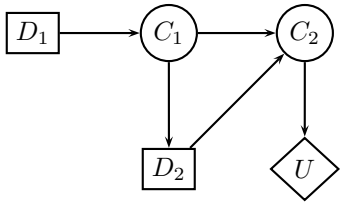


Figure 2.3: Example of an influence diagram

This gives the constraint:

$$\begin{aligned}
&\overline{\rho}_{D_2}(C_1 = c_1^1, D_2 = d_2^2) > \underline{\rho}_{D_2}(C_1 = c_1^1, D_2 = d_2^1) \\
&0.1v_1 + 0.4v_2 + 0.5v_3 > 0.4v_1 + 0.2v_2 + 0.4v_3 \\
&-0.3v_1 + 0.2v_2 + 0.1v_3 > 0
\end{aligned} \tag{2.27}$$

A constraint for D_1 is also be created. Again the upper and lower bounds have to be calculated.

$$\begin{aligned}
\overline{\rho}_{D_1}(D_1 = d_1^2) &= P(c_1^1|d_1^2)\overline{\rho}_{D_2}(c_1^1) + P(c_1^2|d_1^2)\overline{\rho}_{D_2}(c_1^2) \\
&= 0.75(0.1v_1 + 0.4v_2 + 0.5v_3) + 0.25\Big(\\
&\quad \max_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,1,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,1,c_2^2} + P(c_2^3|c_1^2, d_2)\alpha_{U,1,c_2^3}\right)v_1 + \\
&\quad \max_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,2,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,2,c_2^2} + P(c_2^3|c_1^2, d_2)\alpha_{U,2,c_2^3}\right)v_2 + \\
&\quad \max_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,3,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,3,c_2^2} + P(c_2^3|c_1^2, d_2)\alpha_{U,3,c_2^3}\right)v_3\Big) \\
&= 0.075v_1 + 0.3v_2 + 0.375_3 + 0.25\big((0.3 \cdot 1 + 0.1 \cdot 0 + 0.6 \cdot 0)\, v_1 + \\
&\quad (0.1 \cdot 0 + 0.7 \cdot 1 + 0.2 \cdot 0)\, v_2 + (0.3 \cdot 0 + 0.1 \cdot 0 + 0.6 \cdot 1)\, v_3\big) \\
&= 0.075v_1 + 0.3v_2 + 0.375v_3 + 0.25(0.3v_1 + 0.7v_2 + 0.6v_3) \\
&= 0.15v_1 + 0.475v_2 + 0.525v_3
\end{aligned}$$

$$\begin{aligned}
\underline{\rho}_{D_1}(D_1 = d_1^1) &= P(c_1^1|d_1^1)\underline{\rho}_{D_2}(c_1^1) + P(c_1^2|d_1^1)\underline{\rho}_{D_2}(c_1^2) \\
&= 0.3(0.4v_1 + 0.2v_2 + 0.4v_3) + 0.7\Big(\\
&\quad \min_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,1,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,1,c_2^2} + P(c_2^2|c_1^2, d_2)\alpha_{U,1,c_2^3}\right)v_1 + \\
&\quad \min_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,2,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,2,c_2^2} + P(c_2^2|c_1^2, d_2)\alpha_{U,2,c_2^3}\right)v_2 + \\
&\quad \min_{D_2}\left(P(c_2^1|c_1^2, d_2)\alpha_{U,3,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,3,c_2^1} + P(c_2^2|c_1^2, d_2)\alpha_{U,3,c_2^3}\right)v_3\Big) \\
&= 0.12v_1 + 0.06v_2 + 0.12v_3 + 0.7\big((0.1 \cdot 1 + 0.7 \cdot 0 + 0.2 \cdot 0)\, v_1 + \\
&\quad (0.3 \cdot 0 + 0.1 \cdot 1 + 0.6 \cdot 0)\, v_2 + (0.1 \cdot 0 + 0.7 \cdot 0 + 0.2 \cdot 1)\, v_3\big) \\
&= 0.12v_1 + 0.06v_2 + 0.12v_3 + 0.7\,(0.1v_1 + 0.1v_2 + 0.2v_3) \\
&= 0.22v_1 + 0.16v_2 + 0.32v_3
\end{aligned}$$

This gives the constraint:

$$\begin{aligned}
&\overline{\rho}_{D_1}(D_1 = d_1^2) > \underline{\rho}_{D_1}(D_1 = d_1^1) \\
&0.15v_1 + 0.475v_2 + 0.525v_3 > 0.22v_1 + 0.16v_2 + 0.32v_3 \\
&-0.07v_1 + 0.315v_2 + 0.205v_3 > 0
\end{aligned} \tag{2.28}$$

The constraints generated when using the optimal method will now be calculated, with these it is possible to define a smaller feasible space. The constraint from Equation 2.27 is also

present using the optimal method as no maximization is involved in determining it. However, calculating the constraint for D_1 is done differently.

$$\begin{aligned}\rho_{D_1}(d_1^2) &= P(c_1^1|d_1^2)\rho_{D_2}(c_1^1) + P(c_1^2|d_1^2)\rho_{D_2}(c_1^2) \\ &= 0.75(0.1v_1 + 0.4v_2 + 0.5v_3) + \\ &\quad 0.25(\max_{D_2}(P(c_2^1|c_1^2, d_2)v_1 + P(c_2^2|c_1^2, d_2)v_2 + P(c_2^3|c_1^2, d_2)v_3))\end{aligned}$$

This calculation must be done for each subregion consistent with the observation made. As D_2 has two options and only has one unobserved relevant past, there will only be two subregions in the normalized region for which constraints must be calculated. These subregions are found by comparing expected utilities:

$$\begin{aligned}\rho_{D_2}(C_1 = c_1^2, D_2 = d_2^1) &= P(c_2^1|c_1^2, d_2^1)v_1 + P(c_2^2|c_1^2, d_2^1)v_2 + P(c_2^3|c_1^2, d_2^1)v_3 \\ &= 0.1v_1 + 0.7v_2 + 0.2v_3 \\ \rho_{D_2}(C_1 = c_1^2, D_2 = d_2^2) &= P(c_2^1|c_1^2, d_2^2)v_1 + P(c_2^2|c_1^2, d_2^2)v_2 + P(c_2^3|c_1^2, d_2^2)v_3 \\ &= 0.3v_1 + 0.1v_2 + 0.6v_3\end{aligned}$$

Now it is possible to complete the calculation for $\rho_{D_1}(d_1^2)$ in both subregions, as in each subregion a specific choice of D_2 can be considered optimal when C_1 is in state c_1^2:

$0.1v_1 + 0.7v_2 + 0.2v_3 < 0.3v_1 + 0.1v_2 + 0.6v_3$:

$$\begin{aligned}\rho_{D_1}(d_1^2) &= P(c_1^1|d_1^2)\rho_{D_2}(c_1^1) + P(c_1^2|d_1^2)\rho_{D_2}(c_1^2) \\ &= 0.75(0.1v_1 + 0.4v_2 + 0.5v_3) + 0.25(0.3v_1 + 0.1v_2 + 0.6v_3) \\ &= 0.15v_1 + 0.325v_2 + 0.525v_3\end{aligned}$$

$0.3v_1 + 0.1v_2 + 0.6v_3 < 0.1v_1 + 0.7v_2 + 0.2v_3$:

$$\begin{aligned}\rho_{D_1}(d_1^2) &= P(c_1^1|d_1^2)\rho_{D_2}(c_1^1) + P(c_1|d_1)\rho_{D_2}(c_1^2) \\ &= 0.75(0.1v_1 + 0.4v_2 + 0.5v_3) + 0.25(0.1v_1 + 0.7v_2 + 0.2v_3) \\ &= 0.1v_1 + 0.475v_2 + 0.425v_3\end{aligned}$$

In order to create the constraint it is also necessary to determine $\rho_{D_1}(d_1^1)$:

$0.1v_1 + 0.7v_2 + 0.2v_3 < 0.3v_1 + 0.1v_2 + 0.6v_3$:

$$\begin{aligned}\rho_{D_1}(d_1^1) &= P(c_1^1|d_1^1)\rho_{D_2}(c_1^1) + P(c_1^2|d_1^1)\rho_{D_2}(c_1^2) \\ &= 0.3(0.1v_1 + 0.4v_2 + 0.5v_3) + 0.7(0.3v_1 + 0.1v_2 + 0.6v_3) \\ &= 0.24v_1 + 0.19v_2 + 0.57v_3\end{aligned}$$

$0.3v_1 + 0.1v_2 + 0.6v_3 < 0.1v_1 + 0.7v_2 + 0.2v_3$:

$$\begin{aligned}\rho_{D_1}(d_1^1) &= P(c_1^1|d_1^2)\rho_{D_2}(c_1^1) + P(c_1^2|d_1^2)\rho_{D_2}(c_1^2) \\ &= 0.3(0.1v_1 + 0.4v_2 + 0.5v_3) + 0.7(0.1v_1 + 0.7v_2 + 0.2v_3) \\ &= 0.1v_1 + 0.61v_2 + 0.29v_3\end{aligned}$$

The constraint $\rho_{D_1}(d_1^2) > \rho_{D_1}(d_1^1)$ can now be determined:
$0.1v_1 + 0.7v_2 + 0.2v_3 < 0.3v_1 + 0.1v_2 + 0.6v_3$:

$$\begin{aligned}\rho_{D_1}(d_1^2) &> \rho_{D_1}(d_1^1) \\ 0.15v_1 + 0.325v_2 + 0.525v_3 &> 0.24v_1 + 0.19v_2 + 0.57v_3 \\ -0.09v_1 + 0.135v_2 - 0.045v_3 &> 0\end{aligned} \tag{2.29}$$

As this constraint is only valid when $0.1v_1+0.7v_2+0.2v_3 < 0.3v_1+0.1v_2+0.6v_3$, it is intersected with the constraint $0.2v_1 - 0.6v_2 + 0.4v_3 > 0$. This intersection describes the feasible space in the first subregion.

For the second subregion $0.3v_1 + 0.1v_2 + 0.6v_3 < 0.1v_1 + 0.7v_2 + 0.2v_3$ the constraint is:

$$\begin{aligned}\rho_{D_1}(d_1^2) &> \rho_{D_1}(d_1^1) \\ 0.1v_1 + 0.475v_2 + 0.425v_3 &> 0.1v_1 + 0.61v_2 + 0.29v_3 \\ -0.135v_2 + 0.135v_3 &> 0\end{aligned} \tag{2.30}$$

As this constraint is only valid when $0.3v_1+0.1v_2+0.6v_3 < 0.1v_1+0.7v_2+0.2v_3$, it is intersected with the constraint $-0.2v_1 + 0.6v_2 - 0.4v_3 > 0$. This intersection describes the feasible space in the second subregion.

To describe the feasible space found by the optimal method, the feasible space in the two subregions should first be unioned, and the resulting space should then be intersected with the space described by the constraint in Equation 2.27.

In Figure 2.4 the feasible spaces for both methods are shown (it is the space above all the constraints). As can be seen the space described by the intersection of the green and the blue constraints is larger than the space described by the intersection of the green and the grey constraints, meaning that the optimal method defines a smaller region. The difference between the space spanned by the green plane and the grey planes illustrate the inaccuracy of *FLUF*.

The grid shows the division of the normalized region, where the decision of D_2 that yields the maximum expected utility is different given past c_1^2. In the subregion that contains $(1, 0, 1)$, the decision that yields the maximum expected utility for decision D_2 is d_2^2. In the subregion that contains $(0, 1, 0)$ the decision is d_2^1.

□

Considering Example 2.7.1, it can be seen that even though that the point $(0, 1, 0)$ is within the feasible space of *FLUF*, that point would not explain the observation as the expected utilities would become: $EU(D_1 = d_1^1) = 0.61$ and $EU(D_1 = d_1^2) = 0.475$, meaning that the optimal decision of D_1 would always be d_1^1. *EU* is short for *expected utility*. In general, there is no guarantee that the utility functions expressed by any point in the feasible space spanned by *FLUF*'s constraints can explain the observations made. However, the utility functions

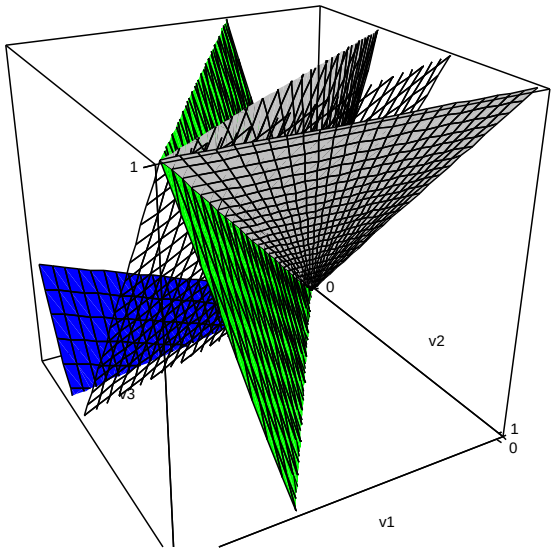


Figure 2.4: The feasible space defined by Equations 2.27(green), 2.28(blue), 2.29 and 2.30(grey). The feasible space is the region above the constraints. It might be hard to see what is above, but all constraints contain the point $(0, 0.75, 1)$. v_1 is from right to left, v_2 is from the back to the front, and v_3 is from the bottom to the top. The grid is the division of the subregions for D_2.

expressed by the space spanned by the optimal method will, by its definition, explain the observations made.

In general it can be concluded that when *FLUF* is incapable of predicting the observations already made, it must be because the utility space created by *FLUF* is larger than the true utility space. This enlargement of the utility space, is a consequence of the fact that the state of decision nodes for unobserved configurations of their relevant past is chosen for each utility value. This is not necessarily the same state for all utility values. If a single state was chosen for these decision nodes the size of the utility space would be smaller, and the amount of utility points that does not conform with the observed behavior will be reduced. This might be a way to achieve better accuracy.

CHAPTER

THREE

Imputing

In this chapter the principle of imputing is introduced. Because as shown in Section 2.7, the feasible space described by *FLUF* is larger than needed, since *FLUF* relaxes constraints when the strategy is partially observed, leading to inaccuracies. Therefore some other method for handling unobserved configurations of relevant pasts could be employed, which does not relax constraints and thereby obtains a smaller feasible space. The concept of imputing observations is introduced for this purpose in this chapter, and two new prediction methods are presented that use imputing techniques to handle partially observed strategies. These methods are named *Utility Iteration* and *Imputing by Comparison*, and will also be referred to as *the imputing methods*.

With a partially observed strategy some configurations of the different relevant pasts in the domain will be observed, while others will be unobserved. The general idea with imputing, is to impute observations with the configurations of the relevant pasts in the domain that are unobserved. Using these imputed observations Utility Iteration and Imputing by Comparison will be able to create constraints, without relaxing them at all, as described for *FLUF* in Section 2.3.

3.1 Imputing Analysis

In this section the consequences of imputing observations is analyzed. Specifically it is analyzed whether imputing could be an improvement over relaxation, and whether imputations can cause conflicts. In the new methods that are suggested in the following sections, the overall idea is to make the observed agent's strategy fully observed by imputing the missing observations. Observations that are imputed have not really been observed, so they are called *virtual observations* , while the observations that have been made will be called *true observations* .

As mentioned the inaccuracy in *FLUF* will stem from relaxing constraints leading to feasible a space that is to large. The reason for developing methods based on fully observed strategies, is that the feasible spaces achieved from such methods will describe smaller spaces, because no relaxation will be necessary. Considering how the constraints are created in *FLUF*:

$$h_1 v_1 + h_2 v_2 + \cdots > l_1 v_1 + l_2 v_2 + \cdots$$

where each h_k is chosen as high as possible and each l_k is chosen as low as possible, both

being coefficients. Assuming the strategy is fully observed, and the constraints are therefore not relaxed, they will be:

$$h'_1 v_1 + h'_2 v_2 + \cdots > l'_1 v_1 + l'_2 v_2 + \cdots$$

where each h'_n and l'_n are the coefficients that matches the chosen decision. As it will always hold that $h_k \geq h'_k$ and $l_k \leq l'_k$ it will also hold that:

$$h_1 v_1 + h_2 v_2 + \cdots \geq h'_1 v_1 + h'_2 v_2 + \cdots \geq l'_1 v_1 + l'_2 v_2 + \cdots \geq l_1 v_1 + l_2 v_2 + \cdots$$

Any space spanned by the constraints derived from a fully observed strategy is a subspace of the space spanned by constraint created in *FLUF*.

Considering the grey constraint in Example 2.7.1 each plane corresponds to different decisions being made in D_2. In general, when imputing an observation, for an unobserved configuration of the relevant past, it results in a constraint being equal to one of the two grey constraints from the optimal method.

3.1.1 Imputations Causing Conflict

Before the methods are introduced, it is discussed how, under certain circumstances, wrong imputations can cause observations to conflict, i.e. the constraints added cause the feasible space to become empty, even though the observed agent never changes its strategy. This analysis is based on the constraints added in the second last decision node (D_{n-1}), in some domain with n decision nodes ($n \geq 2$). For each of the configurations of its relevant past, D_n will yield a (possibly) different expected utility, the decision made in D_{n-1} will influence the likelihood of each of these configurations, and therefore any imputed decisions in D_n will affect the expected utilitiy of decisions in D_{n-1}.

Given a specific combination of imputations in D_n, two constraints can be created in D_{n-1} that together makes the feasible space become empty. For this to be the case the two constraints must coincide, and point in opposite directions, i.e. the constraints would be on the form $c_1 \cdot v_1 \cdot \cdots \cdot c_u \cdot v_u > 0$ and $-c_1 \cdot v_1 \cdot \cdots \cdot -c_u \cdot v_u > 0$ where u is the number of utilities in the domain. Table 3.1 shows the relationship between the coefficients of the constraints.

$Constraint_1$	$Constraint_2$
c_1	$-c_1$
c_2	$-c_2$
$\cdots$	$\cdots$
c_{u-1}	$-c_{u-1}$
$(1 - c_1 \cdots - c_{u-1})$	$-(1 - c_1 \cdots - c_{u-1})$

Table 3.1: Relationship between $Constraint_1$ and $Constraint_2$

The coefficients in these constraints must as always sum to 0, so if the first $u-1$ coefficients of the constraints conform to the described relationship, then the last coefficient will also.

The states of I_{n-1} are written $(c^1_{n-1} \cdots c^m_{n-1})$, where there are m configurations of the nodes in I_{n-1}. Throughout the rest of this section three different probability distributions over I_{n-1} will be used frequently, therefore shorthand notations for these probability distributions are

listed in Equation 3.1.

$$\begin{aligned} p_\delta^k &= P\big(c_{n-1}^k | \delta_{D_{n-1}}(past(D_{n-1})), epast(D_{n-1})\big) \\ p_1^k &= P\big(c_{n-1}^k | d_{n-1}^1, epast(D_{n-1})\big) \\ p_2^k &= P\big(c_{n-1}^k | d_{n-1}^2, epast(D_{n-1})\big) \end{aligned} \tag{3.1}$$

Equation 3.2 shows the general method for calculating one coefficient (c_i) when decision $\delta_{D_{n-1}}(past(D_{n-1}))$ is observed. $\rho_{D_n,i}(epast(D_n))$ is the $i'th$ coefficient determined for D_n, given the past $epast(D_n)$.

$$c_i = \sum_{k=1}^{m} p_\delta^k \cdot \rho_{D_n,i}(epast(D_n)) \tag{3.2}$$

What happens in Equation 3.2 corresponds to replacing decision node D_n by a chance node, and then calculating the coefficient. To calculate ρ correctly then, just as when substituting a decision node by a chance node, the decision node must be fully observed. When this is not the case, virtual observations are imputed so that $\rho_{D_n,i}$ can still be calculated.

Now let d_{n-1}^1 and d_{n-1}^2 be two decisions in D_{n-1} different from $\delta_{D_{n-1}}(past(D_{n-1}))$, it is now known that the expected utility of the two decisions are smaller than the expected utility of $\delta_{D_{n-1}}$, given $past(D_{n-1})$. Let $EU(\delta_{D_{n-1}}(past(D_{n-1}))) > EU(d_{n-1}^1(past(D_{n-1})))$ be $Constraint_1$ while $Constraint_2$ is $EU(\delta_{D_{n-1}}(past(D_{n-1}))) > EU(d_{n-1}^2(past(D_{n-1})))$. So coefficient c_i will be $\rho_{D_{n-1},i}(\delta_{D_{n-1}}(past(D_{n-1})), epast(D_{n-1}))$ with $\rho_{D_{n-1},i}(d_{n-1}^1, epast(D_{n-1}))$ subtracted, in $Constraint_1$. Based on Equation 3.2, Equation 3.3 shows how coefficient c_i is calculated for $Constraint_1$.

$$c_i = \sum_{k=1}^{m} (p_\delta^k - p_1^k) \cdot \rho_{D_n,i}(epast(D_n)) \tag{3.3}$$

The two constraints are created in the same decision node (D_{n-1}) based on two different decisions (d_{n-1}^1 and d_{n-1}^2) given the same relevant past ($past(D_{n-1})$) and observed decision ($\delta_{n-1}(past(D_{n-1}))$). Since the constraints are created in the same decision node given the same past, then the imputations made, and thereby $\rho_{D_n,i}$, will be the same when calculating both constraints. Equation 3.4 shows what must hold, for coefficient c_i in the constraints to conform with Table 3.1.

$$\begin{aligned} &\sum_{k=1}^{m} (p_\delta^k - p_1^k) \cdot \rho_{D_n,i}(epast(D_n)) = -\sum_{k=1}^{m} (p_\delta^k - p_1^k) \cdot \rho_{D_n,i}(epast(D_n)) \\ &\Updownarrow \\ &\sum_{k=1}^{m} (2 \cdot p_\delta^k - p_1^k - p_2^k) \cdot \rho_{D_n,i}(epast(D_n)) = 0 \end{aligned} \tag{3.4}$$

The expression in Equation 3.4 will not always be true. It will depend on the probabilities in the domain (p_δ^k, p_1^k and p_2^k), and the imputed policies ($\rho_{D_n,i}(epast(D_n))$). Each probability distributions will sum to 1, just as the coefficients calculated with $\rho_{D_n,i}$ will sum to 1.

Two probability distributions are added in the expression ($2 \cdot p_\delta^k$) while two other probability distributions are subtracted ($-p_1^k - p_2^k$), so summing this expression over all m configurations

of the relevant past would yield 0. However, all the expressions cannot sum to 0 individually for all m configurations, unless the utility function is trivial or if the different decisions have no impact on the utilities. These situations would both make the decision in node D_{n-1} irrelevant, in turn making no decision wrong and any prediction correct, so it is assumed that this is not the case.

Each of the expressions calculated using the probability distributions is multiplied by the expected utility of their outcome, $\rho_{D_n,i}(epast(D_n))$. The sum of this ρ expression over the the m configurations will be 1, and non of the ρ expressions will be negative. This is due to the utilities being normalized. The $\rho_{D_n,i}$ expressions are decided by the domain and the imputations made, so any combination of $\rho_{D_n,i}$ values is possible as long as they are all non negative and sum to 1. The reason for this is that any chance node later the than D_n in the temporal order have not been used in any expression so far, so the probability distribution for these chance nodes are not restricted in any way.

Now, with the part of the expression that is determined by probability distributions summing to 0 over the m configurations and $\rho_{D_n,i}$ summing to 1, then it will be possible to pick a set of values for $\rho_{D_n,i}$ that makes the entire expression from Equation 3.4 sum to 0.

To summarize, then in a domain where the agent does not change behavior over time, conflicts can still occur when imputations are made. The type of conflict that has been shown to be possible is the case where two constraints describe exact opposite half spaces, this is a very specific situation and is only meant to show that conflicts are possible.

Based on this result the methods that use imputations to make the domain fully observed, must also consider that these imputations can make the feasible space become empty. Therefore the methods should include techniques to handle conflicts.

3.2 Imputing Strategy

Utility Iteration and Imputing by Comparison are only dissimilar in the way the unobserved configurations of relevant pasts are imputed, the overall strategy, that the two methods share, is presented in this section.

The methods maintain a set of true observations, i.e. all the observations made so far. Every time a new observation is made this observation is added to that set. When the utility function is to be estimated by one of these imputing methods then this estimation is done based on the maintained set of true observations. So the two imputing methods are batch learning techniques, since when a new observation is made the results of the imputing methods from before that observations was made are discarded.

When the utility function is to be estimated, this is done by establishing constraints based on observed decisions. Equation 2.4 is used to create constraints, this equation assumes that the decision node being evaluated is the last in the temporal order, and finds the expected utility of each decision by weighing the utility coefficients by their probability of occurrence. For the imputing methods to use this equation, they must impute virtual observations such that the strategy of the observed agent becomes fully observed.

After the methods have created constraints based on all the decisions observed in the true observations, a point in the feasible space is chosen as the estimated utility function. This point is chosen, as in *FLUF*, to be the center of the largest possible hypersphere in the feasible space, see Section 2.5 for a description of the method.

3.2.1 Basic Technique

This section presents a basic imputing technique that illustrates the concept which the imputing methods are based upon. This technique basically dictates the order in which decision nodes should be evaluated and subsequently substituted by chance nodes.

The main idea is to calculate constraints for the last decision node in the temporal order first, and then replace the decision node with a chance node. The chance node encodes the policy of the corresponding decision node with ones and zeros. As mentioned, this is only possible if the decision node has been observed for all configurations of its relevant past, therefore imputing is done for unobserved configurations. With respect to the last decision in an observation, adding these constraints is straight forward since no subsequent decision will exist. Therefore in a decision node (D) where the observed decision was δ_D and the function ρ determines the coefficients for all utilities, as in *FLUF*, then the following constraints can be added: $\forall_{d \in D \setminus \delta_D} : \rho(\delta_D) > \rho(d)$.

Replacing a decision node (D_k) with a chance node (C_k), is done after creating constraints based on the decision in all observations in **O**, **O** being the set of true observations, and thereafter the needed virtual decisions, to replace D_k with C_k, are imputed. Now C_k can be used instead of D_k throughout the rest of the imputing method, and it is now possible to calculate constraints for decision node D_{k-1} and so forth. Algorithm 3.2.1 shows the order of the replacing and evaluations of decision nodes when using the basic technique. n is the number of decision nodes.

Algorithm 3.2.1

- **For** $node = n$ to 1
  - **For** all observations (o) in **O do**
    - Where $\delta_{D_{node}}$ is the observed decision and o_{node} is the relevant past of D_{node} in the observation, add the following constraints to the feasible space: $\forall_{d \in D_{node} \setminus \delta_{D_{node}}} : \rho_{D_{node}}(\delta_{D_{node}}, o_{node}) > \rho_{D_{node}}(d, o_{node})$
  - For every unobserved relevant past of decision node D_{node} impute a virtual observation
  - According to true and virtual observations, replace D_{node} by a chance node

Using this technique means that the number of imputations needed for one decision node will be $relevant - observed$, where $relevant$ is the number of different configurations of the relevant past of the node, while $observed$ is the number of different configurations of the relevant past that has been observed. Since every decision node is replaced once and used throughout the rest of the algorithm, the number of imputations needed in the worst case for the entire algorithm will be $\mathcal{O}(n \cdot (relevant_{max} - observed_{min}))$, where n is the number of decisions, $relevant_{max}$ is the largest relevant past in the domain and $observed_{min}$ is the least number of different relevant pasts observed for some decision node. So the number of imputations is linear in the number of decisions and configurations of their relevant past, and as more observations are made less imputations are needed.

3.3 Utility Iteration

In this section the first of the two imputing methods, called Utility Iteration, is presented and analyzed.

This method imputes virtual observations in order to view the agent's strategy as fully observed, and basically Utility Iteration attempts to stepwise refine the utility function, using the previous version of the utility function. So every time constraints are added to the feasible space a new utility point is chosen, this point is considered more accurate than the previous point since it is chosen based on one more observed decision. Every point chosen in this way corresponds to a so called *temporary utility function*. The newest temporary utility function is used when virtual observations need to be imputed, this is described in Section 3.3.1.

Each step in the refinement is done when the constraints added by one decision in one configuration of its relevant past, are added, i.e. as observed in one of the true observations. When the domain is static, the order of the observations is irrelevant. However, the order in which the decisions are evaluated is not irrelevant, since decisions late in the temporal order will create more accurate constraints.

The inaccuracy in the Utility Iteration method stems from the imputations that are made during execution, so the more imputations that affect the constraints created for a decision, the less reliable the constraints created will be. For this reason constraints are created in reverse order of the temporal order, such that the last decision in the temporal order is evaluated first. Since the order of observations is irrelevant, then for any decision node (D_k) in the temporal order, constraints are created for decision D_k in all observations, before the previous decision node (D_{k-1}) is evaluated in any observation. This is much like the basic technique described in Section 3.2.1, however an extension is presented in Section 3.3.1 that is expected to increases accuracy for Utility Iteration.

3.3.1 Imputing Virtual Decisions

All the way through the Utility Iteration algorithm, a feasible space is maintained, described by the constraints established as the algorithm progresses. So as more decisions are evaluated more constraints will be limiting this feasible space.

When imputations are needed for some decision node, they are done based on the current knowledge about the feasible space, i.e. the temporary utility function. To determine what decisions should be made in the needed virtual observations, the policy can be determined by maximizing expected utility according to the temporary utility function, since the observed agent is assumed to be rational. The determined policy is then used to replace the decision node by a chance node. As mentioned, the inaccuracy of the Utility Iteration methods lies in the imputations, this is because of the risk that the utility point chosen from the feasible space might not describe the strategy of the observed agent. If this occurs then constraints added to the feasible space will be wrong, enlarging the risk of choosing a wrong point again later. Inaccurate imputations may lead to wrong predictions or even the feasible space becoming empty, i.e. causing a conflict in a domain where the agent never changes behavior. This inaccuracy is analyzed in detail in Section 3.3.3.

In the following section an extension to the technique described in Section 3.2.1 is presented. This extended technique imputes decisions repeatedly, as more is learned about the feasible space, resulting in a higher accuracy. A third technique is presented in Appendix B, which uses a backtracing approach, trying different combinations of imputations in order to get the best result, this approach is not a viable alternative, since it has a high complexity.

Extended Technique

It makes sense to evaluate the observations sequentially, since anything learned from one observation can be used to impute more accurately in another observation. So that at some decision node in the temporal order (D_k), first observation o_1 is used to create constraints at D_k, by imputing policies for all subsequent nodes ($D_i|i > k$). Imputations in o_2 for subsequent nodes ($D_i|i > k$) can then be done more accurately, if the constraints created for D_k in o_1 are taken into account when determining the temporary utility function. With the extended technique imputations would be done anew each time they are needed. So decision node D_k would still be used to create constraints in all observation, but using the extended technique all subsequent decision nodes ($D_i|i > k$) will be imputed during the evaluation of D_k in each observation and not only once. Using this, probably more accurate, approach comes at a trade off in complexity, due to the higher number of imputations.

3.3.2 The Utility Iteration Algorithm

In this section a pseudo code algorithm of the Utility Iteration method, using the extended technique from Section 3.3.1, is presented. As mentioned in Section 3.3.1 the method maintains a feasible space described by a set of constraints ($\mathcal{C}_C$), which initially describes the entire normalized region. During execution of the algorithm constraints will be added to $\mathcal{C}_C$. Likewise the algorithm will maintain a list of all true observations, this list is termed L. Every time a new observation (o_{new}) is made it is inserted in the beginning of this list ($o_{new}|L$), in the algorithm it is assumed that this insertion operation has already been done.

As mentioned observations are evaluated sequentially, with respect to each step in the temporal order, which can also be seen in Algorithm 3.3.1. If the agent does not change behavior between observations this sequence is irrelevant.

Also mentioned in Section 3.3.1, was that the method contained inaccuracies when imputing. This means that even though the agent never changes behavior, the virtual observations may end up causing a conflict with a true observation. It is theoretically possible to iteratively step back and forth between observations trying different utility functions, which inevitably would lead to a utility function with which all true observations could conform. This would become very complex however, as shown in Appendix B, so instead the true observations in which the conflict occurs are simply deleted.

Algorithm 3.3.1

- **For** $k = n$ to 1
  - **For** all observations, $o \in L$, the following is done for one observation at a time
    1. Set *point* to be a utility point in the feasible space, as described in Section 2.5.
    2. **For** all relevant pasts, $past(D_i)$, of each decision node, D_i, after decision node D_k $(i > k)$
       - **If** D_i given $past(D_i)$ has been observed **then** no imputing is done
       - **Else**, impute a decision in node D_i for $past(D_i)$ according to the utility point
    3. Substitute decision nodes after node k by chance nodes corresponding to imputed policies
    4. Where δ_{D_k} is the true decision observed in o, create the following set of constraints ($\mathcal{C}_{temp}$): $\forall_{d \in D \setminus \delta_{D_k}} : \rho_{D_k}(\delta_{D_k}, o_k) > \rho_{D_k}(d, o_k)$
    5. **If** $\mathcal{C}_C \cup \mathcal{C}_{temp}$ describes the empty space **then** delete observation o and all its constraints in $\mathcal{C}_C$

6. **Else** let $\mathcal{C}_C = \mathcal{C}_C \cup \mathcal{C}_{temp}$

- Choose a utility point and return it as the predicted utility function

With regard to the complexity of the algorithm the number of imputations needed for one decision node will be $relevant - observed$, but when evaluating an observed decision from node D_k in an observation, all subsequent nodes $(D_i|i > k)$ are imputed, each of these needing $relevant - observed$ imputations. This means that to create constraints for D_k in one observation, $\mathcal{O}((n-k) \cdot (relevant_{max} - observed_{min}))$ imputations are needed in the worst case. Since these imputations are done anew for every observation, then to create constraints for all observations of node D_k the number of imputations needed in the worst case becomes $\mathcal{O}(obs \cdot (n-k) \cdot (relevant_{max} - observed_{min}))$, where obs is the number of true observations. Finally, the number of imputations needed for the entire algorithm will, in the worst case, be :

$$\begin{aligned} &\mathcal{O}\left(obs \cdot (relevant_{max} - observed_{min}) \cdot \sum_{k=1}^{n-1} k\right) \\ &\Updownarrow \\ &\mathcal{O}\big(obs \cdot relevant_{max} \cdot n^2\big) \end{aligned} \tag{3.5}$$

The increased accuracy of the extended technique compared to the basic technique, comes at a trade off in complexity. Since the complexity introduced is not exponential, the second technique is still considered operational and will be used in this project for Utility Iteration.

Example 3.3.1 This is an example of how Utility Iteration imputes virtual observations and generates constraints. This example is set in the same domain (Illustrated in Figure 2.3) as example 2.7.1 and with the same observation: $\langle D_1 = d_1^2, C_1 = c_1^1, D_2 = d_2^2 \rangle$.

Constraints can be generated for the last decision, D_2, without imputing any observations. This results in the same constraint as in Example 2.7.1:

$$\begin{aligned} &\rho_{D_2}(C_1 = c_1^1, D_2 = d_2^2) > \rho_{D_2}(C_1 = c_1^1, D_2 = d_2^1) \\ &0.1v_1 + 0.4v_2 + 0.5v_3 > 0.4v_1 + 0.2v_2 + 0.4v_3 \\ &-0.3v_1 + 0.2v_2 + 0.1v_3 > 0 \end{aligned} \tag{3.6}$$

To establish constraints based on the decision observed in node D_1, imputation has to be done for the unobserved outcome of C_1. Now a temporary utility function is needed, and it is obtained by choosing a utility point in the feasible space. With the only constraint limiting the feasible space being $-0.3v_1 + 0.2v_2 + 0.1v_3 > 0$ and the normal constraints for the normalized region, the center of the largest hypersphere becomes $(0.307956, 0.692044, 0.692044)$. How this point can be found is described in Section 2.5. With this utility point the decision in the virtual observation imputed in D_2 given C_1 in state c_1^2 becomes d_2^1, as can be seen from Equation 3.7 and 3.8.

$$EU(C_1 = c_1^2, D_2 = d_2^1) = 0.1 \cdot 0.307956 + 0.7 \cdot 0.692044 + 0.2 \cdot 0.692044 \approx 0.6536 \tag{3.7}$$

$$EU(C_1 = c_1^2, D_2 = d_2^2) = 0.3 \cdot 0.307956 + 0.1 \cdot 0.692044 + 0.6 \cdot 0.692044 \approx 0.5768 \tag{3.8}$$

Given this imputation and the observed decision, decision node D_2 can now be replaced by a chance node. Thereafter the constraint for decision D_1 can be calculated as if it was the last node in the temporal order, this is done in Equation 3.9.

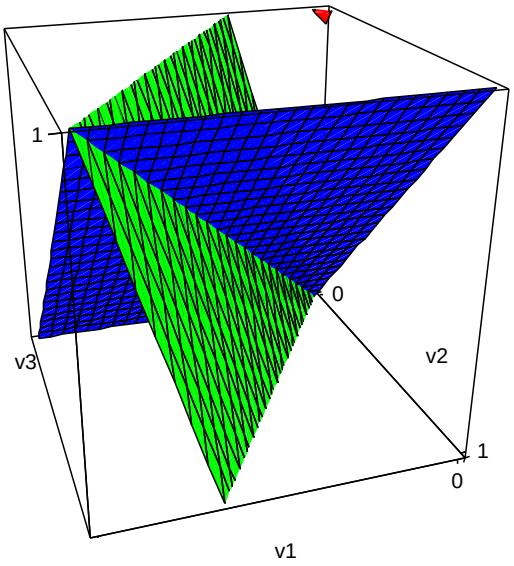


Figure 3.1: Both the green constraint ($-0.3v_1 + 0.2v_2 + 0.1v_3 > 0$) and the blue constraint ($v_2 < v_3$) describe the space containing the coordinate (0,0,1), illustrated by the red spot

$$\begin{aligned} \rho_{D_1}(D_1 = d_1^2) &> \rho_{D_1}(D_1 = d_1^1) \\ 0.1v_1 + 0.475v_2 + 0.425v_3 &> 0.1v_1 + 0.61v_2 + 0.29v_3 \\ -0.135v_2 + 0.135v_3 &> 0 \end{aligned} \tag{3.9}$$

This corresponds to that $v_2 < v_3$. The constraint can be seen in Figure 3.1, comparing this figure to Figure 2.4, it can be seen that Utility Iteration describes a smaller feasible space than *FLUF*. With more observations Utility Iteration will have more information on which it can create constraints and accuracy will increase as a result.

□

3.3.3 Analysis

This analysis will focus on the possible inaccuracies in the temporary utility function. First it is discussed how inaccuracies can be recognized and how often they occur. After that the impact of the inaccuracies is discussed, to determine whether such inaccuracies will reinforce themselves over time. The time complexity of the Utility Iteration method, using the extended technique, is also discussed.

Imputations

At every step during the refinement of the utility function, temporary utility functions may be chosen that are different from the observed agent's actual utility function. These temporary utility functions could therefore impute observations that do not conform with the strategy used

by the agent, and the only way for the Utility Iteration method to discover such an inaccuracy, would be if the feasible space for utility values becomes empty. In such a situation, no method is in place to iteratively find other temporary utility functions, since it would be far to complex. Instead the algorithm will continue executing by deleting the conflicting observation.

The first utility function will be based on the last decision in all observations, and no imputing is done at this point, so the constraints created will be accurate. So the more observations, the better the initial utility function will be. Furthermore, the more observations that have been made, the less imputing is necessary. Even though the first constraints are correct, they may still span a feasible space in which several different strategies are possible, and since only one strategy is completely correct then when imputations are necessary they may be wrong. The next section describes how these inaccuracies can impact the creation of constraints.

Constraints

The utility coefficients for decisions in the last decision node in the temporal order will always be correct, since the probabilities are assumed to be known. But for decisions in earlier decision nodes, the utility coefficients will depend on the imputed policies for subsequent decision nodes. As these imputed policies may not be the same as the policies of the agent, the utility coefficients may be incorrect.

Constraints are added to the feasible space based on the utility coefficients, so inaccuracies in the utility coefficients can result in temporary utility functions being chosen that are different from the observed agents utility function. Initial inaccurate imputations can lead to imprecise temporary utility functions, in turn leading to more inaccurate imputations.

To analyze whether the constraints created by Equation 2.4 will reinforce inaccuracies, some notation is first introduced. If a temporary utility function is inaccurate enough, when evaluating decision node D_k, so that the imputed policy for some decision node $(D_i|i > k)$ is different from the policy of the agent, then Equation 3.10 will be true.

$$\begin{aligned} \exists d \in D_k: & \\ & \rho_{D_k}(d, o_k) = c_{d,1}v_1 + \cdots + c_{d,n}v_n \\ \neq \quad & \rho_{D_k,true}(d, o_k) = c_{d,true,1}v_1 + \cdots + c_{d,true,n}v_n \end{aligned} \tag{3.10}$$

Where $\rho_{D_k,true}(d, o_k)$ describes the values that should have been attributed to option d in decision node D_k given the past o_k. Now, let $\rho_{D_k}(d, o_k)$ be described as in Equation 3.11.

$$\rho_{D_k}(d, o_k) = (c_{d,true,1} + \Delta c_{d,1})v_1 + \cdots + (c_{d,true,n} + \Delta c_{d,n})v_n \tag{3.11}$$

Where $\Delta c_{d,n}$ is the difference between $c_{d,true,n}$ and $c_{d,n}$. In Equation 3.12 this notation is used to establish constraints based on the observed decision δ_{D_k} and an alternative decision $d \in D_k$,

given past o_k:

$$
\begin{aligned}
&\rho(\delta_{D_k}(o_k), o_k) > \rho(d, o_k) \\
&\Updownarrow \\
&(c_{\delta_{D_k}(o_k),true,1} + \Delta c_{\delta_{D_k}(o_k),1})v_1 + \cdots + (c_{\delta_{D_k}(o_k),true,n} + \Delta c_{\delta_{D_k}(o_k),n})v_n > \\
&(c_{d,true,1} + \Delta c_{d,1})v_1 + \cdots + (c_{d,true,n} + \Delta c_{d,n})v_n \qquad (3.12) \\
&\Updownarrow \\
&((c_{\delta_{D_k}(o_k),true,1} - c_{d,true,1}) + (\Delta c_{\delta_{D_k}(o_k),1} - \Delta c_{d,1}))v_1 + \\
&\cdots + ((c_{\delta_{D_k}(o_k),true,n} - c_{d,true,n}) + (\Delta c_{\delta_{D_k}(o_k),n} - \Delta c_{d,n}))v_n > 0
\end{aligned}
$$

Notice that the expression $(c_{\delta_{D_k}(o_k),true,n} - c_{d,true,n})v_n$ corresponds to the actual difference between the coefficients that should have been attributed to the n'th utility, while $(\Delta c_{\delta_{D_k}(o_k),n} - \Delta c_{d,n}))v_n$ is the resulting impact on the constraints from the inaccuracies in the utility functions.

These Δc will depend much on the temporary utility functions, in the sense that the imputed policies are shaped after the utility functions and the coefficients are only changed from their true values when the policies are changed. So if the temporary utility function rates a utility (u_k) lower than it should, then the policy imputed may change accordingly making a different decision, given some relevant past, that reduces the c_k coefficient. In other words, if a Δc value is positive, then the corresponding utility was overrated by the temporary utility function, and underrated for a negative Δc value. This would imply that coefficients for the same utility, such as $c_{\delta_{D_k}(o_k),n}$ and $c_{d,n}$, would have the same sign on their Δc values.

When creating constraints the impact of the Δc values is decided by their relative size. The result of having a positive $\Delta c_{\delta_{D_k}(o_k),n} - \Delta c_{d,n}$ expression would be a relaxed constraint with respect to the corresponding utility, u_k, while a negative value would result in a stricter than normal constraint.

As mentioned earlier, it is certain that the coefficients calculated for any observed decision will have some truth to them, which stems from the last decision having the correct coefficients attached for the observed relevant past. Naturally, there can be a lot more truth to both observed and unobserved decisions than that, but on average the coefficients of an observed decisions may be marginally more accurate than those of an unobserved decisions. Accurate meaning small Δc values.

If an observed decision ($\delta_{D_k}(o_k)$) is more accurate than an unobserved decision ($d \in D_k$), with respect to a single utility coefficient c_r, then $|\Delta c_{\delta_{D_k}(o_k),r}|$ will be smaller than $|\Delta c_{d,r}|$. This will in turn mean that positive Δc_r values would, all else being equal, generate relaxed constraints, rating the corresponding utility (u_r) smaller, while negative Δc_r values would generate stricter constraints, rating u_r higher. This means that inaccuracies on u_r should diminish over time. But whether any significant difference in accuracy between the coefficients of observed and unobserved decisions will exist is uncertain.

However, without conducting experiments it is very difficult to tell whether inaccuracies will reinforce themselves.

Complexity

With regard to the complexity of the Utility Iteration method, then using the extended technique the number of imputations in the worst case will be $\mathcal{O}(obs \cdot relevant_{max} \cdot n^2)$, as shown in Section 3.3.1. For each imputation the influence diagram is solved based on the temporary

utility function, but only with respect to one decision given one relevant past. This task has a time complexity of $\mathcal{O}(nodes^{states})$, where *nodes* are the nodes that are unobserved when the decision being imputed is made, and *states* is the largest number of states those nodes have. So if *nodes* is considered all nodes in the domain, the worst case time complexity of the entire Utility Iteration algorithm becomes $\mathcal{O}(obs \cdot relevant_{max} \cdot n^2 \cdot nodes^{states})$.

3.3.4 Summary

This analysis indicates that the Utility Iteration method will not reinforce inaccuracies, but to ensure that this is the case experiments need to be conducted. In any case, the method is very dependent on an accurate set of initial constraints, since these will impact the accuracy of imputations when later constraints are created. Should conflicts occur due to inaccurate constraints the only feasible solution, of those investigated here, is to delete observations and corresponding constraints. The extent and impact of inaccuracies will depend on the number of true observations, in that with more true observations more constraints can be created before imputations become necessary, making temporary utility functions more accurate.

3.4 Imputing by Comparison

In this section the other imputing method called Imputing by Comparison is described, this method is similar to Utility Iteration in many ways, and the main difference between the two methods is the imputing, which is discussed and analyzed in this section.

The idea in Imputing by Comparison is to impute the decisions, such that the virtual observation becomes the most like a true observation as possible. This is done by imputing the decision that, together with the relevant past in the virtual observation, makes it look the most like some relevant past and decision in a true observation. Considerations have to be made to determine how to best compare two different combinations of relevant past and decision with each other.

Determining which combination of relevant past and decision that looks the most like another, could be done by comparing how many variables that match, i.e. the variables are in the same state, and possibly how important these variables are. However, the importance of variables are derived from the utility nodes, so weighing the variables will not be possible since the utility values are unknown.

Another consideration that has to be made with regard to these comparisons, is that when two scenarios have been observed where neither match with the needed virtual observation with regards to some node N. Then if the two true observations have N in two different states, perhaps one of the states could be said to be closer to the state needed. An example of such could be the variable "want chocolate" that could be in the states: "no", "a little" and "craving", then "no" could be considered closer to "a little" than to "craving". However, the order of states in ordered variables is domain specific prior knowledge, and so comparing states will not be pursued further in this project.

Instead combinations of relevant pasts and decisions could be compared with respect to a set of hypothesis variables H. The set of hypothesis variables is all the variables that have a utility node as a child, denoted as $pa(U)$. This comparison should calculate different relevant pasts' impact on the distribution of H, so that two relevant past configurations are equivalent if they infer the same distribution on H. The intuition behind this idea is that it is not possible to compare utilities for different relevant pasts, but the impact on H may hint the expected utility.

It may be the case that a decision node has a utility node as a child, in which case the decision node would be contained in H, and probability distributions over decision nodes does not make sense. However, it will always be the case that, when the distribution over H is calculated, the decision node will be determined to be in some state, i.e. the calculation is made with respect to a specific decision. So during calculation of H's distribution, the decision node can be substituted by a chance node, with a probability of 1 for the observed or imputed decision. Any chance node in the relevant past of the decision will also be instantiated, so when calculating H any chance nodes in the relevant past will be treated as chance nodes with evidence on them. In this way the probability distribution of H can be calculated in any domain. The two ways to compare probability distributions, that have been examined, is *Kullback-Leibler divergence* and *Euclidean distance*, both described in Section 3.4.1.

3.4.1 Measuring Distance Between Probabilities

A commonly used method for calculating distance between probability distributions is the Kullback-Leibler divergence. This method is also known as the *relative entropy* between two probability distributions:

$$KL(\mathbf{p},\mathbf{q}) = \sum_k q_k \log_2 \left(\frac{q_k}{p_k}\right) \tag{3.13}$$

Where $\mathbf{p}$ and $\mathbf{q}$ are discrete probability distributions and q_k and p_k is the probability for outcome k in the two distributions. Note that Kullback-Leibler divergence is not symmetric in p and q. Kullback-Leibler have the nice property of being *strictly proper*, i.e. $KL(\mathbf{p},\mathbf{q}) = 0$ if and only if $\mathbf{p} = \mathbf{q}$, and $KL(\mathbf{p},\mathbf{q}) > 0$ when $\mathbf{p} \neq \mathbf{q}$. (Jensen, 2001)

Kullback-Leibler could be used by calculating the distance between the probability distribution of H, given the observed past and the observed decision (δ_D) in some true observation, versus the distribution given the unobserved past needed in the virtual observation and the different decisions in D ($\forall d \in D$). This calculation would be done for all true observations, and the decision in the calculation which yields the smallest distance would be chosen as the decision in the virtual observation. If two calculations yields equal distances then a decision, from one of the calculations, would be chosen at random.

Even though Kullback-Leibler is a commonly used method for calculating differences in probability distributions, it is not the best choice for Imputing by Comparison. Basically the intuition behind this approach is that if a set of utility coefficients have been observed as being preferable in some context, then a decision in a virtual observation yielding coefficients close to the observed ones should be good. However, Kullback-Leibler distances weigh small probabilities heavier than larger probabilities, as shown in Equation 3.14 where the Kullback-Leibler divergence between two likely outcomes ($q_{likely} \cdot \log_2 \frac{q_{likely}}{p_{likely}}$) is smaller than the divergence between two less likely outcomes ($q_{unlikely} \cdot \log_2 \frac{q_{unlikely}}{p_{unlikely}}$) even though the difference in probabilities in both cases are the same (0.01). An example where Kullback-Leibler distance is a poor measure is presented in Example 3.4.1.

$$\begin{aligned} 0.96 \cdot \left(\log_2 \frac{0.96}{0.95}\right) &< 0.05 \cdot \left(\log_2 \frac{0.05}{0.04}\right) \\ \Updownarrow & \\ 0.0145 &< 0.0161 \end{aligned} \tag{3.14}$$

Example 3.4.1 Consider an example where H has four outcomes, meaning the domain has four utilities. Each of H's outcomes yield a coefficient of one for a different utility. A true

observation has been made with the distribution $P(H|obs_{true}) = (0.001, 0.2, 0.399, 0.4)$ over H. Only two different decisions can be chosen for the virtual observation, so the one with the distribution that yields the smallest Kullback-Leibler value should be chosen. Decision d_1 yields the distribution $(0.12, 0.2, 0.34, 0.34)$ while the second decision, d_2, yields the distribution $(0.001, 0.1, 0.299, 0.6)$. Table 3.2 shows the Kullback-Leibler distances, as well as numerical difference of the probabilities. The table shows that decision d_2 gives the smallest Kullback-Leibler distance between the distributions of the hypothesis variables, in spite of giving the largest numeric differences, i.e. $|0.119| + |0| + |-0.059| + |-0.06| < |0| + |-0.1| + |-0.1| + |0.2|$ corresponding to $0.238 < 0.4$.

Virtual distributions	**KL distance to $P(obs_{true})$**	**Numeric difference**
(0.12,0.2,0.34,0.34)	0.1789	(0.119,0,-0.059,-0.06)
(0.001,0.1,0.299,0.6)	0.1155	(0,-0.1,-0.1,0.2)

Table 3.2: Kullback-Leibler distances and numeric difference

□

Since these probabilities translate directly to utility coefficients, the numerical difference should be as small as possible, and this is not ensured when using Kullback-Leibler. So instead Imputing by Comparison uses the Euclidean distance between coefficients to measure which distribution is closest to that of a true observation. Equation 3.15 shows how Euclidean distances between probability distributions ($\mathbf{p}$ and $\mathbf{q}$) are calculated. Using Euclidean distance the chosen virtual decision will be the one where the numeric difference is the smallest. Using Euclidean distance in Example 3.4.1 would yield the results shown in table 3.3, the smallest distance being attributed to the virtual decision with the smallest sum of numeric differences.

$$Ec(\mathbf{p}, \mathbf{q}) = \sum_k (q_k - p_k)^2 \tag{3.15}$$

Virtual distributions	**Ec distance to $P(obs_{true})$**	**Numeric difference**
(0.12,0.2,0.34,0.34)	1.2354	(0.119,0,-0.059,-0.06)
(0.001,0.1,0.299,0.6)	1.4086	(0,-0.1,-0.1,0.2)

Table 3.3: Euclidean distances and numeric difference

3.4.2 The Imputing by Comparison Algorithm

In this section the Imputing by Comparison algorithm is presented. This method uses the basic technique presented in Section 3.2. After the new observation has been added to $\mathbf{O}$, then Algorithm 3.4.2 is run. Algorithm 3.4.2 adds constraints for every decision in all observations, such that each decision node D_k is evaluated in all observations before any prior decision node, i.e. D_{k-1}, is evaluated in any decision.

Let D be the decision node that needs to be imputed, in some virtual observation called *obs*. The configuration of the relevant past for decision node D in *obs* is called $v\_relevant_D$. *v_relevant* is short for "virtual relevant", and *t_relevant*, is short for "true relevant", these are used in the algorithm to describe relevant pasts. Now the algorithm for imputing is shown in Algorithm 3.4.1.

Algorithm 3.4.1

1. **For** all observations o in $\mathbf{O}$, where $t\_relevant_D$ is the relevant past of D in o and $\delta_D(t\_relevant_D)$ is the observed decision given the past
   - **For** all $d \in D$
     - Calculate $Ec(P(H|v\_relevant_D, d), P(H|t\_relevant_D, \delta_D(t\_relevant_D)))$
     - If the observation yields the lowest Euclidean distance so far, mark d, and unmark any already marked ds with a greater Euclidean distance
2. Choose an arbitrary d among the marked decisions
   - Use the chosen decision as the decision in *obs*

Using Algorithm 3.4.1 to impute observations, the entire algorithm for Imputing by Comparison is shown in Algorithm 3.4.2. Algorithm 3.4.2 is based on Algorithm 3.2.1 for the basic technique.

Algorithm 3.4.2

- **For** $node = n$ to 1
  - **For** all observations (o) in $\mathbf{O}$ do
    1. Where $\delta_{D_{node}}$ is the observed decision and o_{node} is the relevant past of D_{node} in the observation, add the following constraints to the feasible space: $\forall_{d \in D_{node} \setminus \delta_{D_{node}}} : \rho_{D_{node}}(\delta_{D_{node}}, o_{node}) > \rho_{D_{node}}(d, o_{node})$
    2. If the feasible space has become empty, remove all constraints added by o and remove o from $\mathbf{O}$
    3. **For** every unobserved relevant past of decision node D_{node}, call Algorithm 3.4.1 to impute a decision
    4. According to observed and imputed decisions, replace D_{node} by a chance node

Step 3 and 4 are, strictly speaking, not necessary for the first decision node in the temporal order, as decision nodes are replaced with chance nodes so that the prior decision node becomes the last in the temporal order.

As discussed in Section 3.1.1 conflicts may occur when imputing observations to make the domain fully observed. To handle this, Algorithm 3.4.2 must check if the feasible space becomes empty, i.e. the radius of the largest possible hypersphere in the feasible space is zero, each time a new constraint is added. If the space becomes empty the newly added constraints are removed again.

When the newest constraint reveals a conflict it is, in a static domain, possible to avoid removing constraints altogether by imputing differently. The immediate way of doing this would be to examine those imputations where there were more than one decision with the same Euclidean distance, and one of which was chosen arbitrarily. In these cases the alternative choices should be used instead. Unfortunately it is not certain that it would give a non-empty feasible space. If that is the case the choices with the second shortest Euclidean distance would have to be examined. Again that does not guarantee that the feasible space becomes non-empty, so the choices with the third shortest distance might have to be examined and so on. In the worst case all combinations of possible imputations over all relevant pasts, in all but the first decision node, would have to be examined to find a non-empty space.

The advantage of this alternative method for removing constraints is that, in a static domain, it will eventually find a combination of imputations that describe a non-empty feasible space. However, the time complexity makes it infeasible. In fact, in the worst case

the number of imputation needed would be exponential in the number of decision nodes, $\mathcal{O}(|D|_{max} \cdot relevant^n_{max})$, where $relevant_{max}$ means the largest number of configurations of any relevant past in the domain and $|D|_{max}$ is the largest number of different decisions in one decision node.

Example 3.4.2 This is an example of how Imputing by Comparison chooses virtual observations and generates constraints. This example is set in the same domain (illustrated in Figure 2.3) as Example 2.7.1 and with the same observation: $\langle D_1 = d_1^2, C_1 = c_1^1, D_2 = d_2^2 \rangle$.

Constraints can be generated for the last decision, D_2, without imputing any observations. This results in the same constraint as in Example 2.7.1. The observed decision, d_2^2, must yield a larger expected utility than d_2^1, given chance node C_1 in state c_1^1:

$$\begin{aligned} &\rho_{D_2}(C_1 = c_1^1, D_2 = d_2^2) > \rho_{D_2}(C_1 = c_1^1, D_2 = d_2^1) \\ &0.1v_1 + 0.4v_2 + 0.5v_3 > 0.4v_1 + 0.2v_2 + 0.4v_3 \\ &-0.3v_1 + 0.2v_2 + 0.1v_3 > 0 \end{aligned} \tag{3.16}$$

To establish constraints based on the decision observed in node D_1, imputation has to be done for the unobserved outcome of C_1. This means that the probability distribution of the hypothesis variables, which in this example is limited to C_2, given the true observation, $t = \langle D_1 = d_1^2, C_1 = c_1^1, D_2 = d_2^2 \rangle$, must be compared with the distribution given $v_1 = \langle D_1 = d_1^2, C_1 = c_1^2, D_2 = d_2^1 \rangle$ and $v_2 = \langle D_1 = d_1^2, C_1 = c_1^2, D_2 = d_2^2 \rangle$ respectively, where v_1 and v_2 are the two virtual observations, and t the true observation. The Euclidean distance is calculated according to Equation 3.15, the calculations are shown below:

$$\begin{aligned} Ec\big(P(C_2|t), P(C_2|v_1)\big) &= (0.1 - 0.1)^2 + (0.4 - 0.7)^2 + (0.5 - 0.2)^2 = 0.18 \\ Ec\big(P(C_2|t), P(C_2|v_2)\big) &= (0.1 - 0.3)^2 + (0.4 - 0.1)^2 + (0.5 - 0.6)^2 = 0.14 \end{aligned}$$

In the virtual observation d_2^2 is chosen for C_1 in state c_1^2, since it yielded the smallest Euclidean distance to the true observation, and the final constraint can then be calculated.

$$\begin{aligned} &\rho_{D_1}(d_1^2) > \rho_{D_1}(d_1^1) \\ &0.15v_1 + 0.325v_2 + 0.525v_3 > 0.24v_1 + 0.19v_2 + 0.57v_3 \\ &-0.09v_1 + 0.135v_2 - 0.045v_3 > 0 \end{aligned} \tag{3.17}$$

The constraints generated by Imputing by Comparison can be seen in Figure 3.2, and the space spanned can be compared to Figure 2.4, which shows the space spanned by *FLUF* and the optimal method given the same observation. The space described by the Imputing by Comparison method is smaller than the space spanned by *FLUF*, and it still includes the space spanned by the optimal method.

□

3.4.3 Analysis

This analysis will focus on the inaccuracies that might be introduced in the Imputing by Comparison method. The complexity of the Imputing by Comparison method is also analyzed.

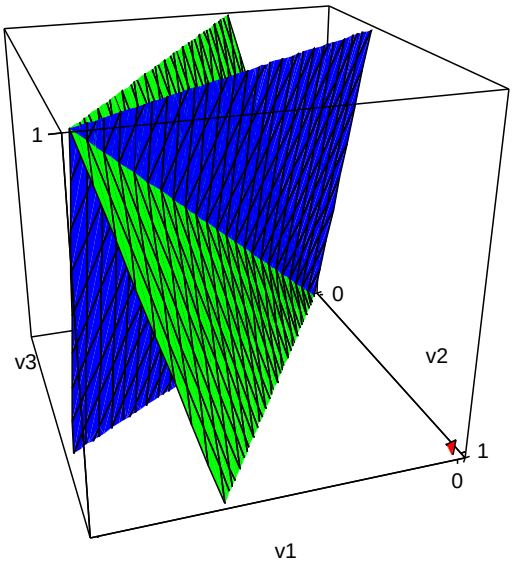


Figure 3.2: Both the green constraint $(-0.3v_1 + 0.2v_2 + 0.1v_3 > 0)$ and the blue constraint $(-0.09v_1 + 0.135v_2 - 0.045v_3 > 0)$ describe the space containing the coordinate (0,0,1), illustrated by the red spot

Accuracy

Inaccuracies can occur when imputing an observation, since decisions that are not optimal, according to the observed agent's utility function, can be chosen. Accuracy of the method will depend on the number of observations made, first of all because with more true observations less virtual observations will be necessary. Furthermore, the method compares all true observations with the possible virtual observations, meaning that a high number of true observations will result in a better foundation for choosing the decision for the virtual observation.

There is a risk that inaccuracies can reinforce themselves, since the distribution of the hypothesis variables, that possible virtual decisions are compared with, is influenced by the chance nodes that replace decision nodes after imputations. So if some imputing for node D_k is incorrect then the distributions, that are compared to determine imputation in node D_{k-1}, will also be incorrect. This inaccuracy increases the risk that decision D_{k-1} is imputed incorrectly. However, as the number of different true observations increase the extent of inaccuracies, and any reinforcement of inaccuracies, will diminish.

Complexity

With regards to the complexity of this method, each imputation is linear in the number of observations, since the Euclidean distance, between the distribution over the hypothesis variables of the virtual observation and the corresponding distribution in each true observation, must be calculated. So the complexity of imputing one decision becomes $relevant_{true} \cdot |D|$, where $relevant_{true}$ is the number different relevant past configurations already observed for the decision node, and $|D|$ is the number of possible decisions. A true observation may contain configuration of a relevant past that have already been observed, but $relevant_{true}$ will be growing with the number of observations made.

In each decision node, the number of imputations needed is $relevant - relevant_{true}$, where $relevant$ is the number of possible configuration of the relevant past. $relevant_{true}$ is subtracted from $relevant$ because imputations are only done for unobserved configurations of the relevant past, and $relevant_{true}$ can at most become equal to $relevant$ in which case no virtual observations are needed.

So as more observations are made, with respect to some decision node, fewer imputations are necessary but each imputation becomes more complex, this is expressed as $(relevant - relevant_{true}) \cdot (relevant_{true} \cdot |D|)$.

During execution, policies will be imputed for all decision nodes in the domain, meaning each relevant past configuration of each decision node, that has not been observed, is imputed. With n decision nodes in the domain the complexity for imputing for one decision node can be used to express the complexity of the entire Imputing by Comparison method as in Equation 3.18, where the i in subscripts imply that the variable corresponds to decision node D_i.

$$\sum_{i=2}^{n} |D_i| \cdot (relevant_{true,i} \cdot relevant_i - relevant_{true,i}^2) \tag{3.18}$$

Since this complexity is a second-order polynomial expression over $relevant_{true,i}$, complexity will increase, for decision node D_i, as observations are made until $relevant_{true,i} = \frac{-(relevant_i)}{2 \cdot (-1)} = \frac{relevant_i}{2}$. After this point complexity will decrease until $relevant_{true,i} = relevant_i$ where no imputations are needed. For this reason $relevant_{true,i}$ is substituted by $\frac{relevant_i}{2}$ when expressing worst case complexity. Now the number of times that Euclidean distance must be calculated, in the entire Imputing by Comparison algorithm, can be seen in Equation 3.19.

$$\begin{aligned} &\mathcal{O}\left((n-1) \cdot |D|_{max} \cdot \left(\frac{relevant_{max}}{1} \cdot relevant_{max} - \left(\frac{relevant_{max}}{2}\right)^2\right)\right) \\ &\Updownarrow \\ &\mathcal{O}\left(n \cdot |D|_{max} \cdot relevant_{max}^2\right) \end{aligned} \tag{3.19}$$

The complexity of Euclidean distance calculations is not constant, but linear in the size of the state space of the hypothesis variables. As the hypothesis variables are the parents of the utility nodes, the size of their state space is the number of utilities in the domain. Making Imputing by Comparisons worst case time complexity $\mathcal{O}\left(utilities \cdot n \cdot |D|_{max} \cdot relevant_{max}^2\right)$.

3.4.4 Summary

This leads to the conclusion that this method may be a viable alternative to *FLUF*. The impact of the inaccuracies that may be introduced by imputing wrong observations and the extent to which the inaccuracies are reinforced, cannot be predicted at this point. The accuracy of this method should grow with the number of observations made, since less imputations will be necessary and a better foundation for imputations will be available, but to get a better idea about the speed at which this accuracy will increase experiments must be conducted.

3.5 Conclusion

In this section results about the accuracy and complexity of the proposed methods from Sections 3.4 and 3.3 are briefly summarized. First the new imputing method's differences compared to *FLUF* are summarized.

3.5.1 Imputing compared to FLUF

There are two big differences between the two imputing methods and *FLUF*. The first is that *FLUF* is an adaptive learning technique, changing its feasible space every time a new observations is made, while the two new imputing methods are batch learning techniques, storing new observations in the set of true observations and when making predictions evaluate all the true observations. The second difference is the way partially observed strategies are handled. As described in Section 2.4 *FLUF* handles partially observed strategies by creating relaxed constraints, while the two new imputing methods handle partially observed strategies by imputing the needed observations to view the strategy as fully observed.

What the imputing methods have in common with *FLUF* is the feasible space, and the concept of creating constraints in this space when observations are made. The methods also share the basic way that constraints are created, namely that the expected utility of observed decisions must be larger than the expected utilities of their alternatives. The way a utility point is chosen in the *FLUF* method, i.e. the center of the largest possible hypersphere, is also used in the imputing methods.

3.5.2 Accuracy

For the imputing methods to be viable utility learning methods they should become more accurate over time, i.e. the number of decisions predicted correctly and expected utility should increase with the number of observations. Both analysis have shown that inaccuracies imposed during execution does not seem to be reinforced, implying that accuracy will increase as more observations are evaluated, making both methods usable. To confirm this result and to better determine the accuracy of the two methods, experiments will be conducted.

3.5.3 Complexity

The only difference between the two new methods is in the way imputing is done. This still leads to significant differences in worst case time complexity, where for Utility Iteration it is $\mathcal{O}\big(obs \cdot relevant_{max} \cdot n^2 \cdot nodes^{states}\big)$ while for Imputing by Comparison algorithm it is $\mathcal{O}\left(utilities \cdot n \cdot |D|_{max} \cdot relevant_{max}^2\right)$. So in some domain, as the number of observations increase, Utility Iteration will become slower relative to Imputing by Comparison. The actual speed of the methods cannot be determined at this point, but the worst case time complexities indicate that the Utility Iteration algorithm will be more sensitive to increasing state spaces of nodes.

CHAPTER

FOUR

Dynamic Domains

The two imputing methods and *FLUF* are basically designed to determine the utilities of an agent that does not change its behavior. Two different policies, for handling agents that change behavior over time, were designed as extensions for *FLUF* in Hansen et al. (2004) (these are briefly described below in Section 4.3.2). In this chapter the ways in which agents can change behavior and different approaches to handling changing behavior, in Utility Iteration and Imputing by Comparison, are discussed.

When modeling the behavior of an agent that does not change its behavior, an influence diagram can be constructed where the choices made by the agent corresponds to maximizing expected utility. Since the agent does not change behavior, then once the influence diagram describes its behavior correctly, it can be used to predict the agents choices and updating the diagram should never be necessary. A domain modeling an agent that does not change behavior is called a *static* domain, and the behavior of the agent is called *static* as well.

When modeling the behavior of an agent with changing behavior, then even though an influence diagram may be constructed where the current strategy of an agent corresponds to maximizing expected utility in the influence diagram, the agent can change its strategy over time such that the influence diagram must be updated as well, to keep describing the agents behavior correctly. A domain is called *dynamic* if the agent being modeled can change its behavior, according to the definitions below. The behavior of an agent in a dynamic domain is called *dynamic* as well.

To enable Imputing by Comparison and Utility Iteration to handle dynamic domains, methods for handling inconsistent observations, incurred by changing behavior, are described in Sections 4.3 and 4.4. Before these methods are presented, the different ways in which the agent's changing behavior can be modeled is analyzed.

4.1 Types of Dynamic Domains

To help design methods for specific kinds of behavioral changes, three different ways in which an agent can change its behavior are described. A good method for handling one kind of behavioral changes may not work for other kinds. Under the assumption that the causalities and probabilities in the domain, as the agent perceives them, are known and do not change, then changing behavior can be expressed as a changing utility function. Below, the three ways

of viewing changing behavior are described in terms of a utility point in the utility space, i.e. the point describing all utility values in the domain.

Drift

One way of viewing changing behavior is as *drift*. This means that the utility values are changing continuously. This kind of dynamic behavior can be seen as the utility point of the observed agent drifting around in the normalized region, which is also why it is called drift. Drift does not necessary to maintain its speed and direction, meaning that over time the drifting of the utility point can slow down or change direction. Changing behavior is categorized as drift when the strategy of the agent changes slowly and gradually, in the sense that the strategy may change often but only with respect to a few policies at the time. An example of drift could be if the utilities in the influence diagram are modeling the price of some goods, and these prices change over time. The prices may increase or decrease thus making the utilities drift accordingly over time.

Fluctuation

Changing behavior could be viewed as *fluctuations*, where fluctuations are radical changes introduced into the influence diagram. Fluctuations can introduce major and sudden shifts in the expected utility for the domain, changing the observed agent's strategy. With regards to the utility point of the observed agent, this corresponds to the utility point jumping from one position in the normalized region to a completely different position independent of the first. Changing behavior is categorized as fluctuating when the strategy of the agent can change radically, in the sense that a large number of policies in the agent's strategy change simultaneously.

Noise

The last way to view changing behavior introduced here is *noise*. Noise is not a change in the domain like drift and fluctuation. Instead noise is unforeseen interference not modeled in the domain and introduced by sources outside the domain, e.g. recording or reading incorrectly in a database of observations. Even though the utility point of the observed agent is not changed by noise, it may appear to have done so. Changing behavior is categorized as noise when the strategy followed by the agent in a single observation deviates from the strategy followed in previous observations, only to return to the strategy followed in previous observations again in subsequent observations. An example of noise could be interference with a humidity sensor resulting in a low reading, this would result in a chance node showing a wrong state. Noise can affect more than one node however, and it may affect decision as well as chance nodes.

The three different types of dynamic behavior presented here are not the only ways to categorize dynamic behavior. However, these are the only kinds of dynamic behavior considered in this project, and dynamic behavior will be classified according to the three categories.

4.2 Conflict Handling

This section describes the different policies for conflict handling in *FLUF*, Imputing by Comparison and Utility Iteration. The main idea behind conflict handling is to remove the constraints

causing the conflict, these are termed the *guilty* constraints. When a conflict occurs, the constraints considered guilty will depend on how the dynamic behavior of the agent has been categorized. In case of drift and fluctuations the oldest constraints will be considered guilty, while in case of noise the newest constraint is considered guilty. This means that the order in which the observations are evaluated are no longer insignificant.

If a domain contains both drift and fluctuation conflict handling can still be done, since the two kinds of dynamic behavior will characterize the same set of constraints as guilty. But if the domain contains noise along with either drift or fluctuations, then choosing which constraints are guilty becomes harder, because the guilty observation would be the newest in case of noise but the oldest in case of drift or fluctuation. One way to determine which constraints are guilty, assuming that an expected frequency of noise is given, would be, when conflicts occur, to view new constraints as being guilty as long as less conflicts occur than suggested by the expected frequency of noise. When more conflicts start occurring it would be a sign that some sort of drift or fluctuation had taken place, and old constraints should then be considered guilty. However the frequency of noise is not always known, and in this project domains with noise are assumed to show no other kind of dynamic behavior.

Two more limitations to the conflict handling policies for Imputing by Comparison and Utility Iteration should be noted. First, the imputations in both the methods may be affected by conflicting observations making them less accurate, this problem could in part be solved by extensive backtracking, as demonstrated in Appendix B, but to keep the conflict handling policies operational no modifications are made to handle imputing inaccuracies. Furthermore, as shown in Section 3.1.1, inaccurate imputations may lead to conflicts, and conflicts incurred due to inaccurate imputations should ideally be handled differently than conflicts that occur due to dynamic behavior, however since there is only one way to detect that a conflict occurs, namely that the feasible space become empty, no method has been developed to tell the difference.

Furthermore, as an alternative to the deletion policies, a policy called the *constraint relaxation policy* is developed, that can be used to handle conflicts that occur in domains containing drift and/or fluctuations. The policy is based on relaxing constraints when conflicts occur, so it would not be suitable for handling noise as it would relax all constraints, thereby reducing their accuracy, even though only one constraint was guilty. The noisy constraint would still be present in the feasible space after this relaxation, adding further to any inaccuracies and increasing the risk that further relaxation will be necessary when new observations are made.

Since Imputing by Comparison and Utility Iteration create constraints in the same way and order, they will have the same conflict handling policies and are therefore described in the same section. In the following sections different conflict handling policies are described, four policies suitable for conflicts caused by drift and fluctuations and one policy suitable for handling conflicts introduced by noise are described in the following.

- Drift and Fluctuation
  - A deletion policy for Imputing by Comparison and Utility Iteration
  - Two deletion policies for *FLUF*
  - The constraint relaxation policy, suitable for *FLUF*, Imputing by Comparison and Utility Iteration
- Noise
  - A deletion policy for Imputing by Comparison and Utility Iteration

4.3 Drift and Fluctuation

In this section policies are described for Imputing by Comparison, Utility Iteration and *FLUF*, that can be used when conflicts occur in dynamic domains. The policies in this section are designed to handle conflicts that occur due to dynamic behavior categorized as either drift or fluctuation. The reason why these types of dynamic behavior are described together, is that they both assume that the older observations are, the less likely they will be to conform to the current strategy. Therefore the policies described below can be applied to both kinds of dynamic behavior.Before these are presented a design issue, that influence conflict handling in both methods, is discussed, namely the coarseness with which constraints should be removed.

In Imputing by Comparison and Utility Iteration the methods maintain a set of true observations, unlike *FLUF* where a set of constraints is maintained instead. The two methods, Imputing by Comparison and Utility Iteration, still construct a set of constraints, but every time a new observation is made, the constraints created earlier are deleted so that a new set can be constructed. The goal of the conflict handling policies described below, is to resolve conflicts by removing the oldest constraints that conflict along with the observations they belong to. The reason why the entire observation is removed, is that if one of the constraints it adds can cause a conflict, then either drift or fluctuation has caused the policy to change for one or more of the decision nodes in their observed relevant pasts. At the time the observation that causes the conflict was made, the observed agent must have been using a different utility function than the current utility function, due to the drift or fluctuation that has occurred since then. So even though only one constraint conflicts, then the other constraints added by the observation may still be incorrect, i.e. the current utility values of the agent cannot be described by those constraints, and if they are not removed their incorrectness will affect which utility point is chosen. Also, the observation that is based on an obsolete utility function, will continue to affect the imputations if only the constraint is removed.

As mentioned, the methods described here will remove entire observations, to ensure that inaccuracies are reduced. Alternatively the extent of these removals could be limited to reduce the number of constraints deleted, e.g. by only removing constraints added based on the same decision node in the same observation. However, due to limited time available for this project, this alternative is not explored.

Another way of reducing the amount of observations removed, would be to use the method from *FLUF*, mentioned in Innocent Until Proven Guilty (Section 4.3.2), for finding the constraints that make the space empty. When these constraints are found the corresponding observations could be termed guilty. Unfortunately this approach would not work, since all constraints added by Imputing by Comparison and Utility Iteration will be based on a fully observed strategy, and any constraint based on a fully observed strategy will intersect with all points in the feasible space that describe the trivial utility function (the diagonal). In Appendix A it is proven that constraints created in domains with fully observed strategies will always intersect the diagonal, and this will be the case in Utility Iteration and Imputing by Comparison since they impute the needed observations to make the strategy fully observed. With all constraints intersecting each other in the diagonal, then all constraints would be termed guilty if the method used by *FLUF* in the Innocent until Proven Guilty policy was adopted.

4.3.1 Drift and Fluctuation in Imputing by Comparison and Utility Iteration

The approach used in Imputing by Comparison and Utility Iteration examines all observations in parallel, in the sense that all observations are examined with respect to decision node D_i

before any observation is examined with respect to decision node D_{i-1}. Therefore it is likely that an observation will have added several constraints by the time it is discovered to be a conflicting observation, and these constraints will not necessarily have been added recently.

As observations are examined in parallel the guilty observation should not necessarily be determined to be the one in which the conflict was discovered. While removing the conflicting observation will resolve the conflict immediately, but over time this approach may cause many accurate observations to be removed that did not have to. Under the assumption that the dynamic behavior in the domain can be categorized as either drift or fluctuation, then to resolve a conflict the method used works much like the Always Guilty policy in *FLUF*. Here observations are deleted one at the time according to age, such that the oldest observation is removed first along with its constraints, and removal of observations continue until the feasible space becomes non empty. Using this approach, then in the worst case the observation in which the conflict was discovered and all older observations will be deleted.

With $\mathbf{O}$ being the set of true observations, ordered by age such that $\mathbf{o}^1$ is the newest observation while $\mathbf{o}^m$ is the oldest, where m is the number of true observations in $\mathbf{O}$. Letting $\mathcal{C}$ be the set of constraints added to the feasible space, then Algorithm 4.3.1 describes how conflicts are handled by this policy. The algorithm is run no differently if more decisions have been evaluated in some observations than in others, e.g. if the first ten observations have been evaluated with respect to one more decision node than all other observations when a conflict occurs, that would have no impact on the algorithm.

Algorithm 4.3.1

- From $k = m$ to 1
  - Remove all constraints added by $\mathbf{o}^k$ from $\mathcal{C}$, and $\mathbf{o}^k$ from $\mathbf{O}$
  - **If** $\mathcal{C}$ describes a non empty space
    - **then** halt this algorithm

4.3.2 Drift and Fluctuation in *FLUF*

FLUF's conflict handling policies are designed to handle drift and fluctuation. The policies are called *Always Guilty* and *Innocent until Proven guilty*. The experiments in Hansen et al. (2004) showed that the two policies performed equally well when the domain drifts, and that the Always Guilty policy recovers from a fluctuation fastest.

Always Guilty

In the Always Guilty policy the constraints are removed in the order they were added. When a new constraint is added which makes the feasible space empty, the oldest constraint is deleted, if the feasible space is still empty the second oldest constraint is also deleted and so on. This way the oldest constraints are deleted until the feasible space becomes non empty. Many of the deleted constraints may not have caused the conflict but nevertheless they are deleted. In fact it is only certain that the last of the deleted constraints was guilty. The argument for deleting this many constraints, is that since the domain has either drifted or fluctuated since the guilty constraint was added, then all constraints added before the guilty would, just like the guilty constraint, have been added based on observations of an agent using a strategy that is now obsolete.

Innocent Until Proven Guilty

The Innocent until Proven Guilty policy removes a minimal amount of constraints, by only removing constraints that actually cause the feasible space to become empty. To determine which constraints cause the space to become empty, corresponds to determining which constraints make the radius of the largest possible hypersphere in the feasible space to become zero. This set of constraints can be found using the method for finding the largest possible hyper sphere, as described in Section 2.5. If the space is empty, the largest possible sphere will have a radius of zero, but the method will still calculate the center of this sphere. Finding the constraints that caused the space to become empty can now be done in linear time, by entering the coordinates of the determined center into each constraint, then the constraints that equals zero are the "guilty" constraints. These guilty constraints are removed one at the time, the oldest being removed first. Each time a constraint is removed the hypersphere is recalculated, and only if the radius is still zero the next constraint is removed.

4.3.3 The Constraint Relaxation Policy

The conflict handling policies described so far are based on deletion of observations and constraints, however the constraint relaxation policy relaxes constraints instead. This conflict handling policy is developed especially with drift in mind. Because constraints become less reliable the older they grow, they could be relaxed as they grew older to retain some reliability. The policy is based on an idea from the future work section in Hansen et al. (2004), where it is suggested that relaxing constraints could be done by adding a constant that grows each time a new observations is made, in an attempt to avoid conflicts incurred by drift. Since all constraints are on the form $f(\mathbf{x}) > 0$ adding a positive constant, $f(\mathbf{x}) + c > 0$, would increase the space spanned by that constraint. As c would gradually become larger, eventually the entire normalized region would be a subspace of the space spanned by the constraint, at which point the constraint would be obsolete and could be removed.

Using this policy conflicts will rarely occur, since each time a new observations is made, the constraints from all other observations will have been relaxed. Since all old constraints have been relaxed at least once, only the newest constraint will intersect the diagonal, ensuring that a hypersphere with a radius larger than zero can always exist near the diagonal. There is one exception to this rule, namely if the new observation conflicts with itself, which is possible if imputations are inaccurate. In this situation, the new constraint should be deleted, as is ordinarily done in the imputing methods when imputations cause conflicts.

A result of using constraint relaxation to avoid conflicts, is that when conflicts are avoided by relaxing constraints, the valid constraints will not be deleted but instead have their influence on the utility point diminished. It is likely that new constraints will at some point make the older constraints superfluous, as at some point the relaxed constraint may span a super space of the feasible space. This effect is desirable in fluctuation as well as drift.

In this project, instead of adding a constant in the constraint relaxation policy, the coefficients in the constraints are changed instead, so that the constraints will still be relaxed but will also continue to intersect with the origin. This is because translating constraints by adding constants will have an unwanted impact on the feasible space. When constraints are created in *FLUF*, Utility Iteration or Imputing by Comparison, they describe a relationship between expected utilities, such as $v_1 < v_2$. Since multiplying all utilities with the same positive constant would yield the exact same strategy, then constraints with constants added ($f(\mathbf{x})+c > 0$), would allow all combinations of utility values, as long as all values are less than c, thereby describing all strategies. For this reason, then instead of translating constraints they are rotated

around the origin, in a direction such that they describe an increasing part of the normalized region, until they describe the entire region at which point they can be deleted.

The Aguilera-Peréz algorithm, presented in Aguilera and Peréz-Aguila (2004), is one possible approach for rotating the constraints that has been considered. By providing the Aguilera-Peréz algorithm with a $(n-2)$ dimensional subspace to rotate around and an angle to rotate, a set of points sufficient to extrapolate the new position of the constraint could be calculated. The advantage of using a policy such as this, would be that the speed at which constraints rotate could be completely controlled, e.g. at five degrees in every rotation. Furthermore, the Aguilera-Peréz algorithm is designed such that a set of transformation matrices are calculated, and once such a matrix is calculated for a constraint it could be used every time the constraint was to rotate. Since the complexity of calculating such a matrix is polynomial in the number of dimensions (n^2), the algorithm could be considered operational. Rotating the hyper planes that define the constraints according to the Aguilera-Peréz algorithm would mean that the constraints are relaxed equally for all the utility values. This would be desirable as no particular utility value is candidate for more relaxation than others.

However, no method has been found to calculate the subspace that should act as the rotation axis, and since implementation of such a method along with the Aguilera-Peréz algorithm would be a time consuming task, a simpler but less elegant method is used.

As constraints are created from $\rho_D(\delta_D) > \rho_D(d)$ it would be possible to relax the constraint by either decreasing the coefficients of $\rho_D(d)$ or increasing the coefficients of $\rho_D(\delta_D)$. This would increase the difference between the two ρs and make it easier to satisfy the constraint. Such a relaxation would express a decreasing confidence in the relationship between the expected utility of the chosen decision (δ_D) and the alternative choice (d). When relaxing like this, it would be hard to control how fast the constraint should span the entire normalized region, therefore the relaxation is done by considering the utility coefficients of $\rho_D(\delta_D) - \rho_D(d) > 0$. This will have a set of positive as well as a set of negative coefficients. The space described by a constraint can be increased by increasing the negative coefficients, until all the coefficients are non negative at which point the constraint will described the entire normalized region. As long as at least one coefficient is negative the constraint will exclude a part of the normalized space, e.g. by subtracting a tenth of the original value of each negative coefficients from that coefficient every time rotation is done, then the constraint would describe the entire normalized region after ten rotations.

The constraint relaxation policy would have to be implemented differently in *FLUF* and the imputing methods, since *FLUF* maintains a set of constraints while the imputing methods maintain a set of observations instead. Using this policy in *FLUF*, Algorithm 4.3.2 would have to be executed every time constraints are created by a new observation. $\mathcal{C}$ denotes the set of constraints established so far, and $\mathbf{c}_{new}$ denotes the new constraints being added.

Algorithm 4.3.2

- Relax all constraints in $\mathcal{C}$
- Remove any constraint that describes the entire normalized region
- Add $\mathbf{c}_{new}$ to $\mathcal{C}$

Using the constraint relaxation policy in Utility Iteration or Imputing by Comparison, Algorithm 4.3.3 would have to be executed every time a new observation is made. In the algorithm $\mathbf{O}$ denotes the set of true observations and $\mathbf{o}^{new}$ denoted the new observation that is not yet part of $\mathbf{O}$. Each observation should have an age attached, such that when constraints are created for that observation they can be relaxed according to this age. $factor$ denotes

the speed at which constraints are to be relaxed, e.g. with a $factor$ of 10 a constraint would describe the entire normalized region after 10 relaxations.

Algorithm 4.3.3

- Increase the age of all observations in $\mathbf{O}$ by 1
- Remove any observation in $\mathbf{O}$ with an age above $factor$
- Add $\mathbf{o}^{new}$ with an age of 0, to $\mathbf{O}$

This policy has a time complexity linear in the number of dimensions making it operational. Furthermore, the policy allows for easy control of how many rotations needed before a constraint becomes obsolete. The downside of this policy is that the constraint may be relaxed faster with respect to some utilities than others.

4.4 Noise

In this section it is discussed how noise can be handled by Imputing by Comparison and Utility Iteration. When noise is the only dynamic behavior that can occur in the domain, the newly entered observation will generally be considered guilty, when conflicts occur. The argument is that if an observation contains noise, it will most likely cause a conflict immediately, at least when there is a large set of true observations already. However, a noisy observation may only conflict with a few and rare configurations of relevant pasts, and perhaps none of these have been observed before the noisy observation, in which case it will not cause a conflict. Furthermore, if a noisy observation is made before a large set of true observations is established, then it may not cause a conflict immediately. Therefore any policy designed to handle conflicts in domains containing noise should not immediately assume that the new observation is guilty.

Therefore this policy removes the constraints added by different observations, when conflicts occur, to see which observation is guilty. Constraints are only removed for one observation at the time, and if that does not solve the conflict they are reinserted. So this will, in the worst case be done once for each observation from which constraints were added before the conflict occurred. Once an observation is found that solved the conflict if removed, it is removed from the set of true observations.

Using this approach the order in which observations are removed to see if the feasible space becomes non empty, is very important. The policy for handling conflicts caused by noisy observations uses both an *oldest first* approach and a *newest first* approach, in an attempt to remove noisy observations without removing large numbers of non-noisy observations. It is important to note that when a conflict occur during execution of either Imputing by Comparison or Utility Iteration, the conflict is resolved using either the oldest or newest first approach, and then the execution continues. Meaning that the set of constraints is maintained, with exception of the newly removed constraints, and execution does not start over.

Algorithm 4.4.1 gives an overview of how conflicts are handled in a domain that contains noise. The two approaches, oldest first and newest first are described below. Every time a conflicts occurs, a counter is incremented, this counter is used to keep track of the number of conflicts that have occurred, it is called $conflicts$ and is initially set to zero. Initially the oldest first approach is used to solve conflicts, but when $conflicts$ reaches a predetermined limit ($limit$), the newest first approach is used instead. Now every time conflicts occur in a domain with noise, Algorithm 4.4.1 is run.

Algorithm 4.4.1

- **If** $conflicts < limit$

- **then** Increment *conflicts* by one, and run the oldest first algorithm
- **Else** Run the newest first algorithm

4.4.1 Oldest First

Because noise may not cause conflicts immediately, the first set of constraints to be removed are those added by the oldest observation, from which constraints could solve the conflict if removed. If constraints have already been added for the last decision node in all observations when the conflict occurs, the oldest of all true observation would be the one examined first. If it cannot solve the conflict then the second oldest is examined and so on. If not before, then the conflict will be resolved when the turn comes to the observation in which the conflict was discovered.

Doing this the first time a conflict occurs ensures that if the newest observation is not noisy but a noisy observation previously has gone undetected. If the new and the noisy observations conflict, the noisy observation will be deleted. There is a risk that inaccurate imputations could result in the new observation also conflicting with a second non noisy observation, in which case the two non noisy observations may conflict, sparing the actual noisy observation. For this reason it could be considered if this oldest first approach should be used more than for just the first conflict, i.e. *limit* should be greater than 1. Increasing the number of times the oldest first approach is used will increase the likelihood that noisy observations, already added to the set of true observations, will be removed. The trade off for increasing the number of times the oldest first approach is used, is that if the new observation is noisy then it will cause more correct observations to be deleted and there will be an increased risk that the noisy observation will be allowed to remain in the set of true observations.

Due to the facts that the frequency with which inaccurate imputations will cause conflicts and the frequency of noise is domain specific, the number of times (*limit*) the oldest first approach is best used will also be domain specific.

The oldest first approach should be implemented as Algorithm 4.4.2, where $\mathbf{O}$ denotes the set of all true observations and the different observations are denoted $(\mathbf{o}^1, \cdots, \mathbf{o}^m)$ where there are m observations in $\mathbf{O}$ and $\mathbf{o}^m$ is the oldest observation. Finally $\mathcal{C}$ denotes the set of constraints added before the conflict occurred.

Algorithm 4.4.2

- For $k = m$ to 1
  1. Remove the constraints added by $\mathbf{o}^k$ from $\mathbf{C}$
  2. **If** $\mathcal{C}$ describes the empty space
     - **then** insert the removed constraints from $\mathbf{o}^k$ into $\mathcal{C}$ again
     - **Else** remove $\mathbf{o}^k$ from $\mathbf{O}$, and halt this algorithm

4.4.2 Newest First

At some point, when the oldest first approach has been used a predetermined number of times (*limit*) and conflicts keep occurring, it is assumed that the new observation is the one causing the conflict. At this point the newest first approach is used instead. Using the newest first approach, the newest observation is examined by removing its constraints to check if the conflict is resolved, and only if this is the case the observation will be deleted. As the set of observations did not conflict before the new observation was made, removing it will most likely resolve the

conflict. If removing the newest observation does not resolve the conflict, then the conflict is occurring because the new observation is causing different imputations, and these imputations are causing the conflict. As can be seen in Sections 3.3 and 3.4, these situations are handled by removing the observation in which the conflict was discovered. This is also done in the newest first approach, when removing the newest observation does not solve the conflict.

The algorithm for the newest first approach is shown in Algorithm 4.4.3, where the same notation is used as in Algorithm 4.4.2. Furthermore, let $\mathbf{o}^{conflict}$ be the observation where the conflict was discovered.

Algorithm 4.4.3

- Remove the constraints added by $\mathbf{o}^1$ from $\mathcal{C}$
- **If** $\mathcal{C}$ describes the empty space
  - **then** insert the removed constraints from $\mathbf{o}^1$ into $\mathcal{C}$ again
  - **Else** remove $\mathbf{o}^1$ from $\mathbf{O}$, and halt this algorithm
- Remove the constraints added by $\mathbf{o}^{conflict}$ from $\mathcal{C}$
- Remove $\mathbf{o}^{conflict}$ from $\mathbf{O}$

CHAPTER

FIVE

Experiments

To evaluate the Imputing by Comparison and Utility Iteration methods, described in Section 3.3 and Section 3.4, several experiments have been conducted. These experiments were conducted to examine how well the imputing methods estimates utility functions, in static as well as dynamic domains.

The methods have been evaluated with regard to two different aspects: *speed* and *accuracy*. Speed is measured in the number of observations, so a method that reaches a higher level of accuracy with fewer observations is considered fast. Accuracy is measured in three different ways. The first is comparison of expected utility of the predicted strategy using the real utility values. The purpose of this measurement is to evaluate whether the choices predicted by the method would yield good decisions. This would be interesting when determining if the method should be used in some advisory role. The second measure of accuracy is how frequent a single decision is predicted correctly. This measure will be the fraction of the relevant past configurations where the method's estimated utility function would result in the same decision as the agents real utility function (ignoring those relevant pasts that cannot occur). As likely configurations of the relevant past are not given any greater weights than unlikely configurations, this indicates how well the method predict configurations that has only been observed a few time, if at all. To determine how accurately the method performs in general, a third measure is used where the predictions are weighed based on each configuration of the relevant past's probability of occurring. Measuring the accuracy of prediction for each decision would be interesting in a scenario where the observed agent is some sort of opponent, e.g. if the application using one of the methods should try to counter the action that the observed agent is about to make.

The weighed accuracy of predicting a decision node is calculated with Equation 5.1, where $correct_D(past(D))$ is one when the method is able to predict which choice the agent would make for decision D given the relevant past $past(D)$, and zero otherwise. If a decision node is a part of the relevant past, it is replaced with a deterministic chance node that have the same state space as the decision node, and enters the state that corresponds to the choice the agent would make.

$$\sum_{past(D)} \big(P(past(D)) \cdot correct_D(past(D))\big) \tag{5.1}$$

The unweighed accuracy of predicting a decision is calculated with Equation 5.2 where $|past(D)|$

is the number of possible configurations of the relevant past. Configurations that have zero probability are not included in this measurement.

$$\frac{\sum_{past(D)} \left(correct_D(past(D))\right)}{|past(D)|} \tag{5.2}$$

Rather than just comparing the two imputing methods against each other, the experiments will also be conducted using *FLUF*. This will make it possible to evaluate whether the new methods are improvements compared to *FLUF*.

FLUF works with a constraint removal policy to handle conflicting observations. The experiments in Hansen et al. (2004) revealed that between the "Always Guilty" and the "Innocent Until Proven Guilty" policies there was only minor differences in accuracy, with the exception of handling fluctuations in the utility values, in which case the "Always Guilty" policy recovered faster. For that reason experiments with *FLUF* in this project was only conducted using the "Always Guilty" policy. See Section 4.3.2.

The comparison between *FLUF* and the new methods is slightly uneven, as *FLUF* only keeps constraints whereas the new methods keeps observations and rebuilds the constraints for each prediction. This means that the new methods will use more space and be somewhat slower (in terms of execution time) than *FLUF*. To make the comparison more even and to reduce execution time and use of space, the methods will only use a number of the newest observations to create constraints, the number of observations chosen is called *window size*. The window size in these experiments has been chosen to be 100, forcing the methods to discard their oldest observations if they at any point have more than 100 observations, thus limiting the number of observations and thereby also the number of constraints being used. As *FLUF* removes constraints when conflicts occur all the constraints derived from one observation may eventually be removed, in which case that observation is considered as being removed.

5.1 Domain

The influence diagram used in the initial experiments is the same as used in Hansen et al. (2004) and is shown in Figure 5.1, the nodes have the number of states shown in Table 5.1.

Node	Number of states
A	4
OM	4
OQ	4
M	4
Q	4
M^*	4
H	7
OH	7
T	3

Table 5.1: Number of states

The utility values in node C are generally lower than those in node U. The reason for this is that decision node A could otherwise be dictated by utility node C, regardless of the configuration of the relevant past of A. With the chosen utility values the decision that yields the maximum expected utility in decision node A changes depending on the configuration of the relevant past of decision node A.

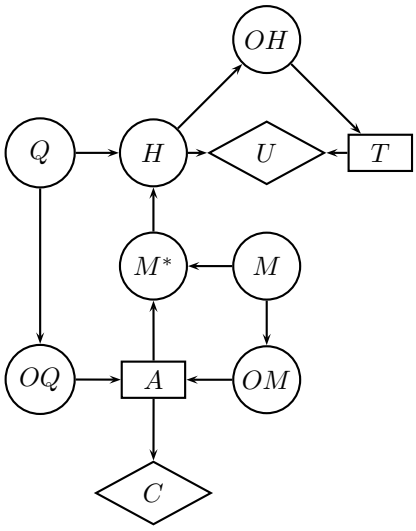


Figure 5.1: Influence diagram for experiments

5.2 The Experiment Program

The program used to perform the experiments on *FLUF* is the program develop in Hansen et al. (2004), with minor modifications. The library for solving linear inequalities is replaced by `lp_solve` (Berkelaar et al., 2005) due to a simpler interface. The implementation of *FLUF* is also extended to allow the use of the constraint relaxation policy for conflict handling, as described in Section 4.3.3. The two methods Utility Iteration and Imputing by Comparison is implemented as designed in Sections 3.3 and 3.4, including conflict handling policies, described in Section 4.2, for drift/fluctuation and noise along with the constraint relaxation policy. To handle the probabilistic networks Hugin Researcher from Hugin Expert A/S (A/S, 2004) is used.

With regard to execution time of the methods, *FLUF* is the fastest of the three in the original domain, but when the domains increase in size, *FLUF* decrease significantly in speed, (see Section 5.3.6). Naturally the exact time spent by each method will depend on the system on which the experiment is conducted. The execution times presented here were running on a 1.6 GHz Pentium M laptop. Experience have shown that Utility Iteration is almost as fast as *FLUF* in all experiments on the original domain. *FLUF* generally took just less than 30 minutes per 200 observations while Utility Iteration took just above 30 minutes. With regard to Imputing by Comparison, it took about two and a half hours per 200 observations in drifting domains, because several conflicts occur and observations are removed as a result, as later result will show, this is fast. Recall that the worst case time complexity of the Imputing by Comparison algorithm was polynomial in the largest number of possible configurations of the relevant past, in the domain, as shown in Equation 3.19, because the method grows more complex as more observations are entered, until the number of observations equals $\frac{\#relevant_i}{2}$. Since a window size has been introduced into the algorithm the complexity is reduced to $min(windowsize, \frac{relevant_i}{2})$, this is a significant decrease with respect to decision T, which has a relevant past of size 448. In spite of this reduction the Imputing by Comparison is very slow when observations are not regularly removed due to conflicts, which is the case in the static domain, single fluctuation and noisy observations experiments. In these experiments each run with 200 observations took 10 hours to complete using the Imputing by Comparison method, while using Utility Iteration and *FLUF* it took less than 40 minutes per 200 observations.

5.3 The Experiments

In this section the experiments themselves are presented. Different scenarios have been developed in order to test the methods under different circumstances. The scenarios are developed to represent a wide range of different possible situations where methods like *FLUF* , Imputing by Comparison and Utility Iteration could be used. Scenarios with both dynamic and static behavior are chosen. In the experiments dynamic behavior is categorized as either drift, fluctuation or noise as described in Chapter 4. To achieve reliable mean values and variance all experiments are run 10 times, and all experiments are performed with 200 observations in each run. Since the window size is only 100, then if the methods converge toward some mean values, they will do so before all 200 observations have been evaluated.

A number of experiments are conducted with the domain described in Section 5.1. The first experiment presented is conducted with a static domain. Then experiments are done on different kinds of drifting domains. Finally experiments with two kinds of fluctuation are presented followed by an experiment with a domain containing noise.

After these initial experiments have been presented, an experiment is presented where the domain has been slightly altered, to examine specific properties of the methods.

All results from the experiments are shown in Appendix D, and in this chapter only selected results are shown.

5.3.1 Experiment One - Static Domain

In this experiment the domain is static, meaning it does not change between the observations. The experiment serves as a baseline for how well the methods perform with regard to both accuracy and speed. Utility values have been chosen for the static domain such that different decisions should be made given different relevant pasts, when following the strategy for maximizing expected utility.

Even though the setup may seem simple it must be considered realistic. An example of such a situation could be something as buying office supplies as long as the prices does not change.

Results

The expected utility of the methods can be seen in Figure 5.2. *FLUF* starts with an expected utility of 0.86 but rapidly increases and reach 0.98 after 22 cases. *FLUF* does not improve after that. Utility Iteration and Imputing by Comparison starts with expected utilities of 0.87 and reaches 0.99 after 34 and 32 cases respectively. All methods have variance of less than 0.01 after the first 20 observations and throughout the remaining training cases.

For decision A *FLUF* starts with an accuracy of 0.6 and after only 2 training cases predicts 0.8 correctly (weighed). Unweighed the results are lower by approximately 0.03-0.05. Utility Iteration starts almost identically to *FLUF* but continues to increase its accuracy until 35 training cases have been entered. After that it has an accuracy between 0.97 and 1 throughout the remaining observations. When considering unweighed prediction, it takes Utility Iteration 30 additional training cases to reach the same level of accuracy. Imputing by Comparison behaves identically to Utility Iteration, except that after the first 60 observations Imputing by Comparison's accuracy remains about 0.02 below that of Utility Iteration, but only with regard to unweighed predictions.

For decision T *FLUF* starts with predicting 0.65 (weighed) correctly and reaches 0.98 after

50 cases. Utility Iteration starts at 0.7 and reaches 0.98 after 53 cases, but does, on average, predicts more accurately than *FLUF*. The accuracy of Imputing by Comparison starts at 0.64 and reaches 0.98 after 39 cases. After they reach 0.98 all methods maintain that accuracy. After 39 cases the variance of Utility Iteration and Imputing by Comparison decreases to 0.01 whereas *FLUF* does not reach that until after 71 cases. When considering decision T unweighed the methods performs almost equally well. They start with little more than 0.5 correct and reach 0.9 after 20-30 cases. They increase slowly after that but none of them increase above 0.98.

Conclusion

In general it must be said that Utility Iteration and Imputing by Comparison performs equally well, and that they both are faster and more accurate than *FLUF* for static domains. The higher level of accuracy that Utility Iteration and Imputing by Comparison reaches is most apparent for decision A, and the better expected utility must be attributed to that. *FLUF* cannot predict decision A as precise as Imputing by Comparison and Utility Iteration due to the fact that *FLUF* uses relaxations to create the constraints for the partial observed domain. These relaxations induce inaccuracies in the chosen utility point used to explain decision A (see Section 2.7). Imputing by Comparison and Utility Iteration are both more precise at predicting decision A can it be concluded that, for a static domain, imputing is a better method than the relaxation techniques of *FLUF*.

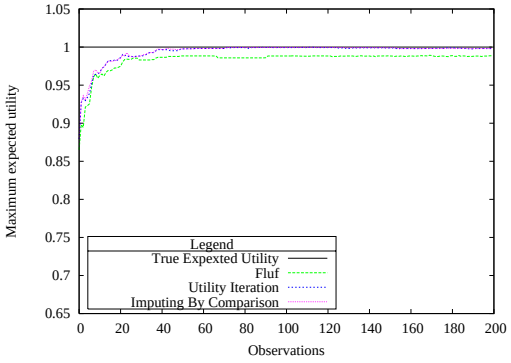


Figure 5.2: Expected utility for static domain (experiment 1)

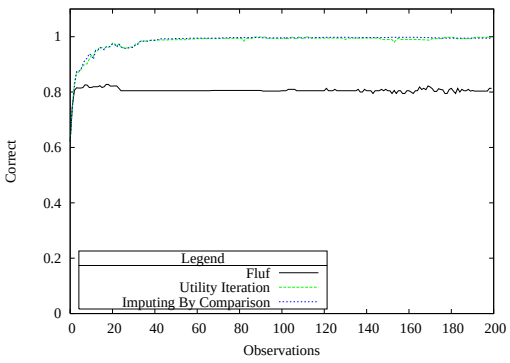


Figure 5.3: Prediction of decision A (weighed) (experiment 1)

5.3.2 Experiment Two - Domain with Drift

To test the methods with regard to domains containing drift, three different forms of drift was used, to examine if and how different kinds of drift would impact the accuracy and speed of the methods. The three kinds of drift are called *one way drift*, *random drift* and *local drift*. Ten experiments with 200 observations were conducted for each type of drift. Experiments using deletion policies was conducted with all three kinds of drift, while experiments were only conducted using the constraint relaxation policy under local drift.

One Way Drift In one way drift the utility values are drifting in the same direction between the observations, i.e. utility values that are increasing continue to increase, and decreasing utility values continue to decrease. This means that after a number of observations the strategy

of the agent may change, and it may happen again later as time progresses. But at some point, as the utility values continue to drift in the same direction, the strategy of the agent will not change anymore. It should be noted that this point is not reached within the 200 observations included in the conducted experiments.

The utility values that are decreasing will be denoted U^- and the increasing values will be denoted U^+. Now, between each observation the utility values will be updated as follows:

$\forall u \in U^+ : u := u + c$ where $c \in (0; 0.01)$ and $\forall u \in U^- : u := u - c$ where $c \in (-0.01; 0)$.

As the utilities are normalized between every observation a change of 0.01 is significant. Each utility value is increased or decreased by the same amount between each observation.

An example of such a scenario could be the development of prices of goods. The prices may for a period have a steady development, and the shopper's strategy will continue to be adjusted to these prices.

Local Drift With local drift the utility values drift within certain boundaries. These boundaries are that each utility at most may be 20% above or below its original value. The purpose of this experiment is to simulate that the agent is not completely sure about the domain and that the utility values actually are estimated values. This means that nothing necessarily changes in the environment, but just that the agent may judge situations differently from time to time.

When updating the utility values between the observations it is always relative to the original utility values. Let u be the original utility value, u' the utility value prior to the update and u'' the utility value after the update. Then the values are updated as:

$u'' := u' + c$ where $c \in (-0.01; 0.01) \wedge u' + c \in (0.8u; 1.2u)$.

The utility values have original values so that the 20% boundaries allow for more than one strategy. The c value is chosen randomly from the values that would satisfy the equation. Note that the utilities are normalized between observations and the maximum drift speed is 0.01, as in one way drift.

This could for example occur when the observed agent is trying to decide what kind of advertisement that should be used. The effect of each type of advertisements will probably be estimates.

Random Drift With random drift the agent's utility values also drift, like with local drift, but this time they are unbounded. They are, however, limited in how much they change between each observation. To avoid letting the drift becoming fluctuation, each utility value only drifts between -0.01 and 0.01 between each observation, i.e. a utility value can at most increase or decrease by 0.01. Having the same maximum drift speed as in local and one way drift also makes it easier to compare results.

Updating utilities is very similar to how the utility values are updated in local drift except that they are updated relative to the current value only.

$\forall u \in U : u' := u + c$ where $c \in (-0.01; 0.01)$

Again, the c value is chosen randomly from the values that would satisfy the equation, and the utilities are normalized between observations.

This could for example be customer's preferences based on what is fashionable. What is considered fashion may vary greatly over time.

Results

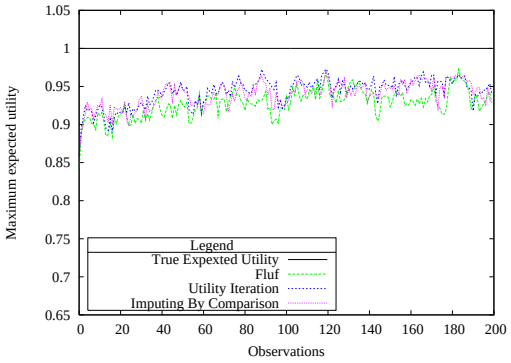


Figure 5.4: Expected Utility for One Way Drift (experiment 2)

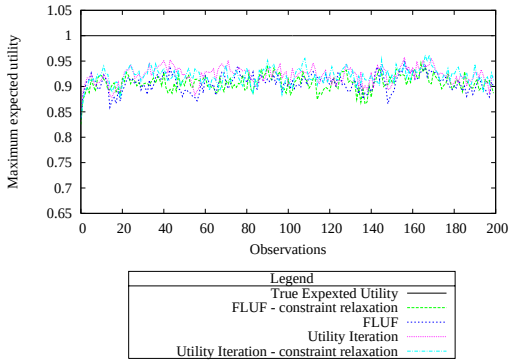


Figure 5.5: Expected utility for local drift with constraint relaxation policy (experiment 2)

The experiments show that all utility learning methods achieve a higher expected utility in the domain with one way drift (see Figure 5.4) than in the other two, and between local drift and random drift, the methods all handle local drift better than random drift. In one way drift there is an indication that the methods improve over time, converging around an expected utility between 0.92 and 0.96 after about 80 observations. In the two other kinds of drift the methods expected utilities increase only until observation number 20. In local drift the expected utility of the three methods are about the same after the first 20 observations, remaining between 0.86 and 0.96, but showing a tendency that *FLUF*'s expected utility is the smallest.

With respect to the accuracy of predicting decision A Utility Iteration and Imputing by Comparison are equal in all the experiments, *FLUF* only obtains equal result in local drift, while in one way and random drift *FLUF* performs worse than Utility Iteration and Imputing by Comparison. In local drift all three methods maintain a predicting accuracy around 0.7 with a variance around 0.2 throughout the experiment, for weighed as well as unweighed measurements. With respect to experiments with one way drift and random drift, Utility Iteration and Imputing by Comparison has an accuracy around 0.8. In one way drift the variance is mostly below 0.2 for Utility Iteration and Imputing by Comparison, while it is around 0.3 in random drift. The mean accuracy of *FLUF* in one way drift and random drift is around 0.6 while the variance in both cases is around 0.35 and goes as high as 0.4. When predicting decision T all methods work equally well for all three kinds of drift. The three methods having a mean accuracy around 0.6 in one way drift and random drift, with a variance around 0.15 for one way drift and 0.2 for random drift. With respect to local drift the three methods perform better, with a mean accuracy around 0.7 and a variance around 0.25.

An experiment was conducted where the constraint relaxation conflict handling policy was used in conjunction with each of the three methods in a domain with local drift. The same training cases were used in both this and the original experiments with local drift, to increase comparability. Figure 5.5 shows the expected utilities achieved by the three methods when using constraint relaxation, variances are not shown on the graph. The experiments showed that constraint relaxations works well with all three methods, but does not significantly improve the accuracy of any of the methods. All three methods predict both decisions almost equally well, with a mean accuracy around 0.9, very much like without constraint relaxation.

Conclusion

Throughout these drift experiments *FLUF* has, with or without constraint relaxation, shown expected utilities lower than the ones generated by Utility Iteration and Imputing by Comparison. Examining the accuracies of the methods with respect to prediction of decisions, it is obvious that the reduced expected utility of *FLUF* is a result of *FLUF*'s inability to predict decision A as well as the to imputing methods. The two imputing methods seem equal, with the exception of one way drift, where there is an indication that Utility Iteration achieves a higher mean on expected utility than Imputing by Comparison.

Concerning the constraint relaxation policy, there is no noteworthy change in the accuracy of the methods. Without constraint relaxation the constraints contribute to the feasible space by their original coefficients, whereas with constraint relaxation only the newest constraint contribute with its original coefficients. So one possible explanation of that the results does not vary much is that the dominant constraint is the newest, meaning that when not using constraint relaxation, a high number of constraints must be deleted. This is supported by the fact that the average number of observations kept by the methods is around 10.

5.3.3 Experiment Three - Domain with Fluctuation

Another kind of dynamic behavior is fluctuations where the agent being observed makes a radical shift in its strategy. How the methods handle this is tested in two different setups; *Single Fluctuation* which is a single shift in the strategy and *Multiple Fluctuations* where the agents change strategy very often. Experiments have been conducted using the deletion policy with all three methods in both kinds of fluctuation, while constraint relaxation has been used with all methods under single fluctuation and only with *FLUF* under multiple fluctuation.

Single Fluctuation For this setup the agent's utility values will make one great shift and otherwise remain unchanged. The main purpose of this is to assesses the methods' ability to handle a radical shift from one strategy to a completely different strategy. The shift in utility values is designed such that at least a third of the possible relevant past configurations lead to a changed policy.

This could represent a scenario where an agent in a poker game makes a sudden shift in the strategy to throw off opponents, or it may represent changes in a farmer's priorities if there is a sudden change in the weather, given that the domain does not explicitly model the weather development.

Multiple Fluctuations For this setup the agent's utility values will make several great shifts. The purpose of this is to determine how well the methods handle a very uncertain strategy.

This could represent a scenario where an agent is deliberately trying to prevent the method from determining the strategy.

Results

The expected utility for each method is shown in Figure 5.6. *FLUF* starts with a mean expected utility at 0.88 and reaches 0.96 after 10 observations in single fluctuation. The mean expected utility continues to increase until the strategy is changed at which the mean expected utility is above 0.98 (0.98 is in fact reached after only 25 observations, and only minor improvements

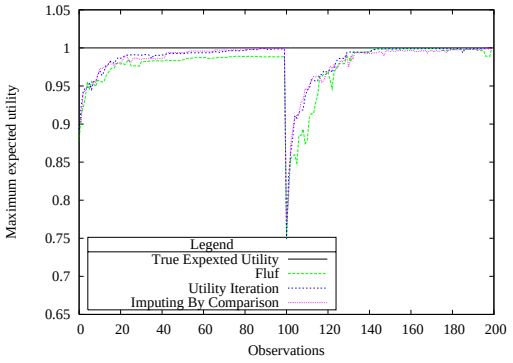


Figure 5.6: Expected utility for single fluctuation (experiment 3)

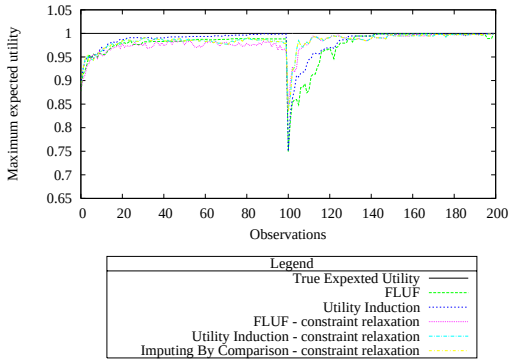


Figure 5.7: Expected utility for single fluctuation using constraint relaxation (experiment 3)

are made after that). After the strategy is changed it takes *FLUF* 30 observations to reach an expected utility of 0.98.

In multiple fluctuation *FLUF* starts with a mean expected utility at 0.88. The strategy is changed after 10 observations and as a result *FLUF*'s mean expected utility then drops to 0.64. It quickly recovers and gains a mean expected utility of 0.96 before the strategy is changed again. This pattern repeats itself throughout all 200 observations.

Both Imputing by Comparison and Utility Iteration follow the same pattern as *FLUF*. They generally have higher mean expected utility than *FLUF* and recovers faster than *FLUF* from a change in the strategy. In particular in multiple fluctuations, Imputing by Comparison and Utility Iteration methods recover faster than *FLUF*.

Concerning the prediction of the decisions all three methods are almost equally good at predicting decision T. In single fluctuation, the methods reach an accuracy of approximately 0.95 after 25 observations, with *FLUF* being five observations slower at reaching that level. When the strategy changes after 100 observations the methods drop to an accuracy of 0.2, but they then regain a level around 0.98 after additional 30 observations. For multiple fluctuation, Imputing by Comparison and Utility Iteration peaked between each change in the strategy at approximately 0.97, and then dropped to approximately 0.7. *FLUF* increased to, on average, 0.96 and dropped to 0.72 when the strategy was changed.

For decision A *FLUF* in single fluctuation, had an accuracy of 0.8 throughout the first 100 observations, whereas Imputing by Comparison and Utility Iteration reach 0.9 after 10 observations and 0.99 after 20 observations. After the strategy changes all three methods drops to around 0.65 in accuracy for decision A and then increases to 0.99 after additional 30 observations. For multiple fluctuation the methods' accuracy ranges from 0.2 to 1.0 at the most extreme. Which of the methods that recover quickest from a change in strategy varies each time the strategy varies.

The expected utility for each method using constraint relaxation is shown in Figure 5.7. For comparison *FLUF* without constraint relaxation is also shown. For single fluctuation the constraint relaxation technique described in Section 4.3.3 was tested. With regard to the mean expected utility *FLUF* reaches 0.96 after 20 observations where Imputing by Comparison and Utility Iteration reach 0.98 after 20 observations. Before the strategy changes the mean expected utility of all three methods are generally 0.02 lower than without constraint relaxation. After the change in strategy all three methods drops to around 0.83 in mean expected utility and then increases to 0.91 after observation 102 and reaches 0.96 after observation 105. The methods perform better after the change in strategy with constraint relaxation in that they

reaches the same level of accuracy as without constraint relaxation but faster.

Due to the results achieved for *FLUF* with constraint relaxation in single fluctuation, *FLUF* with constraint relaxation was also tested with multiple fluctuation. The mean expected utility of *FLUF* with constraint relaxation drops as low as 0.37 whereas *FLUF* without constraint relaxation only drops to 0.53. Both have a peak at 0.96. However, between the changes in strategy *FLUF* with constraint relaxation have a mean expected utility at 0.86 whereas *FLUF* without constraint relaxation have a mean expected utility of 0.8.

Conclusion

In general Imputing by Comparison and Utility Iteration performs equally well, both with regard to decision prediction and expected utility. Compared to *FLUF* they both have a higher mean expected utility and are generally faster at reaching high level of accuracy. The interesting element is how well the methods recover after the strategy changes. Here Imputing by Comparison and Utility Iteration are faster than *FLUF*, in particular in single fluctuation. One explanation for this might be that they not only throw away constraints when conflicts occur, but throw away all constraints related to the guilty observation. This means that number of constraints from old observations will decrease very fast when fluctuation occur.

With constraint relaxation the three methods performed almost equally well. Considering the results prior to the change in the strategy indicate that whether the methods work better with or without constraint relaxation may be domain specific. The methods increase the mean expected utility after the change in the strategy faster with constraint relaxation than without. The mean expected utility after the change in the strategy is almost identical for all three methods, indicating that for the utility function used in the last 100 observation the constraint relaxation is more important than how the constraints are created. The general improvement in mean expected utility is probably a consequence of old constraints quickly being made irrelevant by new constraints, as old constraints are relaxed.

After the change in strategy, the decision with the highest expected utility in node A is same, independent of the configuration of its relevant past. This allows *FLUF* to achieve a very high accuracy on decision A. The fact that a good mean expected utility is achieved faster after the change in the strategy is most likely a result of the increased accuracy on decision A.

5.3.4 Experiment Four - Domain with Noise

In this experiment the training cases contain noise, that is observations that does not necessarily conform with the utility values of the domain. Each new sample has probability p of being noisy; in this experiment p equals 0.05. The noise is introduced by creating an ordinary sample and then randomly picking three non utility nodes. Each of these three nodes are then put in a random state, which might be their original state. So in fact less than three nodes may be altered in noisy observations, and using this approach the noise introduced might actually conform with the agents utility function.

Noise could occur in almost any scenario, e.g. because of human failure to record correctly what is observed, or because of corrupted data due to computer failure, or faulty network transmissions.

Results

The expected utility of the methods can be seen in Figure 5.8.

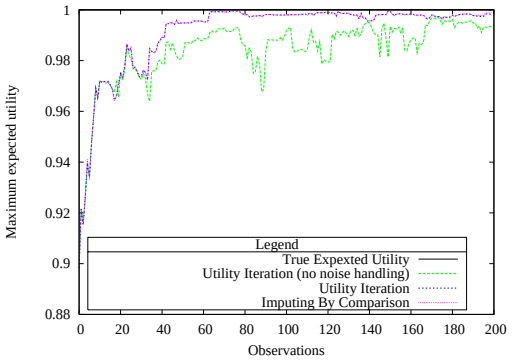


Figure 5.8: Expected utility with noise (experiment 4)

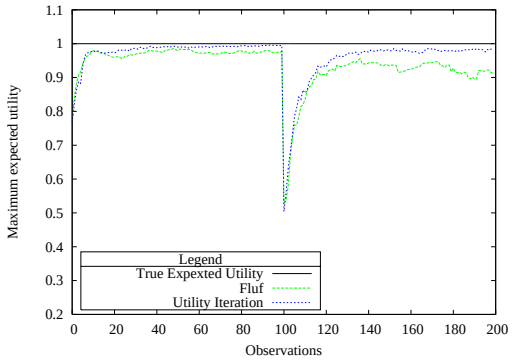


Figure 5.9: Expected utility with single fluctuation in alternative domain (experiment 5)

For this experiment the results of using Utility Iteration using the conflict handling policy for static domains, is included for comparison. Utility Iteration starts with a mean expected utility of 0.89 and then increases to 0.98 after 27 cases. Then its mean expected utility drops to almost 0.96 and for the remaining cases the mean expected utility varies between 0.97 and 0.99. Both Imputing by Comparison and Utility Iteration with noise handling start with a mean expected utility at 0.89 and increases slowly to 0.99 after approximately 40 cases. Neither of them drops significantly below that level.

For decision node A Utility Iteration without noise handling predicts correctly for 0.68 of the weighed configurations. Its ability to predict decision A varies between 0.95 and 0.99, and never really stabilizes. For decision T the accuracy varies between 0.9 and 0.99. However, it does not reach the same level as for Utility Iteration with noise handling.

Both Imputing by Comparison and Utility Iteration with noise handling start with predicting decision A correct for 0.68 of the weighed configurations and then continues to increase until they reach 0.99 after approximately 30 cases. After 30 cases both Imputing by Comparison and Utility Iteration remain at predicting correctly in approximately 0.99 of the weighed relevant past configurations. Decision T is predicted correctly in 0.95 of the cases after approximately 40 observations, and after that it increases to an accuracy of 0.98 after 65 observations.

Conclusion

The mean expected utility (see Figure 5.8) for the noise experiment indicates that the policy used by Imputing by Comparison and Utility Iteration to handle noise in the training cases, does take care of most of the noise. The results of the experiment show that Utility Iteration without noise handling takes a severe cut in its ability to get a good expected utility, compared to how it performs for static domains, while Imputing by Comparison and Utility Iteration with noise handling are largely unaffected. Imputing by Comparison and Utility Iteration with noise handling take about an additional 20 cases to reach almost the same level of accuracy for the decisions as for the static domain. This does have a minor impact on the mean expected utility. The fact that Imputing by Comparison and Utility Iteration are almost identical is not surprising considering that they also had almost identical results in the experiment with a static domain, and that they are using the same conflict handling policy.

5.3.5 Experiment Five - Alternative Domain

A question that has risen from experiments conducted so far, is why *FLUF*'s accuracy on decision A is lower than on decision T. It is examined if its relatively low accuracy may be due to poor estimations of the utilities in utility node C.

Considering that C is the dominant utility node, with respect to predicting decision A, the explanation could be that the only constraints created by *FLUF* that can be used to express anything about the utilities in node C are the constraints created in decision node A. All constraints created in decision node A are subjected to relaxation in *FLUF*, and this relaxation could be the cause of inaccuracies.

To examine this question an alternative domain is constructed, which is shown in Appendix C. This alternative domain is created as a modification of the original domain. Here a third decision node is introduced into the domain, called D, and a new chance node N is introduced as parent for D and C. D has indegree of 2 and outdegree of 1 with nodes OM and N as parents and Q as child, meaning that the edge from node OM to decision node A has now become obsolete due to the assumption of no-forgetting. The new decision will have a relevant past of size 12, and should due to an increased number of either relaxations or imputations be a difficult node to predict for all methods.

With respect to this alternative domain, tests are conducted with local drift and single fluctuation. Only one version of drift is used, in that the tendencies are expected to be the same for all three versions of drift, much as experienced in the earlier experiments. Single fluctuation is used, because the first 100 observations can be used as an indication of what would happen in a static domain, and the experiment will still allow for examination of the impact of altering the domain on fluctuation. Due to the increased number of imputations needed in the alternative domain, since decisions must be imputed for node A in that domain, only *FLUF* and Utility Iteration are run.

Results

The experiment on the alternative domain with local drift, showed that both Utility Iteration and *FLUF* achieved a mean expected utility around 0.87, which is a reduction with respect to the original domain. The accuracy of decision node A was reduced compared with results from the original domain, which in part explains the reduced expected utility. The mean accuracy of both methods lay around 0.6 with respect to decision A. So the reason why a lower expected utility is achieved alternative domain compared to the original domain, is found in decision node D. Both method have very large variances with respect to decision D, namely about 0.3 in both cases. The mean accuracy of Utility Iteration goes as low as 0.35 and as high as 0.85, while *FLUF* goes even lower at 0.29 and equally high at 0.85. With only 3 possible decisions in D, this accuracy is at time as bad as random guessing. Therefore the drop in mean expected utility must be attributed to the D predictions.

With respect to fluctuation on the alternative domain, both *FLUF* and Utility Iteration predicted T without any significant difference from the original domain. With respect to the A decision it was initially predicted better by *FLUF* in this alternative domain than in the original domain, the mean accuracy for *FLUF* was just below 0.9 up until the fluctuation. After the fluctuation *FLUF*'s mean accuracy stayed between 0.7 and 0.8, whereas *FLUF* could predict decision A with an accuracy of 1 after recovering from the fluctuations in the original domain.

Utility Iteration achieved a mean accuracy for predicting decision A of 0.96 both before and after the fluctuation, and it showed no difficulty recovering from the fluctuation, it actually

converged faster after the fluctuation than before. Concerning decision node D, Utility Iteration predicted it very well, achieving a mean accuracy of 0.95 both before and after the fluctuation converging equally fast. *FLUF* on the other hand had difficulties with decision node D, with a mean accuracy varying between 0.4 and 0.8 up until the fluctuation took place, showing no sign of improvement. Immediately after the fluctuation *FLUF* predicted D with an accuracy of only about 0.2, but after about 50 observations it seemed to recover to the same accuracy as before the fluctuation. Throughout the experiment *FLUF*'s accuracy had a variance of 0.35 when predicting decision D. The accuracy of decision D for single fluctuation is shown in Figure 5.11.

Despite *FLUF*'s accuracy on the D decision it achieved a mean expected utility around 0.97 before the fluctuation and 0.93 after, before the fluctuation *FLUF* converged after 10 observation and after the fluctuation it took about 30. Utility Iteration does better, with a mean expected utility around 0.99 before and 0.98 after, it should be noted that it took only 10 observations for Utility Iteration to converge before the fluctuation, but 40 observations to do so after. These results would indicate that *FLUF*'s inaccuracy in the D decision did not have a very large impact on expected utility, since the expected utility of the two methods was almost equal before the fluctuation. *FLUF*'s decrease in expected utility is more likely due to its reduced accuracy on the A decision, after the fluctuation. The expected utility for single fluctuation can be seen in Figure 5.9.

Conclusion

Concerning *FLUF*, this experiment indicates that the accuracy of *FLUF*, with respect to A, depends a lot on the domains used. In the fluctuation experiment, the change of strategy for the observed agent is the same in all 10 runs, allowing for a very high or low mean accuracy depending on how the chosen utilities fit a specific method. The varying accuracies observed during the fluctuating setup has probably more to do with the values chosen for utilities, than whether or not a fluctuation has occurred yet. Seeing as Utility Iteration achieves a high accuracy in fluctuation, unlike *FLUF*, indicates that the imputing method is more robust.

With respect to decision D, Utility Iteration handles it very well, under fluctuation, while *FLUF* only achieves an accuracy slightly better than random. Neither method achieves good accuracies on D in case of drift. The D node was included in an attempt to explain *FLUF*'s inaccuracy on the A decision. Since the expected utility of decision D, unlike decision A, is equally dependent on both utility nodes, *FLUF*'s inaccuracy cannot be explained only by inaccurate estimations on the utilities in node C due to relaxed constraints, since then A should have been more inaccurate than D. It turns out that accuracy has as much to do with the number of nodes between the decision node and its utility descendants as it does with good estimations of the utility values.

To get decision T right, it is almost enough for the methods to order the utilities correctly in utility node U, since T only depends on that utility node. Since the other parent of node U (H) is unobserved when a decision is made in node T, the relative size of the utilities becomes important since the expected utility of the decisions in T becomes a weighed average of the outcomes of H. So even if the order of the utilities are correct, the expected utility of the decisions in T can be ordered incorrectly if the relative size of the utilities in U are wrong, and this would result in a wrong decision. Now, since the uncertainties with respect to H are relatively low, due to the OH node, and because the constraints created at node T does not need to be relaxed, a high accuracy is often achieved by *FLUF* on the T decision.

This experiment indicates that the accuracy that is achieved in the A decision, is most likely a result of the A decision node's proximity to the C utility node. In the conducted experiment,

when the A decision is to be made, all other parents of C has already been observed, meaning that to calculate the expected utilities achieved from C for the decisions in A no averaging out is necessary. So had the expected utility of A only been dependent on C, then getting the order of the utilities in C right would be enough to get A correct. However the expected utility of A also depends on U, making the relative size of all utilities in the domain important when predicting A. The utilities in C become the most important since their differences are not being averaged out, as the utilities from U are. So the results observed in this experiment, where D is predicted poorly, indicates that the further a decision node is from utility nodes, the more accurate estimations on the utilities are necessary in order to be able to predict decisions.

In other words, a rough estimate of the utilities in U, that orders the utilities correctly, will be enough to predict decision T very well, but not necessarily completely correct. To predict decision A with a high accuracy, the estimates of utilities for both C and U must be quite good since the utilities from U are averaged out. In fact the more averaging out that is necessary, the better estimations will be needed to achieve a high degree of accuracy. The reason why the decision in node D is predicted so badly by *FLUF*, is most likely that utilities from both utility nodes are averaged out when calculating the expected utilities in node D, and *FLUF* only makes a rough estimate on the utilities in node C due to the relaxation done when constraints are created.

5.3.6 Experiment Six - Scalability

This experiment was conducted to examine the accuracy and execution time of the three methods, when the number of utilities increase. This was tested by letting the methods try to predict the behavior of an agent, modeled by a domain significantly larger than the original domain, this domain is called the *scalability* domain and is described in detail in Appendix C.

To briefly describe the scalability domain, it includes 4 decision nodes, 7 chance nodes and 3 utility nodes accounting for a total of 121 utilities (In the original domain there was only 25 utilities). Decision node $D2$ has a relevant past with 9 possible configurations, $D1$ and RD have a relevant past with 16 possible configurations and LD has a relevant past with 64 possible configurations.

Results

The experiments was not completed for the *FLUF* method, since it did not manage to evaluate even the first observation after 30 minutes on a 1.6 GHz Pentium M laptop. An execution time this poor can be explained by considering the worst case time complexity of *FLUF*, as determined in Section 2.6.2. The complexity of *FLUF* was determined to be $\mathcal{O}(nodes^{states \cdot utilities})$, so with 11 nodes in the domain and 121 utilities it is no surprise that the execution time of *FLUF* becomes extremely high.

Utility Iteration completes one run, containing 200 observations, in about 16 hours. After the first 10 observations the mean expected utility varies between 0.83 and 0.91 throughout the experiment with local drift. This is less than in the original domain, where the mean expected utility varied between 0.86 and 0.96, which indicates that the accuracy of Utility Iteration has decreased in the scalability domain. Decision node $D1$, which has 5 states, is predicted with a mean accuracy around 0.5, $D2$ which has 3 states is predicted with a mean around 0.7, LD has 4 states and is also predicted with a mean accuracy around 0.5 while RD which also has 4 states is predicted with a mean accuracy around 0.7. In the original domain, both A and T was predicted with an accuracy around 0.7 in local drift, so the results from this experiment are slightly worse explaining the lower expected utility.

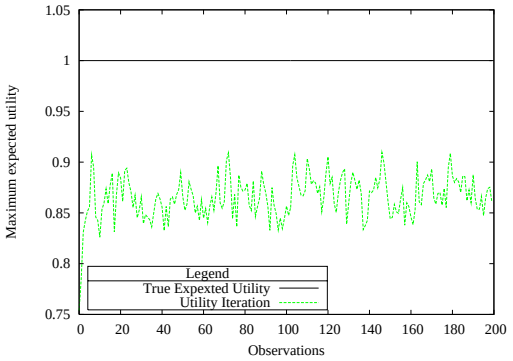


Figure 5.10: Expected utility for local drift (experiment 6)

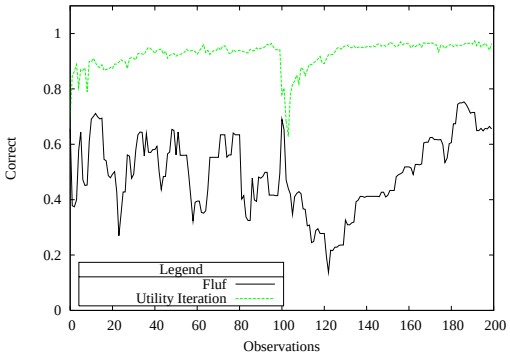


Figure 5.11: Prediction of decision D for single fluctuation (weighed) (experiment 5)

The expected utility is shown in Figure 5.10.

Decision $D1$ was predicted with an accuracy of only 0.5, even though it is the only parent of two of the utility nodes that has not been observed, when the decision is made. $D1$ is much like node A in the original domain, they can both with certainty determine the utility in early utility nodes, and they both influence the probabilities of the parents of the last utility node, i.e. the last utility node is a utility descendant. Furthermore the relevant pasts of the two nodes have the same number of configurations. The fact the the accuracy of node $D1$ is still 0.2 lower than A can, in part, be explained by node $D1$ having one more possible decision, but it is more likely due to the increased number of utility values. $D1$ must consider over 6 times as many utilities as A has to, and in the utility nodes that are children of the decision nodes, and therefore assumed dominating compared to the utility node that is far away, $D1$ has 40 utilities while A only has 4.

Decision node $D2$ is much like decision node T in the original domain. Both nodes are the last in the temporal order, parents of a utility node, have 3 possible decisions. Two big differences are that T has a relevant past with 416 configurations while $D2$ only has 9, and T has 21 utilities to consider while $D2$ has 81. In spite of these two differences the two nodes are pretty much predicted with the same accuracy, with a mean around 0.7, indicating that the difference in the size of relevant past, which should have given $D2$ a higher accuracy than T, is evened out by the difference in the number of utilities.

Only one run was completed for the Imputing by Comparison method, due to high execution time. The Imputing by Comparison method was faster than $FLUF$ though, since it after approximately 60 hours had evaluated all 200 observations, on the same $1.6GHzPentiumM$ laptop. The worst case complexity was found to be $\mathcal{O}(n \cdot |D|_{max} \cdot relevant^2_{max})$ in Section 3.4.3, the high execution time can at first seem strange, since the number of different relevant past configurations in the scalability domain is smaller than in the original domain, where Imputing by Comparison executed the corresponding experiment in two and a half hours. The explanation why Imputing by Comparison is still slower in this domain must be due the the increased number of decision nodes instead.

In that only one run was completed using Imputing by Comparison, the results are much less reliable than if 10 runs had been completed. The expected utility varies, between 0.68 and 0.99. It should be noted that, when comparing the number of observations retained by the method and the expected utility achieved, there is clear relationship. When the number of observations that the method is able to keep increases, then so does the expected utility. Actually, whenever the method has more than 10 observations its expected utility is above 0.9. With respect to

the accuracy of the decisions, it is difficult to conclude anything with respect to Imputing by Comparison, except to say that when comparing the accuracies obtained in the first run of Utility Iteration with the run in Imputing by Comparison, they were similar.

Conclusion

These experiments support the results from Section 2.6.2, where *FLUF*'s complexity is determined to be exponential in the number of utilities. It proved infeasible to conduct this experiment with *FLUF* in the scalability domain. With respect to Imputing by Comparison the execution time also increased, but even though the increase was less dramatic than for *FLUF*, not enough runs could be completed to achieve reliable measurements.

Furthermore these experiments showed that the execution time of Utility Iteration increased to 16 hours per 200 observations in the scalability domain. The experiment conducted with Utility Iteration showed a reduction in accuracy as the number of utilities increase, but while the reduction was significant, the method still achieved accuracies considerably better than random guessing would, and a mean expected utility just below 0.9, which is only slighty lower than for local drift in the original domain.

5.4 General Results

The experiments confirmed the result from Hansen et al. (2004) that *FLUF* was not able to predict decision A in the domain used for most of the experiments. The reason for this was investigated by trying to use *FLUF* on a modified domain where another decision node was added. The experiments indicated that the reason for *FLUF*'s poor prediction, of decision node A, was that chance nodes between the decision node and the U utility node made it difficult to establish the correct relationship between the utility values in node C and in node U. The imputing methods was more accurate in their prediction of decision node A.

During the experiments a window size of 100 observations was used to keep the methods reasonably fast (in execution time) and to make the comparison of *FLUF* and the imputing methods as even as possible. The experiments showed that the chosen window size actually had little effect. For the experiments with a static domain, the average number of observations any of the methods had in the windows were 75, over all 200 observations. Here it should be noted that with the static domain almost no observations were deleted, so after 100 observations, the windows held 100 observations. However, for the experiments with drifting domains and multiple fluctuations the methods had an average of approximately 10 observations in the window, and a maximum of 35 observations. For the noise experiment the methods had on average around 75 observations in the window, except for *FLUF* which had an average around 35. For the noise experiment and the single fluctuation the highest number of observations in the windows was 100. The single fluctuation experiment gave around 50 observations in the window on average for all the the methods.

This indicates that if the domain changes frequently the size of the window can be as low as 40 and still be used without any impact on the results, as the methods will delete observations aggressively. This also holds for the domains where the strategy of the observed agent changes rarely, as a window size of 40, would still be large enough for the results to converge.

CHAPTER

SIX

Conclusion

In this chapter the report is concluded. First some thoughts on how the utility learning methods in this project, can be considered in contexts outside that defined in the introduction, are presented in Section 6.1. Then the results and conclusions drawn throughout the project are summarized in Section 6.2. Finally possible subjects for future work, related to the work done in this project, are discussed in Section 6.3.

6.1 Perspective

In the first part of this section the concept of dynamic domains is discussed, along with the advantage of being able to handle dynamic domains. The second part considers the prerequisites for Imputing by Comparison and Utility Iteration, namely that they require that the probabilities of the variables in the influence diagram are known. *FLUF* allows differences between the probabilities in the influence diagram it uses and the one used by the agent. It is argued that the two imputing methods might handle differences in probability distributions as well as *FLUF*, even though they were not designed to.

6.1.1 Dynamic Domains

When comparing the method presented in Chajewska et al. (2001) with *FLUF*, Imputing by Comparison and Utility Iteration, the latter three methods are designed to work with less restrictive assumptions than the method presented in Chajewska et al. (2001), in that dynamic domains can be handled using different conflict handling policies. It should be noted that the conflict handling policies developed for *FLUF* can be used in conjunction with the method from Chajewska et al. (2001), since the two methods create the exact same constraints.

Considering the capability of handling dynamic behavior, it should be considered why dynamic behavior is observed. For example, if a soccer player normally plays defensively to conserve strength, but he exhibits drifting behavior since he over time begins to play more aggressively, then there is probably a cause for this drifting behavior. In the example the reason for the change in the player's tactics could be that he has been getting in better shape over time, and if his shape was not modeled in the domain then the change would seem as drifting behavior.

As in the example, it can be argued that all changing behavior is due to an incomplete model. From this perspective then all domains are static if they are modeled completely, i.e. all variables that influence decisions are taken into account.

The complexity of a model that literally takes all causalities into account can easily become so large that it is infeasible to represent it. It is not even certain that all causalities are known. So since it can be infeasible to model domains completely, then including as many variables as possible and accepting the apparent dynamic behavior is a possibility. In other words, a dynamic domain can be considered an approximation of the "real" domain, and the policies designed for the utility learning methods, presented in this project, actually increase their area of application into "real" domains of otherwise infeasible complexity.

The bigger the difference is between the "real" and the "modeled" domain, the more dynamic behavior should be expected. So when modeling some domain for use with a utility learning method, the relationship between the methods' complexity and ability to handle dynamic behavior should be considered, e.g. if the method is very good at handling dynamic behavior but has poor scalability then a simple model should be chosen.

6.1.2 Unknown Probability Distributions

The imputing methods are designed under the assumption that the probability distributions of the observed agent are known, whereas *FLUF* is able to handle differences between the probabilities used in the influence diagram it uses and the ones used by the agent. *FLUF* handles such differences by choosing a utility function that compensates for these difference so that the strategy of the agent is still predicted correctly. This compensation means that the utility function estimated by *FLUF* might not be very accurate, in the sense that the utilities are different from the agents utilities.

Since different probabilities will result in different utility coefficients, the created constraints will weigh the utilities differently than they should due to these probabilities. With several decision nodes in the domain, then the difference in probabilities may only influence the expected utilities in some of the decision nodes. If only a subset of the decision nodes are influenced by differences in probabilities, then the policies for those decision nodes would seem to follow a different set of utility values than the unaffected decision nodes. Therefore, when there are differences between the probabilities used by the agent and those used by *FLUF*, it might be impossible for *FLUF* to establish a set of utility values that predicts the observed strategy of the agent, because the utilities might be distorted by these differences with respect to only a subset of the decision nodes.

For a domain where the utility learning method does not have the same probability distributions as the observed agent, the imputing methods might be usable aswell. The reason for this is that no specific policy is needed to handle such situations, due to the way the imputing methods are designed they will implicitly compensate for inaccurate probabilities when estimating a utility function, just like *FLUF*.

6.2 Summary

In this project two methods similar to *FLUF* has been designed. The two new methods are called Utility Iteration and Imputing by Comparison, and share *FLUF*'s concept of generating a set of constraints based on observations to describe possible utility values. *FLUF* has been shown to be a viable prediction method in the past (Hansen et al. (2004)), but there are inaccuracies in the method that leave room for improvement. These inaccuracies are, to some

extent, due to the fact that the constraints describe a feasible space that is to large. Therefore methods developed during this project are based on the idea, that describing smaller spaces could increase accuracy. The main reason why *FLUF*'s utility space is to large is that relaxation of the constraints are done when the domain is not fully observed. The new methods avoid this relaxation by imputing the missing observations so that the domain becomes fully observed. The experiments conducted indicate that this results in a higher degree of accuracy and also requires fewer training cases. However, the two different ways of imputing used by the methods showed no significant difference from each other with respect to accuracy or speed.

In addition to using imputing as a technique to achieve higher accuracy, a technique called constraint relaxation has also been developed. The idea here is to avoid conflicts by relaxing the constraints as they grow older. The experiments showed that this worked well with *FLUF* and especially in domains with fluctuating utilities it enables *FLUF* to achieve better results faster.

Besides developing new methods to improve the accuracy of prediction, a policy to handle noisy observations was also developed. The technique for handling noise was developed so that it could be used together with the imputing methods, to increase their area of application. The experiments indicated that using the noise policy, the imputing methods handles domains with noise almost as well as static domains. However, it was only tested in one scenario and other domains and higher frequencies of noise may reveal some limitations of the noise handling technique.

Finally, it seems that imputing unobserved decisions is preferable to relaxing the constraints the way *FLUF* does it. Generally the imputation methods achieve more accurate predictions with fewer observations, no matter which of the imputing methods is used.

6.3 Future Work

The two new utility learning methods presented in this report, together with the constraint relaxation policy have made it possible to predict the behavior of an observed agent more accurately than *FLUF*, as it was presented in Hansen et al. (2004). However, the experiments conducted in this project have also shown areas that can be investigated and possibly improve the accuracy even further.

6.3.1 Handling Noise

The policy developed to handle noise in this project is based on assuming that when it is no longer possible to explain the behavior of the observed agent, it is because of noise or imputation error. The method does not try to determine if the individual observation was contaminated, meaning that it cannot determine if the guilty observations are causing conflicts due to imputation errors or noise. The experiments showed that the way the observations are removed works reasonably well.

It could be possible to integrate noise handling with a method for handling drift, by evaluating comparing new observation to the true observations already made, thereby determining the likelihood of the new observation. This could for example be done by using the utility values estimated before the new observation was made. Doing this, it would be possible to discard observations that seem unrealistic compared to the expected utility of the observed decisions.

To measure if some observation is realistic, using the utility function that was estimated before the observations was made (called V_{old}), then the expected utility of each of the decisions made

in the new observation could be compared to the maximum expected utility of that decision node when using V_{old}. With a large deviance in the expected utility of one or more decisions, the new observation could be categorized as noisy and ignored. If only some of the decisions in the observation yield a large difference in expected utility, then it should be considered whether the entire observation should be discarded, or if the constraints from some of the decisions could still be considered reliable. In any case it wold also have to be considered how large the divergence in expected utility would have to be, for the observations to be categorized as noisy.

The reason why this policy cannot be used in domains with fluctuation, is that the first observations after a fluctuation could easily yield low expected utilities with respect to V_{old}, without being noisy. A conflict handling policy using such as the one suggested here, would have to take into account that with very few observations the next observation might easily seem unrealistic, even if it is not noisy.

6.3.2 Complexity

Every time constraints are generated, the different formulae presented throughout the report are used by the utility learning methods. In *FLUF* most of the execution time is spent calculating these constraints, while the imputing methods spend time imputing virtual observations as well. As the probabilities are considered static, it is possible to reduce the number of calculations needed to generate constraints. Instead of calculating coefficients every time constraints are generated for some decision with some relevant past, they could be saved the first time they are calculated, so that later calculations would not need to compute the same coefficients. Such an approach would benefit all methods, but it would be a space for speed tradeoff and the memory consumption would be higher than the naive implementation. Whether it is worth it would depend on the system doing the calculations.

6.3.3 Missing Data

FLUF, Imputing by Comparison and Utility Iteration all assume that each observation shows the state of all decision nodes and all chance nodes, prior to the last decision node. Just as it is possible that some of the observations are contaminated with noise, it could also happen that some of the states of the nodes are lost. An extension to the methods described in the report could be to handle such cases.

One way of handling missing data could be to simply discard the observations with missing data. A couple of drawback with this approach would have to be considered however. There might be so many observations containing missing data, that the prediction method will only keep very few true observations. Another problem when simply removing observations, is that if specific configurations are more likely to contain missing data than others, then the prediction method can become biased since the constraints that would have been added in those configurations are never considered. If neither of these problems occur however, then it is not unrealistic that deleting observations would be a good strategy, as the methods in general are very fast, meaning that they get close to the real utility function with very few observations.

Another way of handling missing data could be to instantiate the missing nodes, and then create constraints from the observed decisions as normal. The chance nodes could be instantiated in their most likely state, given the configuration of their parents and children. Decision nodes where the unobserved node is part of the relevant past could also be included. If decision nodes are to be included in this calculation, the policy those nodes are assumed to follow,

or perhaps even a temporary utility function, should be available. Missing decision nodes could be instantiated based on what would yield the highest expected utility given the current utility function. As with missing chance nodes the outcome of the missing decision node's children, and any other decision nodes where the missing node is in the relevant past, could be considered. Even though data might not be missing at random, this approach could instantiate missing nodes correctly, if enough can be learned about the utility function. The drawback of using such an approach, is that it could potentially reinforce the already predicted strategy which might be wrong.

6.3.4 Improved Comparison

As Imputing by Comparison only compares the distributions of the hypothesis variables given the chosen decisions, any information that could have been used from the discarded decisions is lost. Imputing by Comparisons accuracy could be increased by including this discarded information in its comparison technique.

Currently comparison is done only with respect to the Euclidean distance between the joint distribution of a set of hypothesis variables, given the different true observations and the possible virtual observations. However, in each true observation a set of decisions were discarded in favor of the chosen decision, the discarded decisions would have resulted in different joint distributions over the hypothesis variables, these are called the *discarded distributions* in that observation. When creating virtual observations for some decision node with n states, then n different virtual observations are possible. In each virtual observation the set of discarded distributions will correspond to the distributions that would have been generated by the $n-1$ other virtual observations.

A problem with the current comparisons, is that a virtual observation can yield a distribution over the hypothesis variables that is very close to a distribution yielded by a true observation, while one of the discarded distributions in that virtual distribution would in fact have yielded a higher expected utility. As an example, if imputing a virtual observation of a decision, where the relevant past allow for high expected utilities, then the worst decision might result a distribution on the hypothesis variables that is much like the distribution induced by the best decision in a different relevant past that has already been observed.

So in some respect the true observation that is the most like a virtual observation, is the one where the set of discarded distributions, as well as the distribution induced by the observed decision, yield short Euclidean distances to the corresponding distributions from the virtual observation. Discarded distributions could be compared to ensure that no discarded distribution in the virtual observation yields a higher expected utility than the chosen distribution. So discarded distributions in the virtual observation should somehow be compared to distributions from a true observation, to determine whether they yield a smaller expected utility than the distribution of the chosen decision in that virtual observation.

This can be examined in two steps, by first investigating if the discarded distributions yield smaller expected utilities than a decision chosen in a true observations, and then investigating if the chosen distribution in the virtual and true observations are alike.

For the first step, then if all discarded distributions, in a virtual observation, have short Euclidean distances to at least one discarded distribution in some true observation, it is an indication that they yield about the same expected utility as that discarded distribution, and therefore less than the distribution of the observed decision in that true observation. This means that it is no problem if there are some discarded distributions in the true observations with a large Euclidean distance to all discarded distributions in the virtual decision, as long as it is true for all discarded distributions in the virtual observation. This also means that

the Euclidean distance should be calculated with respect to all discarded distributions in the true observation for each discarded distribution in the virtual observation, to find the shortest distance for all distributions in the virtual observation. This results in $(|D|-1)^2$ calculations, where $|D|$ is the number of decisions in the node with which the imputation is concerned. So the measure of how close two sets of discarded distributions are from each other, a formula much like the one shown below in Equation 6.1 could be used. In the formula δ_v is the chosen decision in the virtual observations, while δ_t is the decision chosen in the true observation.

$$\frac{1}{|D|} \sum_{d_v \in D/\delta_v} min_{d_t \in D/\delta_t} \left(EC(P(H|d_v), P(H|d_t))\right) \tag{6.1}$$

If the discarded distributions are close and the distributions induced by the chosen decisions in the two observations simultaneously yield a short Euclidean distance to the distributions of each other, then this in an indication that the decisions yield about the same expected utility. Meaning that the chosen decision in the virtual observations is likely to be the optimal decision. It should be noted that when a true observation is used for comparison, then, since the chosen decision is only a factor due to its impact on the distribution, it should be compared with all the possible virtual observations, as it is the distributions that determine which observations are alike. This means that each true observation should be compared with $|D|$ different virtual observations.

Incorporating these measurements in the comparison, could reduce the risk that decision that are not optimal are chosen, thereby increasing the likelihood of imputing correctly.

APPENDIX

A

Placing Constraints

In this section the constraints generated for a fully observed strategy will be examined closer. Note that when imputing missing observations the strategy becomes fully observed, so the result described here will also apply for the constraints generated by Imputing By Comparison and Utility Iteration.

The proposition presented here generally says that any constraints generated for a fully observed strategy will always intersect the diagonal.

In order to express and prove the proposition, in Theorem A.1, some notation is needed. Let C be a constraint and $\sum_{c_i}$ be the sum of all the coefficients of the utility values for the $i'th$ utility node. Also let $|U|$ be the number of utility nodes in the domain.

Assuming that a strategy is fully observed and constraints are generated according to Equation 2.4, the following proposition will hold.

Theorem A.1 *Let C be any constraint generated for a fully observed strategy. Then the following will hold for that constraint:*

$$\bigwedge_{1 \leq i \leq |U|} : \sum_{c_i} = 0$$

Proof (Theorem A.1) Let D be the last decision node in the temporal order. Then for some observation where D has been observed in state δ_D for some relevant past $past(D)$ and d' is some other state for D, each constraint will be of the following form:

$$\sum_{I_n} P\big(I_n|\delta_D, past(D)\big)V\big(pa(u)\big) \geq \sum_{I_n} P\big(I_n|d', past(D)\big)V\big(pa(u)\big)$$

$$\Updownarrow$$

$$\sum_{I_n}\left(P\big(I_n|\delta_D, past(D)\big)\sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\right) \geq \sum_{I_n}\left(P\big(I_n|d', past(D)\big)\sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\right)$$

$$\Updownarrow$$

$$\sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\sum_{I_n} P\big(I_n|\delta_D, past(D)\big) \geq \sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\sum_{I_n} P\big(I_n|d', past(D)\big)$$

As the expression under $\sum_{i=1}^{|U|}$ is the summation of the probabilities of each parent configuration of the $i'th$ utility node, each expression will sum to one. These probabilities are also the coefficients for each utility value, so for each expression in the inequality it holds that:

$$\sum_{I_n} P\big(I_n|d, past(D)\big) = 1$$

Where d is any decision from D. When subtracting the expression on the right side from both expressions the result is:

$$\sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\cdot\sum_{I_n} P\big(I_n|\delta_D, past(D)\big) - \sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\cdot\sum_{I_n} P\big(I_n|d', past(D)\big) \geq 0$$

$$\Updownarrow$$

$$\sum_{i=1}^{|U|} V_i\big(pa(u_i)\big)\cdot\left(\sum_{I_n} P\big(I_n|\delta_D, past(D)\big) - \sum_{I_n} P\big(I_n|d', past(D)\big)\right) \geq 0$$

where

$$\bigwedge_{1\leq i\leq|U|} : \sum_{c_i} = 0$$

□

An important consequence of Theorem A.1 is that all constraints created from a fully observed strategy will intersect where all utility values from the same utility node are equal. Note that when this is the case, the utility values will describe the trivial utility function.

Whenever the feasible space becomes empty, it means that at least two constraints intersect. Since all constraints are linear and always intersect in the trivial utility function they can intersect nowhere else, unless they lie on top of each other.

APPENDIX

B

The Accurate Technique - Utility Iteration

Based on the extended technique described in Section 3.3.1, an accurate technique is described in this appendix that will always find a utility function satisfying all observations. Basically the accurate technique will do the same as the extended technique, but furthermore the accurate technique will maintain a list of alternative imputations that was not done, and when conflicts occur it will iterate backwards imputing differently to avoid the conflict. This technique is more accurate than the extended technique, but it is shown in this section that the time complexity becomes too high for the technique to be operational.

When constraints are created at some decision node, D_k, in some observation o, then imputations are done for all $(D_i|i > k)$, yielding a set of policies $\Sigma = (\sigma_{k+1}, \cdots, \sigma_n)$. The number of different Σ that are possible after decision node D_k is $(number_{\sigma_{k+1}} \cdot \ldots \cdot number_{\sigma_n})$, where $number_{\sigma_i}$ is the number of different policies for decision node D_i and n is the number of decision nodes in the domain.

In the accurate technique a policy set is not found by choosing a point in the feasible space. Instead all possible combinations of policies are used to create constraints, and separately these constraints are inserted into the feasible space, to examine if it becomes empty. Now, for decision node D_k in observation o, a list is created containing all sets of policies, $L_{k,o} = (\Sigma_1, \cdots, \Sigma_m)$, that did not make the feasible space empty. The intuition is that, if all previously created constraints are correct, then the correct imputation, for $(D_i|i > k)$ in o, must be a member of $L_{k,o}$. There is a significant difference between the two first techniques and the accurate technique, in that the temporary utility function is no longer used to choose a specific imputation, but to exclude a set of imputations instead.

When the list of possible policy sets has been created, e.g. $L_{k,o} = (\Sigma_i, \Sigma_{i+1}, \Sigma_{i+2})$, then the constraints created using Σ_i are used, and the technique proceeds to the next observation. There is no reason why Σ_i is chosen above the others, since they are all equally valid, but one must be chosen for the technique to proceed. The other Σ may still be used, in case backtracing becomes necessary.

The algorithm for the accurate technique is shown in Algorithm B.0.1, which described how $L_{k,o}$ is found and how constraints are created, and in Algorithm B.0.2, which describes how backtracing is done when conflicts occur.

In Algorithm B.0.1 o_p is the true observation in which the constraints are being created for decision node D_k. The observation has a subscript, p, which indicates the observations number in the order of observations, and m is the total number of observations. Recall the order of observations is irrelevant as the domain is assumed to be static, but to evaluate the observations sequentially the order becomes necessary.

$\mathcal{C}_C$ is the set of constraints describing the feasible space, before D_k is evaluated in observation o_p. $O_{imputed}$ is the set of virtual observations and O_{true} describe the true observations, the elements in these sets consist of a relevant past and the decision made in a decision node. Initially $O_{imputed}$ is empty.

Algorithm B.0.1

1. Let δ_{D_k} be the observed decision of D_k in o_p
2. Let $\mathcal{C}_{k,o_p} = \mathcal{C}_C$ and $O_{imputed_{k,o_p}} = O_{imputed}$
3. Let L_{k,o_p} be a list of all policy sets (Σ) over the nodes $D_i|i > k$, that are consistent with $O_{imputed}$ and O_{true}
4. **For all** policy sets (Σ) in L_{k,o_p}
   - Create an empty set of constraints called $\mathcal{C}_\Sigma$
   - **For all** configurations of the relevant past of decision nodes $D_i|i \geq k$, (o_i) consistent with o_p, for which constraints has not yet been added
     - Replace $D_j|j > i$ with chance nodes, C_j, according to Σ
     - Where δ_{D_i} is the decision dictated by Σ given past o_i, add the following constraints to $\mathcal{C}_\Sigma$: $\forall_{d \in D_i \setminus \delta_{D_i}} : \rho_{D_i}(\delta_{D_i}, o_i) > \rho_{D_i}(d, o_i)$
     - Return the chance nodes C_j to the original decision nodes D_j
   - **If** the set of constraints $\mathcal{C}_\Sigma \cap \mathcal{C}_C$ describes the empty space
     - **then** remove Σ from L_{k,o_p}
5. **If** L_{k,o_p} is empty
   - **then** call Algorithm B.0.2, and halt this algorithm
6. For the first set of policies (Σ_{first}) in L_{k,o_p}
   - Add imputed decisions in Σ_{first} to $O_{imputed}$
   - Add constraints created at step 4 using Σ_{first} to the set of constraints U_C
   - Remove Σ_{first} from L_{k,o_p}
7. Save L_{k,o_p}, $\mathcal{C}_{k,o_p}$ and $O_{imputed_{k,o_p}}$ (These are used by Algorithm B.0.2)
8. **If** $p \neq m$
   - **then** call this algorithm recursively for the next observation o_{p+1} and decision D_k
   - Halt this algorithm
9. **If** $p = m$ and $k \neq 1$
   - **then** call this algorithm recursively for the first observation o_1 and decision D_{k-1}
   - Halt this algorithm
10. **If** $o = m$ and $k = 1$

- **then**, **if** all true observations conform with the utility function described by the chosen utility point in $\mathcal{C}_C$
  - **then** the Utility Iteration algorithm is done
  - **Else** the backtracing algorithm is called (Algorithm B.0.2)

It will always be possible for Algorithm B.0.1 to find a utility function that conform with all observations. No matter the domain and the observations made, it may happen that all imputations made during execution are correct, i.e. the decisions imputed are the same that the observed agent would have made. Given such a set of perfect imputations, then the constraints (C_o) created by some observation o, will describe a space ($space_o$)in which o conforms with all utility functions described by points in that space. So with a set of different observations $(o_1, \cdots, o_m)$, each having created a set of constraints with which they conform $(C_{o_1}, \cdots, C_{o_m})$, this algorithm would describe a feasible space by the constraints $(C_{o_1} \cup \cdots \cup C_{o_m})$, which is the space $(space_{o_1} \cap \cdots \cap space_{o_m})$, or in other words the space where all points will conform with all observations $(o_1, \cdots, o_m)$.

It is unlikely that Algorithm B.0.1 will guess exactly the correct decision at every imputation, this is where the backtracing algorithm comes in, see Algorithm B.0.2. The backtrace algorithm is called if the space becomes empty at some point during execution of Algorithm B.0.1 or if the utility function found by Algorithm B.0.1 does not conform with all observations. The backtrace algorithm steps backwards through the imputations made by Algorithm B.0.1, until it finds an observation o_p in which the imputations done to create constraints for some decision node (D_k) could have been done in another way, i.e where L_{k,o_p} is not empty. After finding such a combination of observation and decision node, denoted (o_p, D_k), the backtrace algorithm create constraints for (o_p, D_k) according to one of the alternative imputations, and then starts Algorithm B.0.1 again.

The notation in Algorithm B.0.1 is used in the backtrace algorithm as well. When the algorithm is called, then all previously examined combinations (o_p, D_k) will have saved a list of the alternative sets of policies that could have been imputed, $(L_{k,o})$, and a set of constraints describing the feasible space $(\mathcal{C}_{k,o})$ as well as the decision already imputed $(O_{imputed_{k,o_p}})$, when they were examined $(\mathcal{C}_{k,o})$, see Algorithm B.0.1 step 11.

Algorithm B.0.2

1. Let (o,D_k) be the chosen combination that have alternative Σ's in list $L_{k,o}$
2. Remove the first element, Σ_1, from $L_{k,o}$
3. Create an empty set of constraints called $\mathcal{C}_{\Sigma_1}$
4. **For all** configurations of the relevant past of decision nodes $D_i|i \geq k$, (o_i) consistent with o_p, for which constraints has not yet been added
   - Replace $D_j|j > i$ with chance nodes, C_j, according to Σ_1
   - Where δ_{D_i} is the decision dictated by Σ_1 given past o_i, add the following constraints to $\mathcal{C}_{\Sigma_1}$: $\forall_{d \in D_i \setminus \delta_{D_i}} : \rho_{D_i}(\delta_{D_i}, o_i) > \rho_{D_i}(d, o_i)$
   - Return the chance nodes C_j to the original decision nodes D_j
5. Set $\mathcal{C}_C = C_{\Sigma_1} \cup \mathcal{C}_{k,o}$
6. Set $O_{imputed}$ to $O_{imputed_{k,o_p}}$
7. Add imputed decisions in Σ_1 to $O_{imputed}$

8. Call Algorithm B.0.1 for the observation and decision node that succeeds the combination (o, D_k) (Either (o_{p+1}, D_k) or (o_1, D_{k-1}))

Algorithm B.0.1 has a time complexity, with respect to (o_p, D_k), that is $(|D_{k+1}| \cdot (relevant_{k+1} - relevant_{true,k+1})) \cdot \ldots \cdot (|D_n| \cdot (relevant_n - relevant_{true,n}))$, where $relevant_i$ is the number of different relevant past configurations possible for decision node D_i, $|D_i|$ is the number of different decisions in the node and $relevant_{true,i}$ is the number of different configurations of the relevant past observed for node D_i. In other words the complexity for Algorithm B.0.1, when examining decision node D_k in some observation, is the number of different policy sets for the nodes $D_i | i > k$, consistent with all true observations. This is because the task of creating the constraints $\mathcal{C}_\Sigma$ and comparing them with $\mathcal{C}_C$, is done for all Σ consistent with the true observations.

The worst case complexity of Algorithm B.0.2 is only $\mathcal{O}((n-1) \cdot relevant_{true})$, being the maximal number of steps backwards the algorithm can take, where $relevant_{true}$ is the number of true observations. For comparison the worst case complexity of Algorithm B.0.1 is $\mathcal{O}((relevant_{max} \cdot |D|_{max})^{n-1})$, where $relevant_{max}$ is the highest number of different configurations a relevant past can have in the domain and $|D|_{max}$ is the highest number of different decisions one decision node can have.

For both algorithms, they will run $\mathcal{O}(policy\_max^{combinations})$ times, in the worst case. Where $combinations = relevant_{true} \cdot (n-1)$ is the number of different combinations of observation and decision node where imputations are needed and $policy\_max = relevant_{max} \cdot |D|_{max}$ is the highest number of different policies a decision node can have.

When describing the worst case complexity of the accurate method Algorithm B.0.2 becomes irrelevant since its complexity is linear while the complexity of Algorithm B.0.1 is exponential in the number of decisions. The worst case time complexity of the entire Utility Iteration method, using the accurate technique, is expressed in Equation B.1. As can be seen from the equation, the time complexity of the entire algorithm becomes exponential in both the number of decisions and the number of true observations.

$$\begin{aligned} &\mathcal{O}((relevant_{max} \cdot |D|_{max})^{(relevant_{true} \cdot (n-1))} \cdot (relevant_{max} \cdot |D|_{max})^{n-1}) \\ &\Updownarrow \\ &\mathcal{O}((relevant_{max} \cdot |D|_{max})^{relevant_{true} \cdot n}) \end{aligned} \tag{B.1}$$

APPENDIX

C

Domains

During the experiments, described in Chapter 5, two domains are briefly introduced in that chapter and experiments were conducted using these. These domains are called the *alternative* domain and the *scalability* domain. The first domain was used in the experiment described in Section 5.3.5 while the second domain was used in the experiment described in Section 5.3.6.

C.1 The Alternative Domain

The alternative domain is a modification the original domain, with a number of extra nodes inserted to examine the reason why the accuracy of A was lower than T. An extra decision node, called D, was added. D was added to examine if the reason why the accuracy of A was lower than T was due to A being very dependent on a utility node that was poorly estimated. To make D equally dependent of C and U a chance node N was inserted, so that if D was predicted better than A, it would be an indication that C was estimated poorly. The alternative domain is shown in Figure C.1.

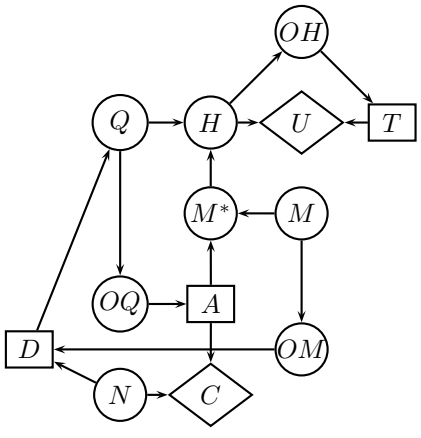


Figure C.1: The second alternative domain

C.2 The Scalability Domain

The scalability domain was introduced in Section 5.3.6, and was designed to test how well the different methods performed when the number of utilities in the domain grew. It contains 7 chance nodes, 4 decision nodes and 3 utility nodes, with the number of states shown in Table C.1. As the table shows, the total number of utilities become 121. The domain is shown in Figure C.2.

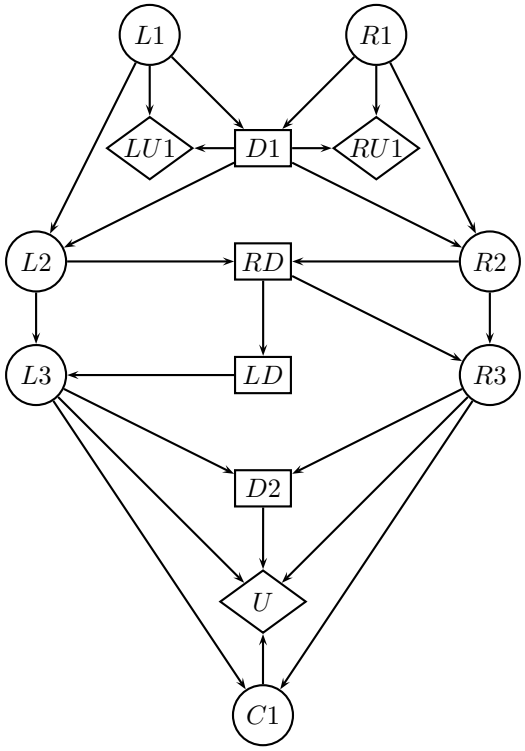


Figure C.2: The scalability domain

Name	Node type	No. States
$L1$	Chance Node	4
$L2$	Chance Node	4
$L3$	Chance Node	3
$R1$	Chance Node	4
$R2$	Chance Node	4
$R3$	Chance Node	3
$C1$	Chance Node	3
$D1$	Decision Node	5
RD	Decision Node	4
LD	Decision Node	4
$D2$	Decision Node	3
$LU1$	Utility Node	20
$RU1$	Utility Node	20
U	Utility Node	81

Table C.1: Number of states in nodes

APPENDIX

D

Results

In this appendix the results from the experiments described in Chapter 5 are shown. Measurements were done for every method in every experiment on the expected utility and weighed accuracy of decision predictions as well as unweighed. In this appendix the expected utility and the weighed decision prediction accuracies are shown. Unweighed accuracy is not shown, as it in all experiments resembled weighed accuracy, only a bit lower. Variance is included in the graphs, so to ensure that the graphs can be easily read, each graph will only contain one set of results.

D.1 Static Domain

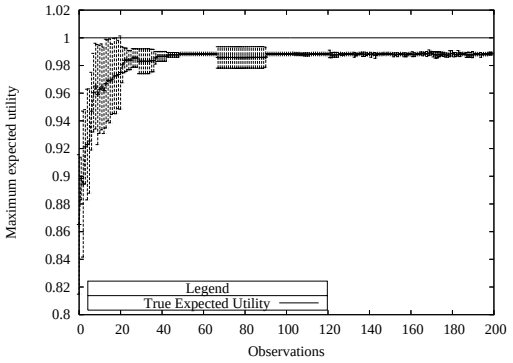


Figure D.1: Expected Utility for *FLUF* in static domain

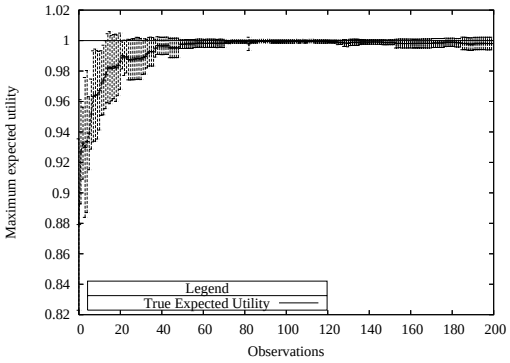


Figure D.2: Expected Utility for Utility Iteration in static domain

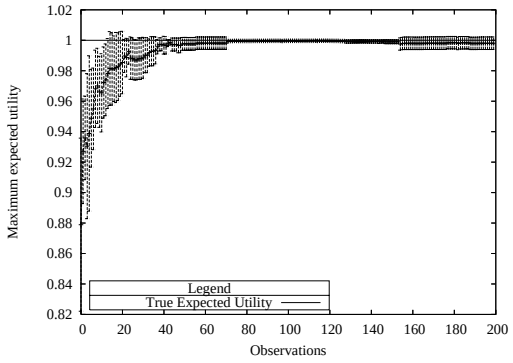


Figure D.3: Expected Utility for Imputing by Comparison in static domain

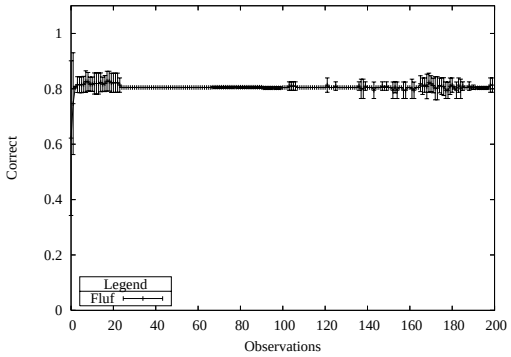


Figure D.4: *FLUF*'s chance of predicting decision A in a static domain

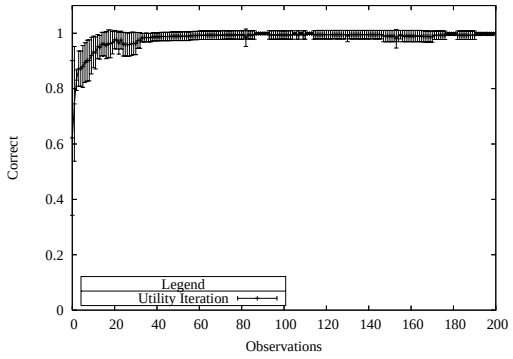


Figure D.5: Utility Iteration's chance of predicting decision A in a static domain

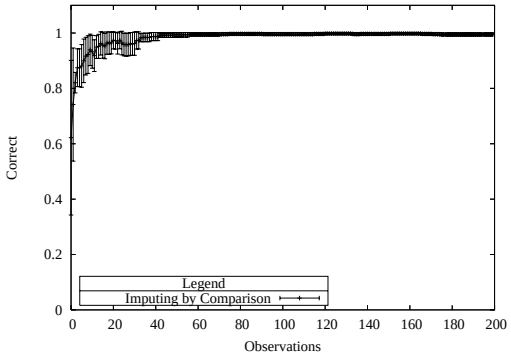


Figure D.6: Imputing by Comparison's chance of predicting decision A in a static domain

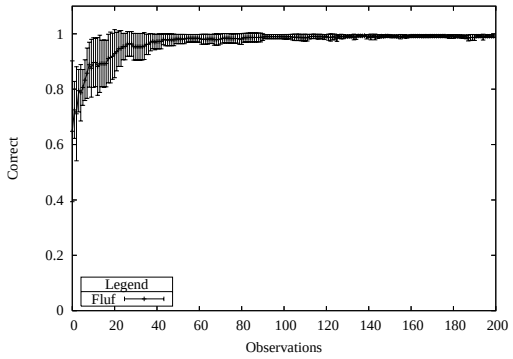


Figure D.7: *FLUF*'s chance of predicting decision T in a static domain

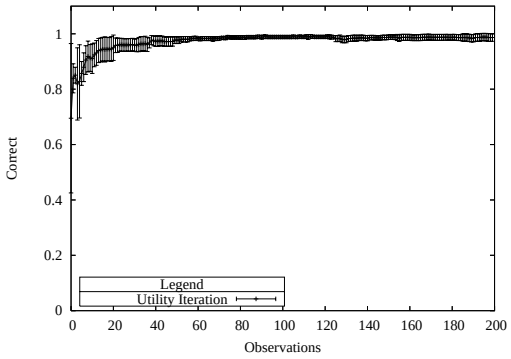


Figure D.8: Utility Iteration's chance of predicting decision T in a static domain

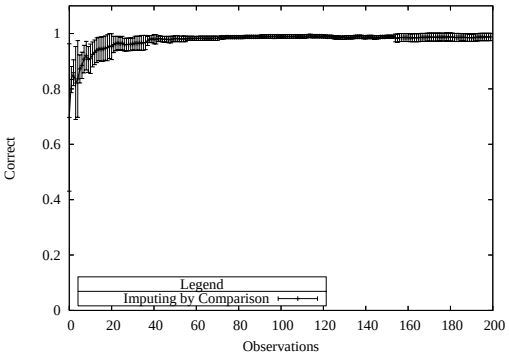


Figure D.9: Imputing by Comparison's chance of predicting decision T in a static domain

D.2 Domain with Drift

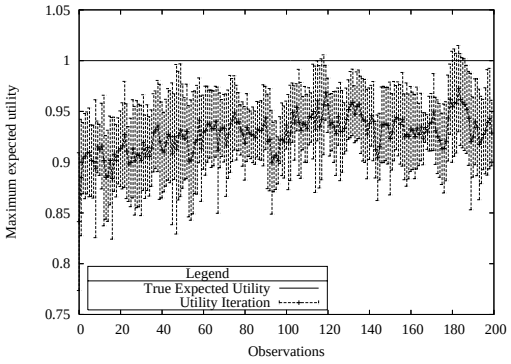


Figure D.10: Expected Utility for *FLUF* in one way drift

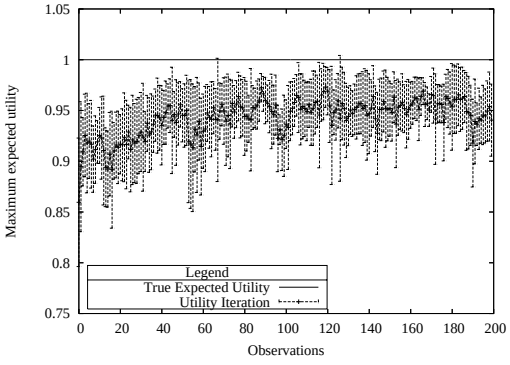


Figure D.11: Expected Utility for Utility Iteration in one way drift

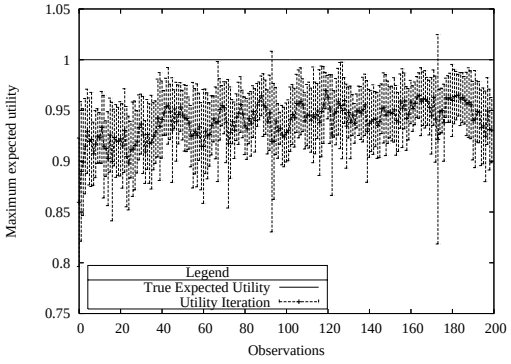


Figure D.12: Expected Utility for Imputing by Comparison in one way drift

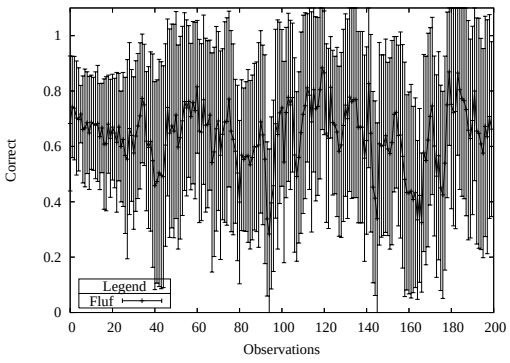


Figure D.13: *FLUF*'s chance of predicting decision A in a one way drift

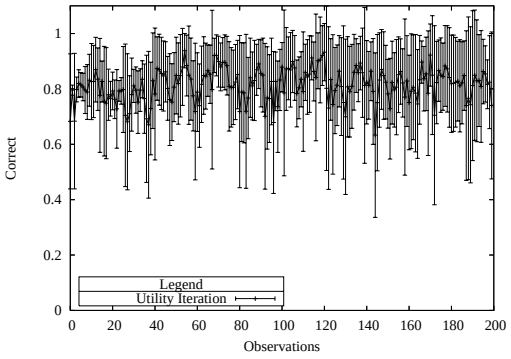


Figure D.14: Utility Iteration's chance of predicting decision A in a one way drift

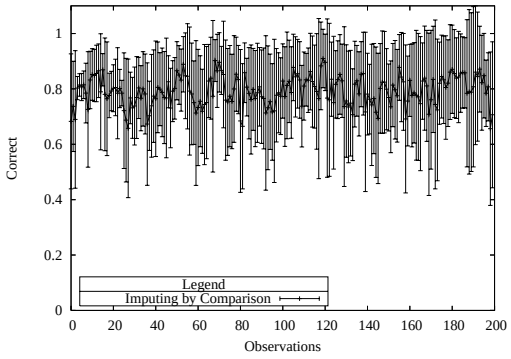


Figure D.15: Imputing by Comparison's chance of predicting decision A in a one way drift

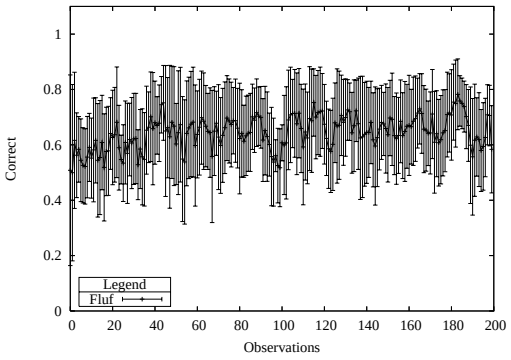


Figure D.16: *FLUF*'s chance of predicting decision T in a one way drift

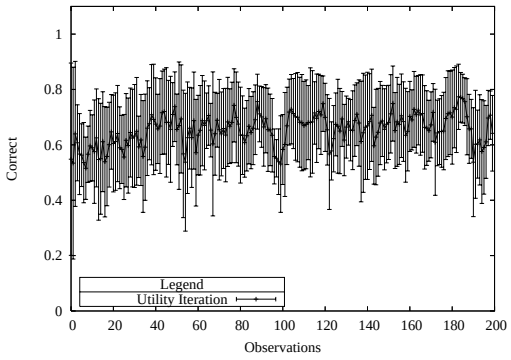


Figure D.17: Utility Iteration's chance of predicting decision T in a one way drift

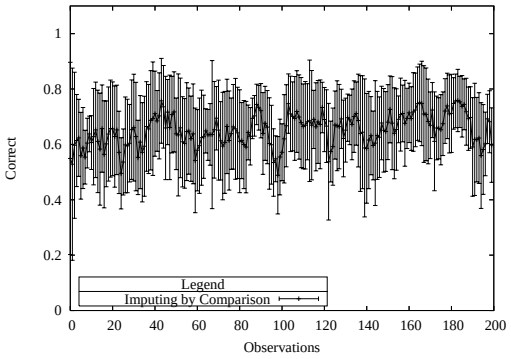


Figure D.18: Imputing by Comparison's chance of predicting decision T in a one way drift

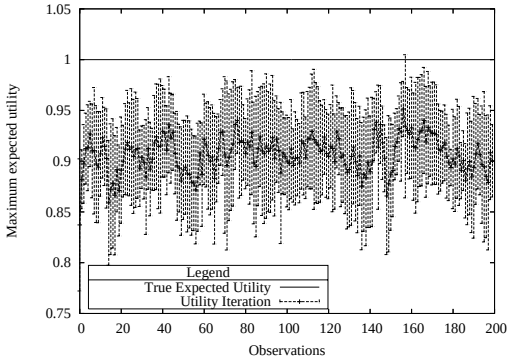


Figure D.19: Expected Utility for *FLUF* in local drift

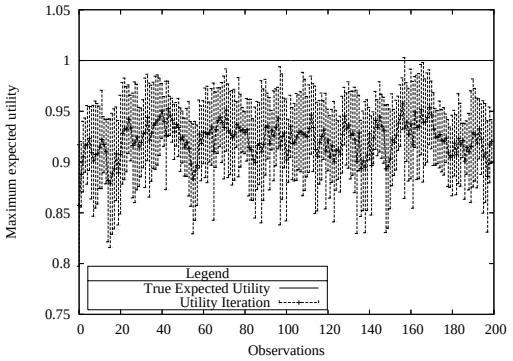


Figure D.20: Expected Utility for Utility Iteration in local drift

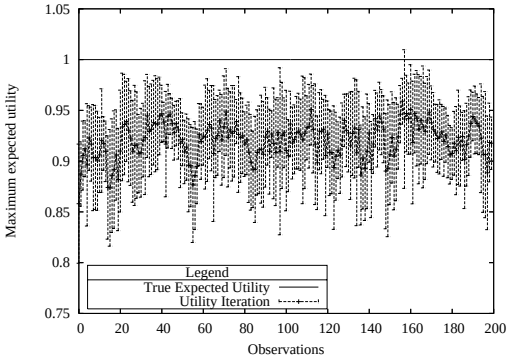


Figure D.21: Expected Utility for Imputing by Comparison in local drift

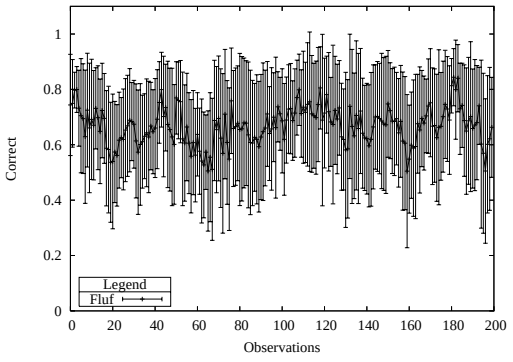


Figure D.22: *FLUF*'s chance of predicting decision A in a local drift

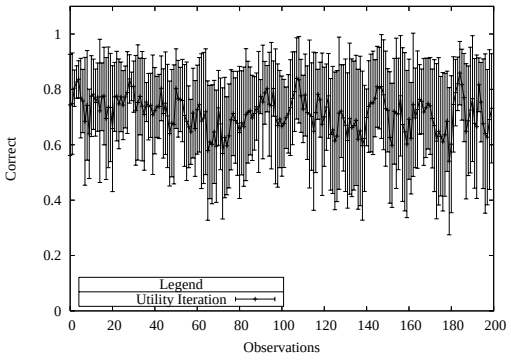


Figure D.23: Utility Iteration's chance of predicting decision A in a local drift

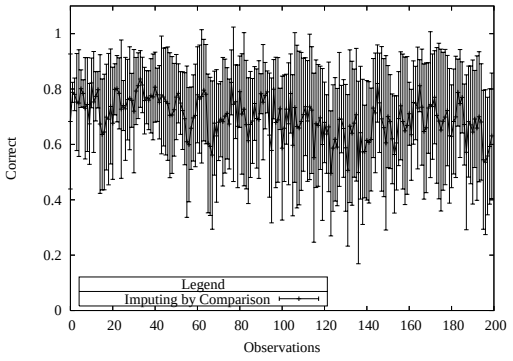


Figure D.24: Imputing by Comparison's chance of predicting decision A in a local drift

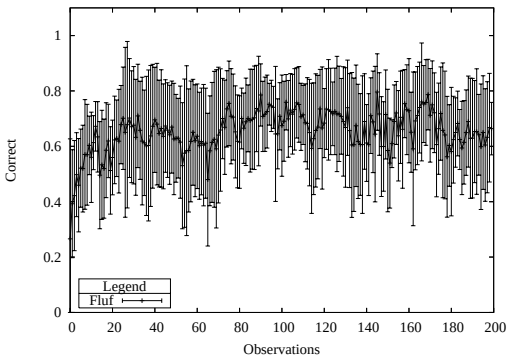


Figure D.25: *FLUF*'s chance of predicting decision T in a local drift

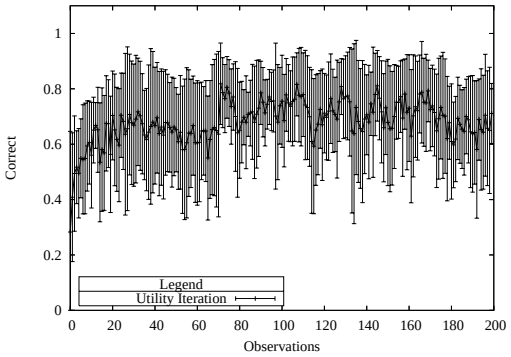


Figure D.26: Utility Iteration's chance of predicting decision T in a local drift

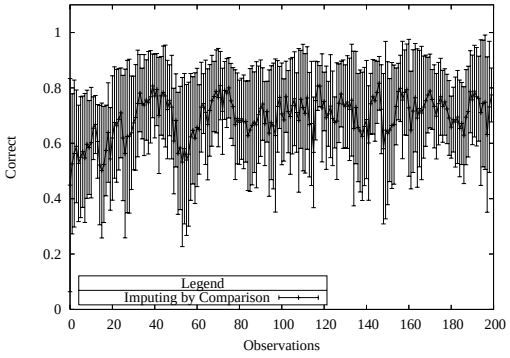


Figure D.27: Imputing by Comparison's chance of predicting decision T in a local drift

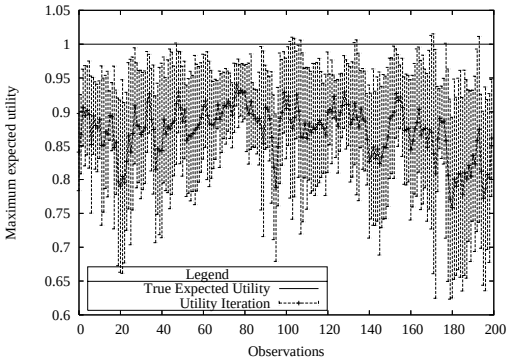


Figure D.28: Expected Utility for *FLUF* in random drift

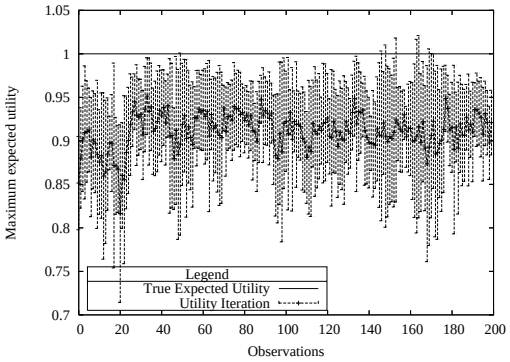


Figure D.29: Expected Utility for Utility Iteration in random drift

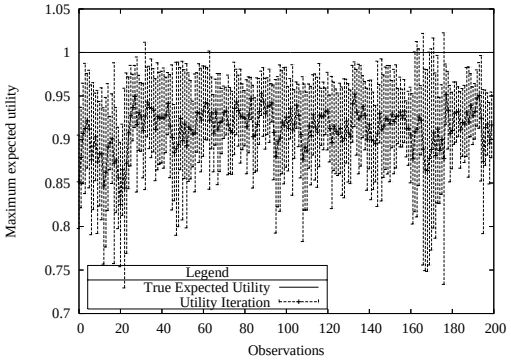


Figure D.30: Expected Utility for Imputing by Comparison in random drift

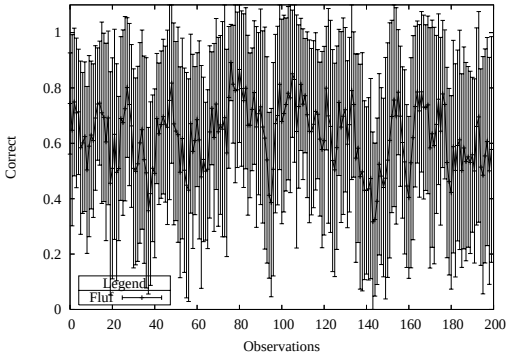


Figure D.31: *FLUF*'s chance of predicting decision A in a random drift

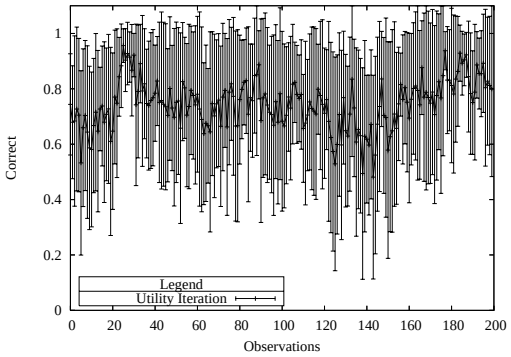


Figure D.32: Utility Iteration's chance of predicting decision A in a random drift

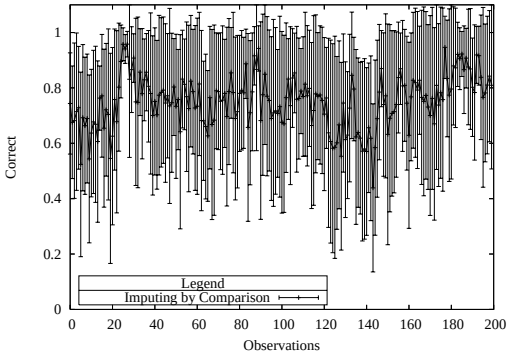


Figure D.33: Imputing by Comparison's chance of predicting decision A in a random drift

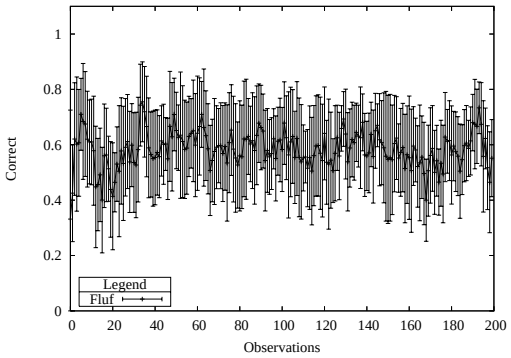


Figure D.34: *FLUF*'s chance of predicting decision T in a random drift

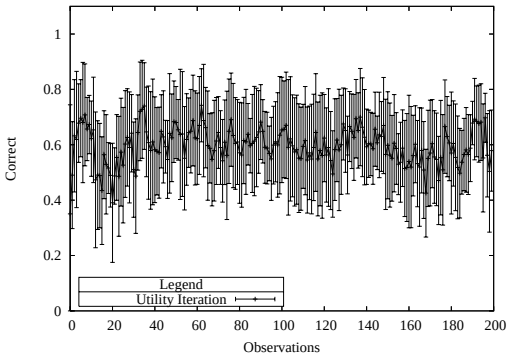


Figure D.35: Utility Iteration's chance of predicting decision T in a random drift

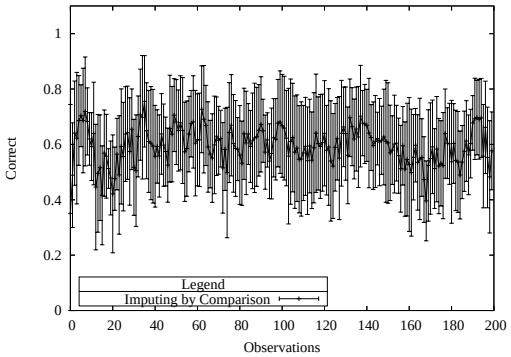


Figure D.36: Imputing by Comparison's chance of predicting decision T in a random drift

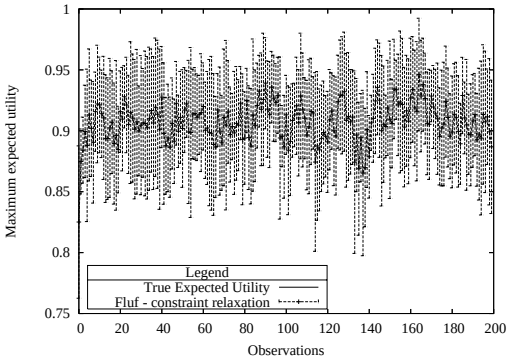


Figure D.37: Expected Utility for *FLUF* in local drift using constraint relaxation

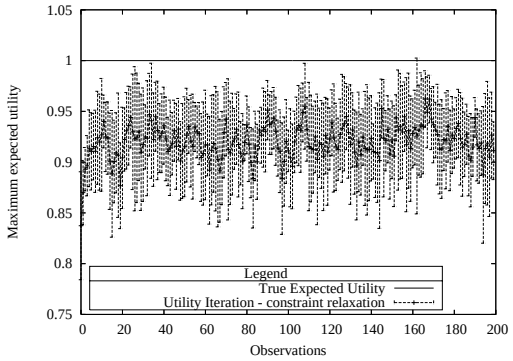


Figure D.38: Expected Utility for Utility Iteration in local drift using constraint relaxation

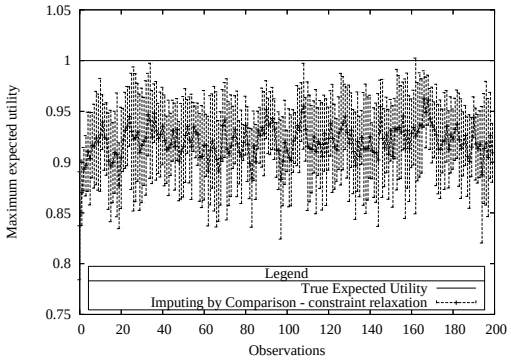


Figure D.39: Expected Utility for Imputing by Comparison in local drift using constraint relaxation

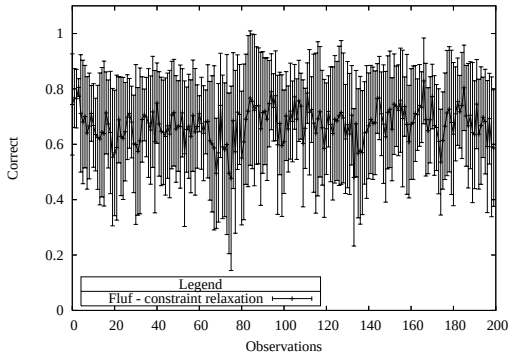


Figure D.40: *FLUF*'s chance of predicting decision A in a local drift using constraint relaxation

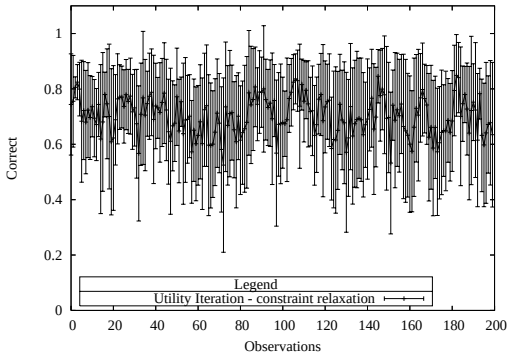


Figure D.41: Utility Iteration's chance of predicting decision A in a local drift using constraint relaxation

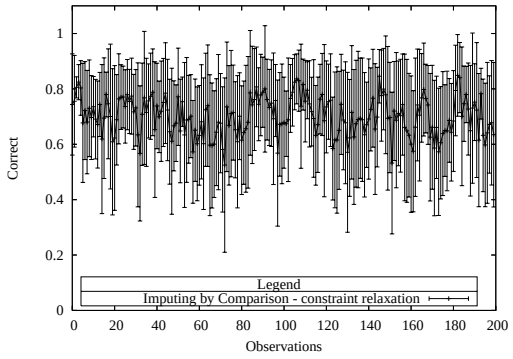


Figure D.42: Imputing by Comparison's chance of predicting decision A in a local drift using constraint relaxation

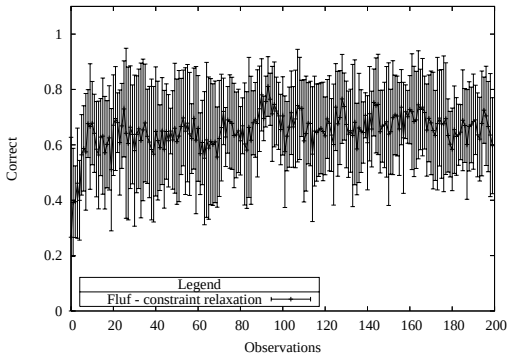


Figure D.43: *FLUF*'s chance of predicting decision T in a local drift using constraint relaxation

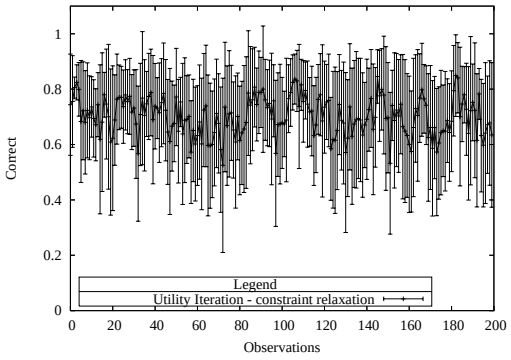


Figure D.44: Utility Iteration's chance of predicting decision T in a local drift using constraint relaxation

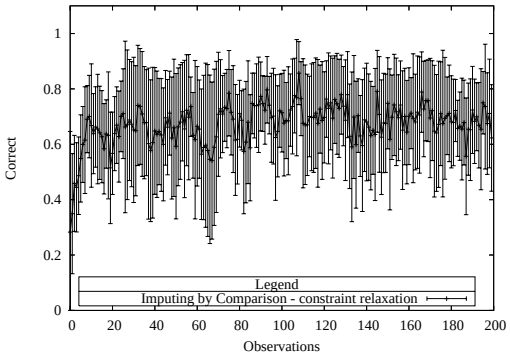


Figure D.45: Imputing by Comparison's chance of predicting decision T in a local drift using constraint relaxation

D.3 Domain with Fluctuation

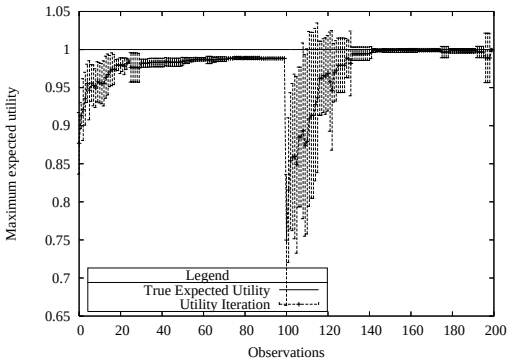


Figure D.46: Expected Utility for *FLUF* in single fluctuation

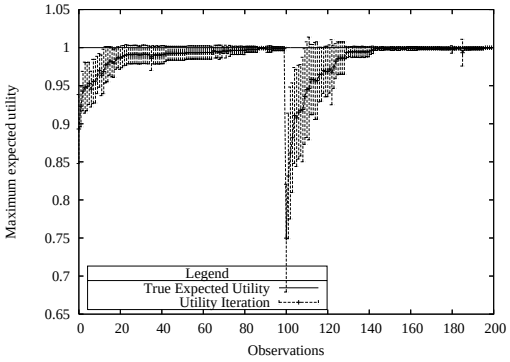


Figure D.47: Expected Utility for Utility Iteration in single fluctuation

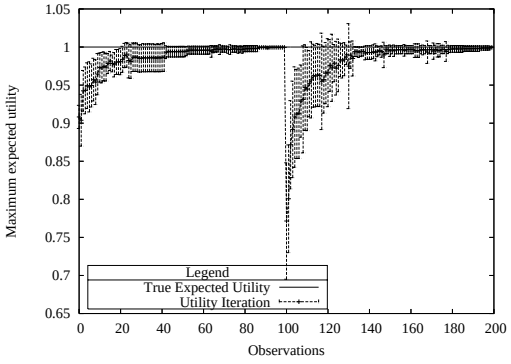


Figure D.48: Expected Utility for Imputing by Comparison in single fluctuation

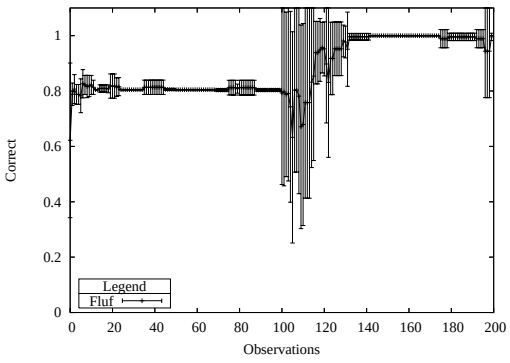


Figure D.49: *FLUF*'s chance of predicting decision A in single fluctuation

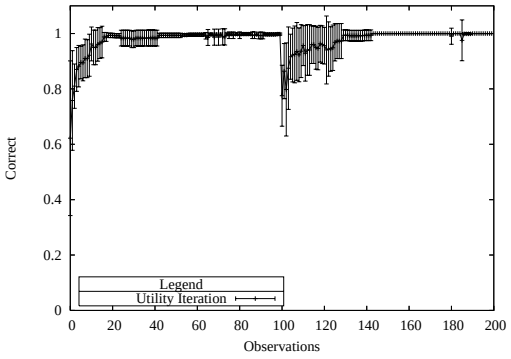


Figure D.50: Utility Iteration's chance of predicting decision A in single fluctuation

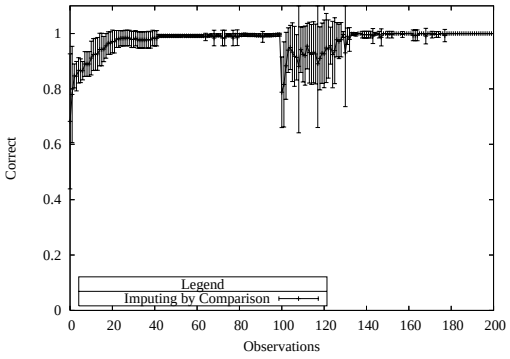


Figure D.51: Imputing by Comparison's chance of predicting decision A in single fluctuation

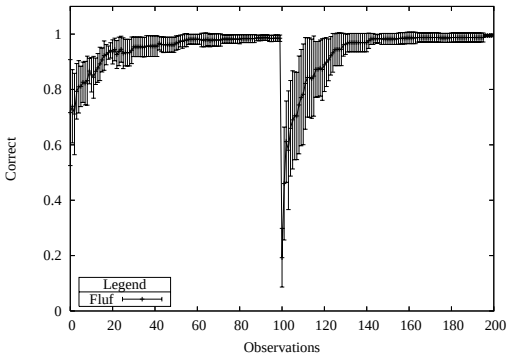


Figure D.52: *FLUF*'s chance of predicting decision T in single fluctuation

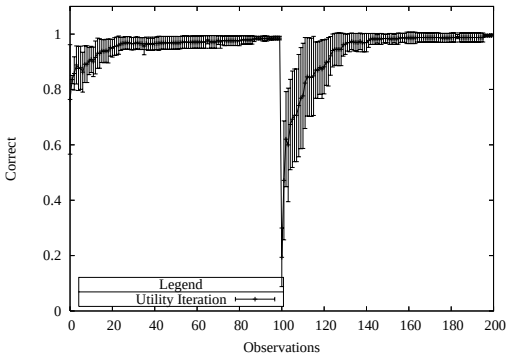


Figure D.53: Utility Iteration's chance of predicting decision T in single fluctuation

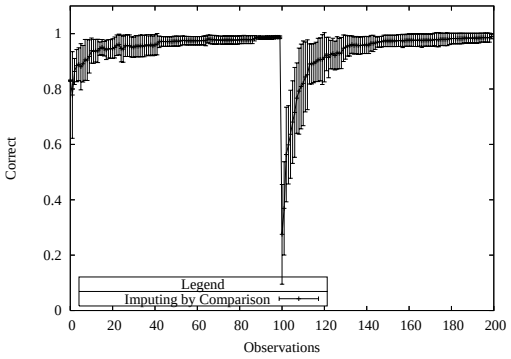


Figure D.54: Imputing by Comparison's chance of predicting decision T in single fluctuation

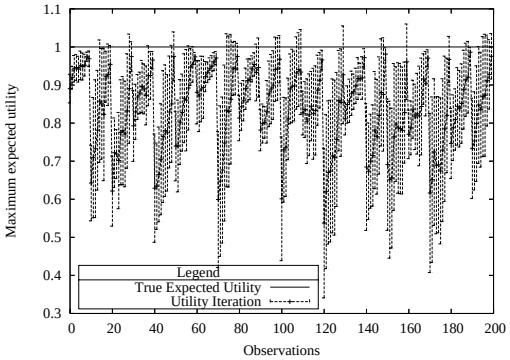


Figure D.55: Expected Utility for *FLUF* in multiple fluctuations

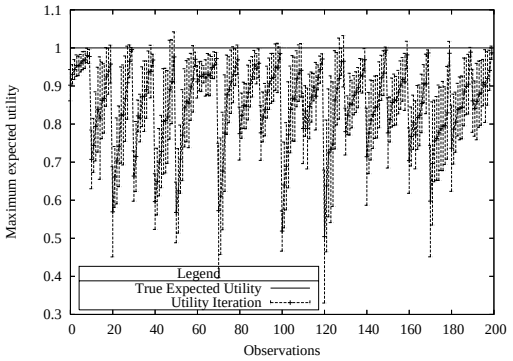


Figure D.56: Expected Utility for Utility Iteration in multiple fluctuations

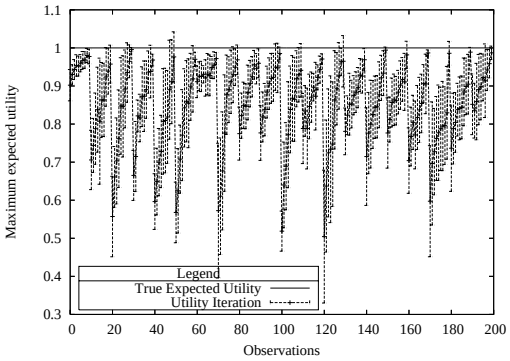


Figure D.57: Expected Utility for Imputing by Comparison in multiple fluctuations

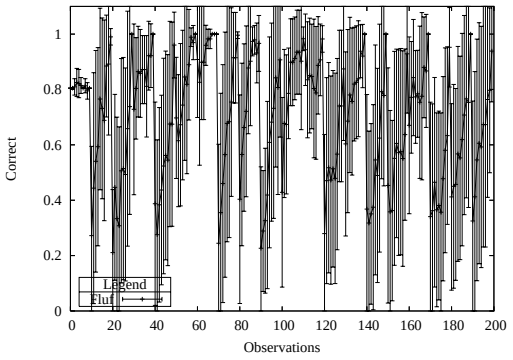


Figure D.58: *FLUF*'s chance of predicting decision A in multiple fluctuations

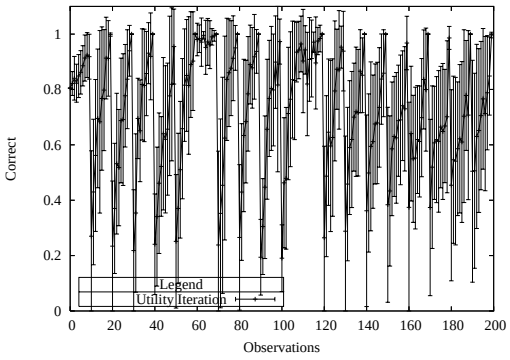


Figure D.59: Utility Iteration's chance of predicting decision A in multiple fluctuations

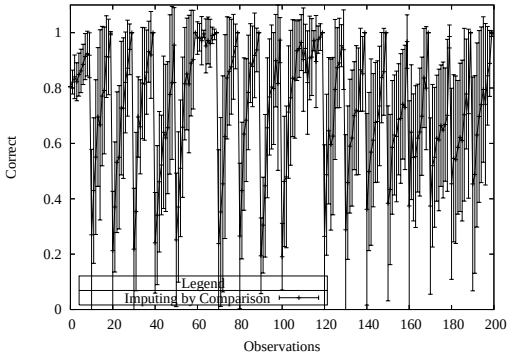


Figure D.60: Imputing by Comparison's chance of predicting decision A in multiple fluctuations

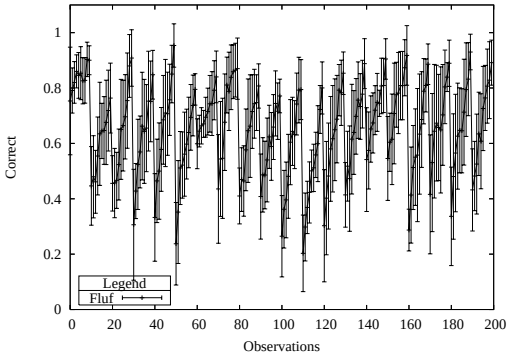


Figure D.61: *FLUF*'s chance of predicting decision T in multiple fluctuations

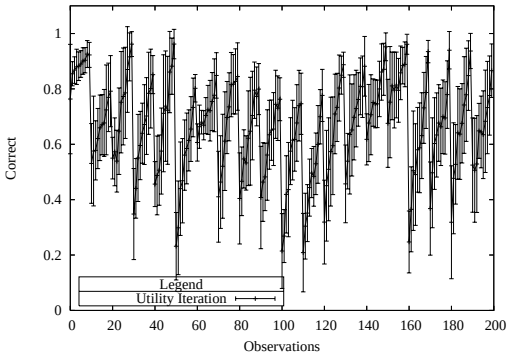


Figure D.62: Utility Iteration's chance of predicting decision T in multiple fluctuations

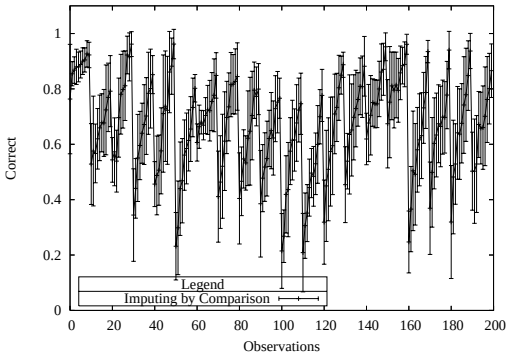


Figure D.63: Imputing by Comparison's chance of predicting decision T in multiple fluctuations

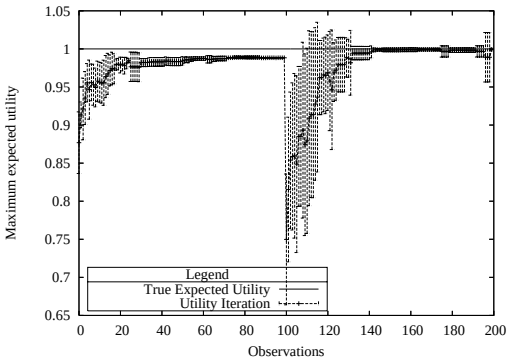


Figure D.64: Expected Utility for *FLUF* in single fluctuation using constraint relaxation

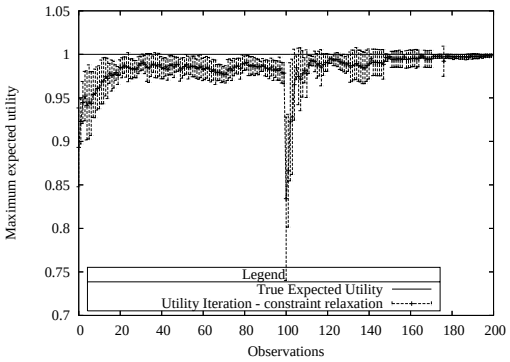


Figure D.65: Expected Utility for Utility Iteration in single fluctuation using constraint relaxation

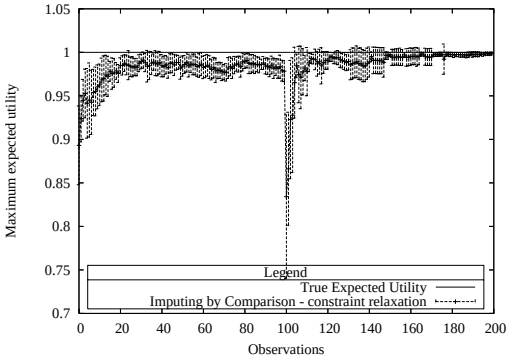


Figure D.66: Expected Utility for Imputing by Comparison in single fluctuation using constraint relaxation

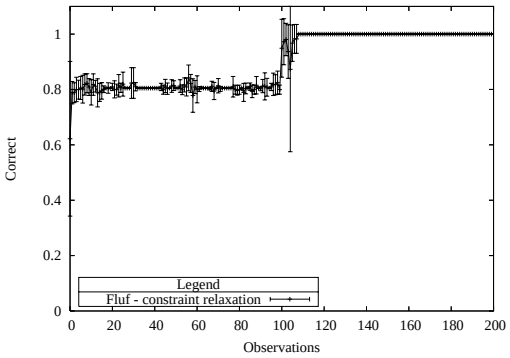


Figure D.67: *FLUF*'s chance of predicting decision A in single fluctuation using constraint relaxation

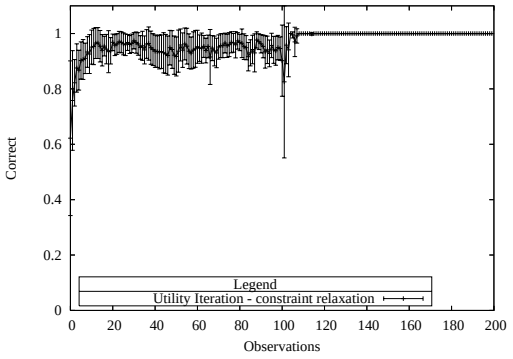


Figure D.68: Utility Iteration's chance of predicting decision A in single fluctuation using constraint relaxation

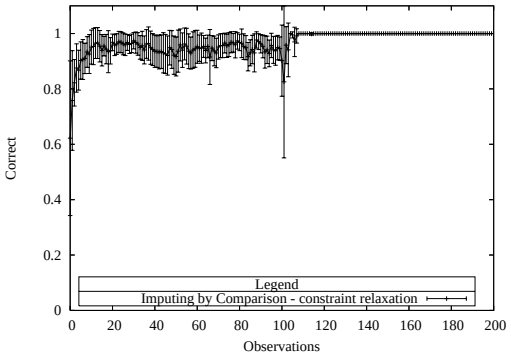


Figure D.69: Imputing by Comparison's chance of predicting decision A in single fluctuation using constraint relaxation

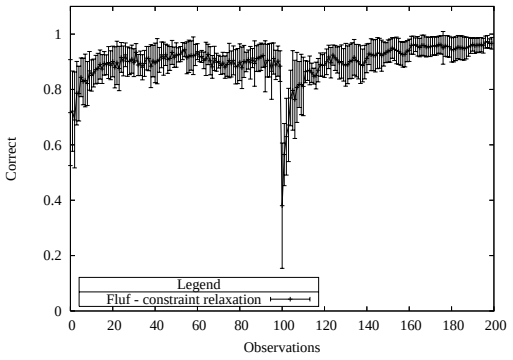


Figure D.70: *FLUF*'s chance of predicting decision T in single fluctuation using constraint relaxation

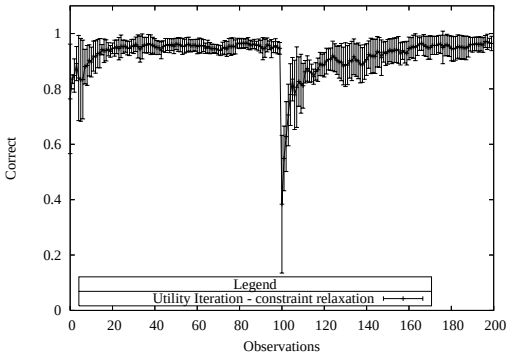


Figure D.71: Utility Iteration's chance of predicting decision T in single fluctuation using constraint relaxation

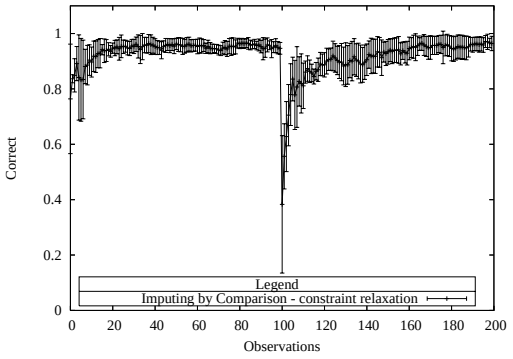


Figure D.72: Imputing by Comparison's chance of predicting decision T in single fluctuation using constraint relaxation

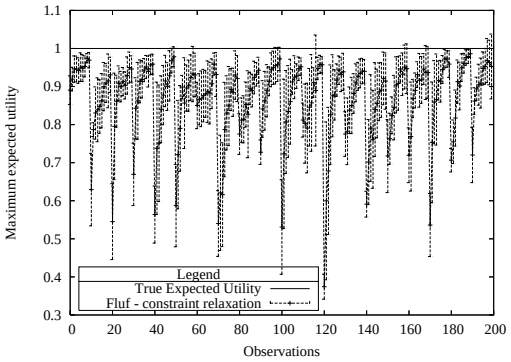


Figure D.73: Expected Utility for *FLUF* in multiple fluctuation using constraint relaxation

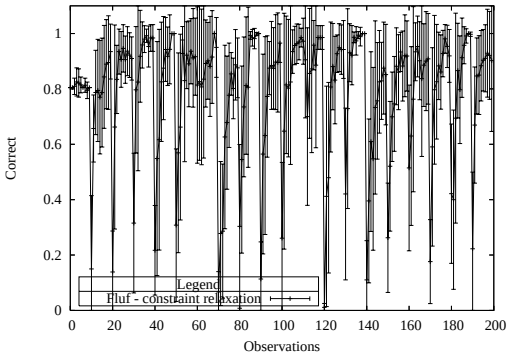


Figure D.74: *FLUF*'s chance of predicting decision A in multiple fluctuation using constraint relaxation

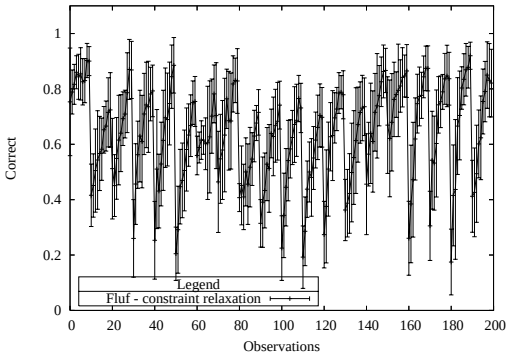


Figure D.75: *FLUF*'s chance of predicting decision T in multiple fluctuation using constraint relaxation

D.4 Domain with Noise

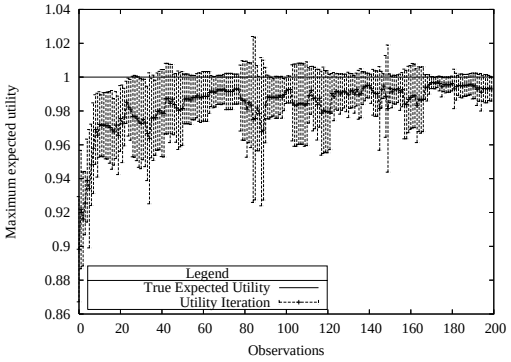


Figure D.76: Expected Utility for Utility Iteration for static domain, in noisy domain

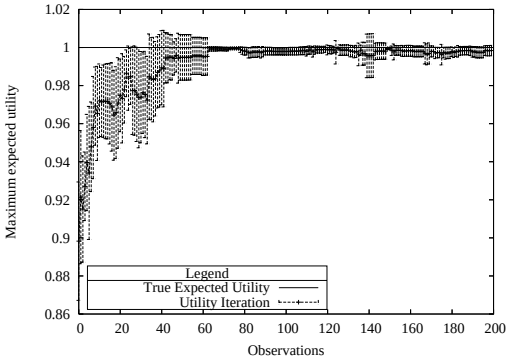


Figure D.77: Expected Utility for Utility Iteration in noisy domain

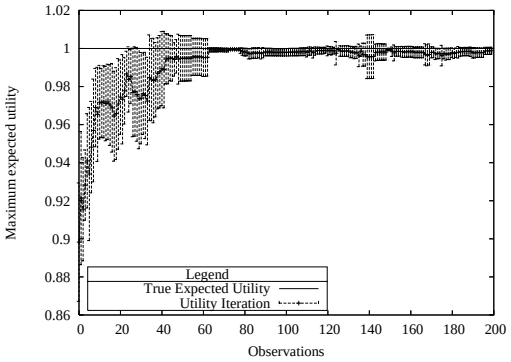


Figure D.78: Expected Utility for Imputing by Comparison in noisy domain

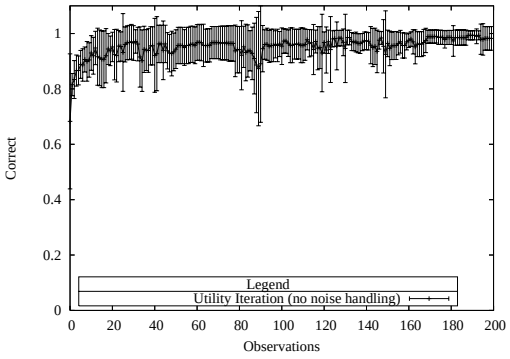


Figure D.79: Utility Iterations chance of predicting decision A in a noisy domain, using policy for static domain

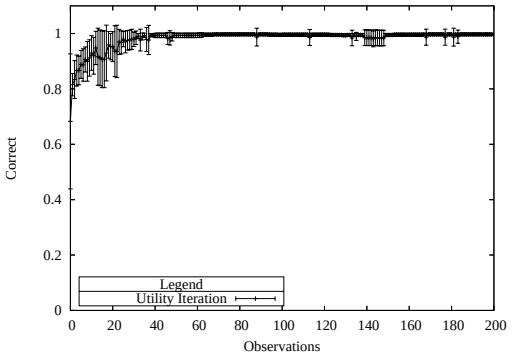


Figure D.80: Utility Iteration's chance of predicting decision A in a noisy domain

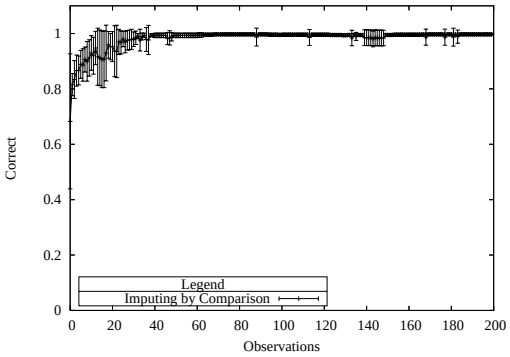


Figure D.81: Imputing by Comparison's chance of predicting decision A in a noisy domain

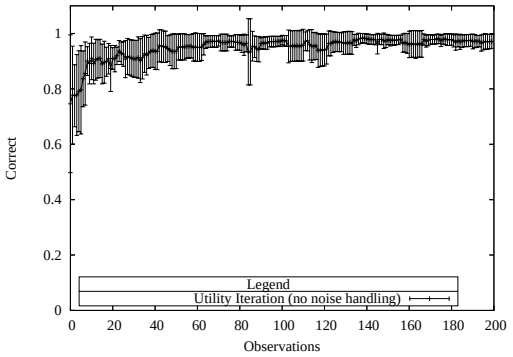


Figure D.82: Utility Iterations chance of predicting decision T in a noisy domain, using policy for static domain

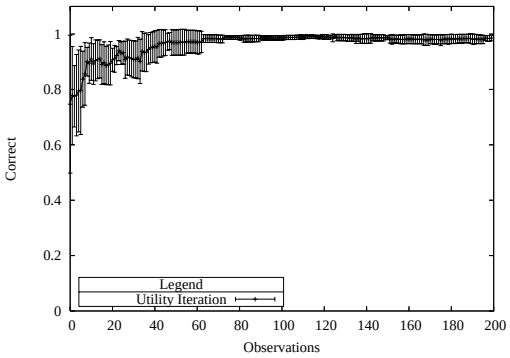


Figure D.83: Utility Iteration's chance of predicting decision T in a noisy domain

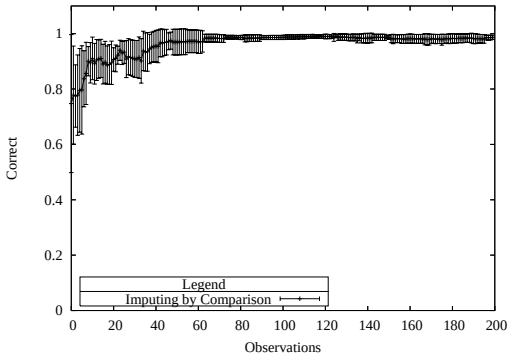


Figure D.84: Imputing by Comparison's chance of predicting decision T in a noisy domain

D.5 Alternative Domain

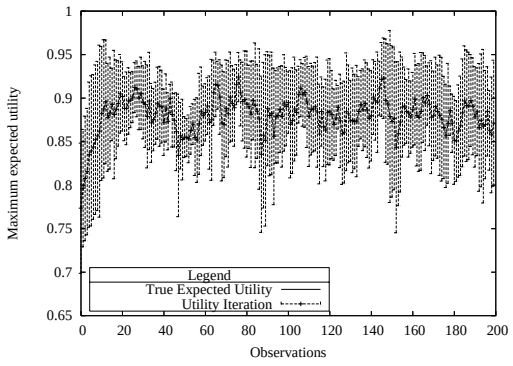


Figure D.85: Expected Utility for *FLUF* in the alternative domain with local drift

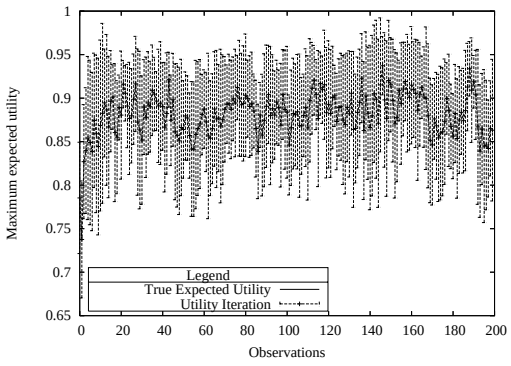


Figure D.86: Expected Utility for Utility Iteration in the alternative domain with local drift

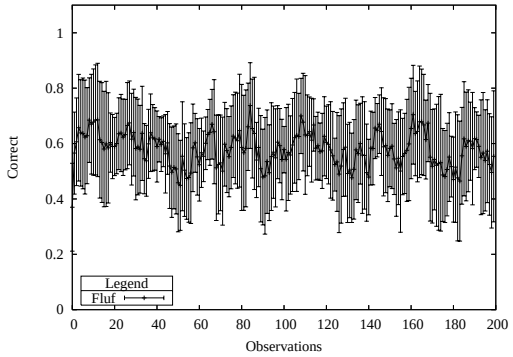


Figure D.87: *FLUF*'s chance of predicting decision A in the alternative domain with local drift

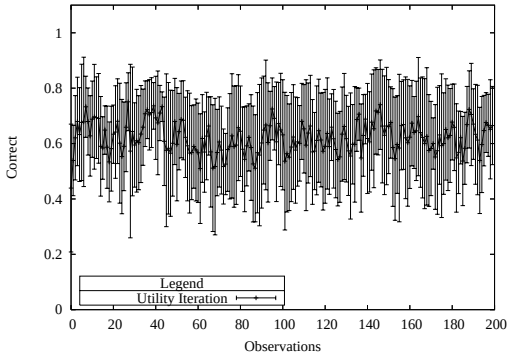


Figure D.88: Utility Iteration's chance of predicting decision A in the alternative domain with local drift

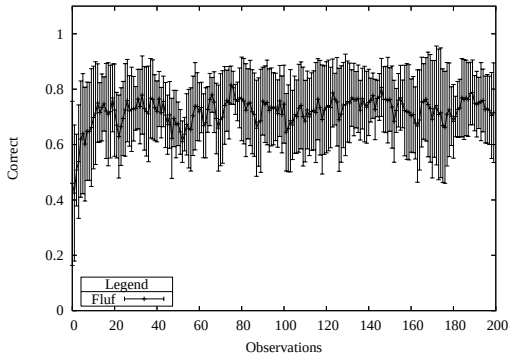


Figure D.89: *FLUF*'s chance of predicting decision T in the alternative domain with local drift

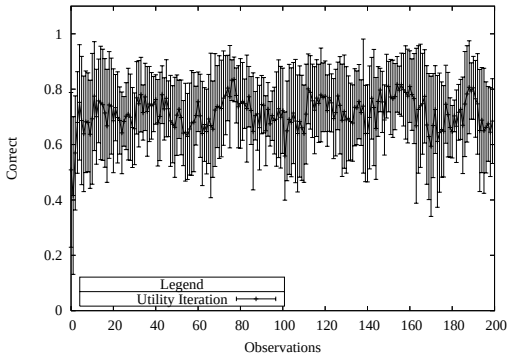


Figure D.90: Utility Iteration's chance of predicting decision T in the alternative domain with local drift

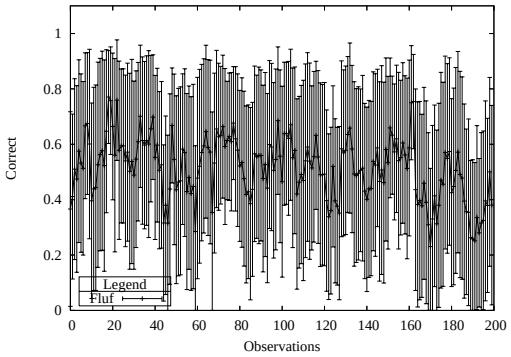


Figure D.91: *FLUF*'s chance of predicting decision D in the alternative domain with local drift

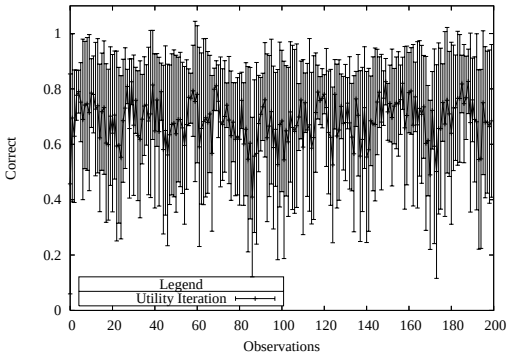


Figure D.92: Utility Iteration's chance of predicting decision D in the alternative domain with local drift

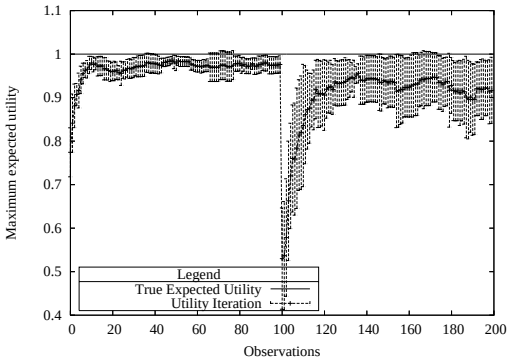


Figure D.93: Expected Utility for *FLUF* in the alternative domain with single fluctuation

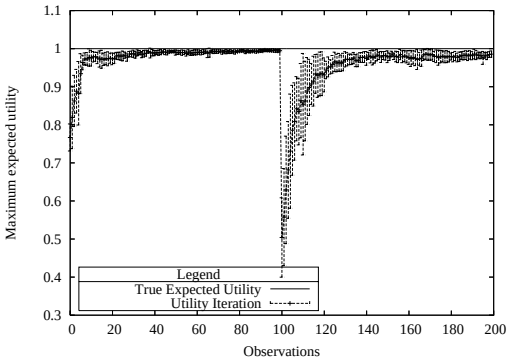


Figure D.94: Expected Utility for Utility Iteration in the alternative domain with single fluctuation

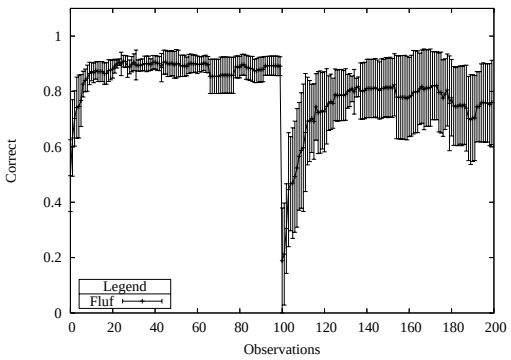


Figure D.95: *FLUF*'s chance of predicting decision A in the alternative domain with single fluctuation

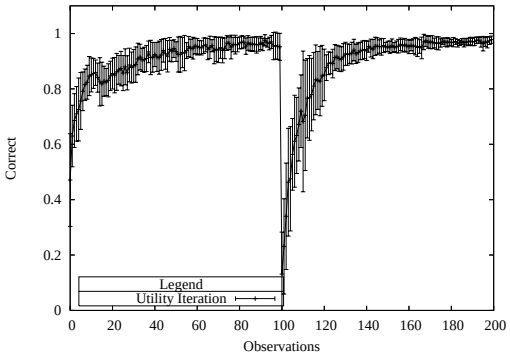


Figure D.96: Utility Iteration's chance of predicting decision A in the alternative domain with single fluctuation

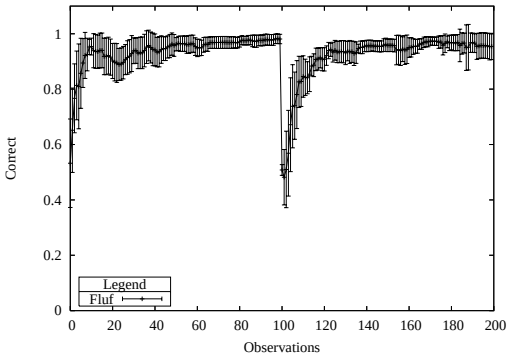


Figure D.97: *FLUF*'s chance of predicting decision T in the alternative domain with single fluctuation

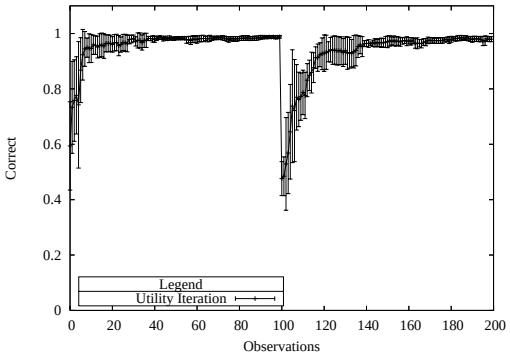


Figure D.98: Utility Iteration's chance of predicting decision T in a alternative domain two with single fluctuation

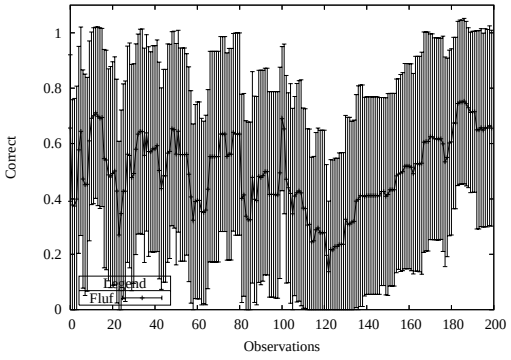


Figure D.99: *FLUF*'s chance of predicting decision D in the alternative domain with single fluctuation

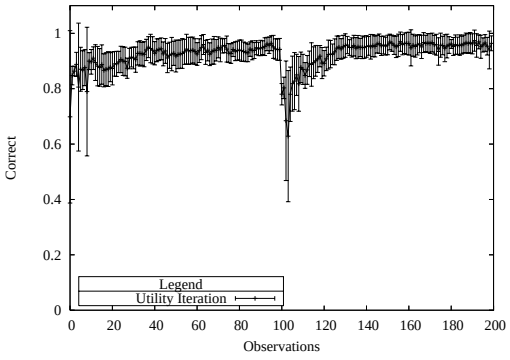


Figure D.100: Utility Iteration's chance of predicting decision D in a alternati3ve domain two with single fluctuation

D.6 Scalability

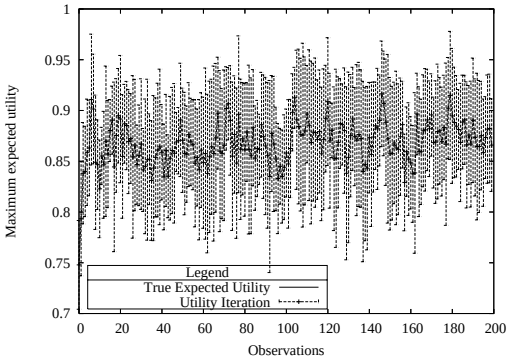


Figure D.101: Expected Utility for Utility Iteration in scalability domain

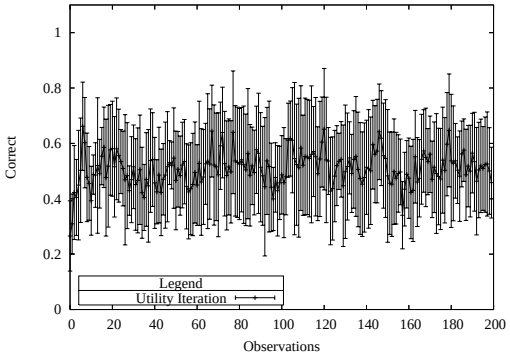


Figure D.102: Utility Iteration's chance of predicting decision D1 in scalability domain

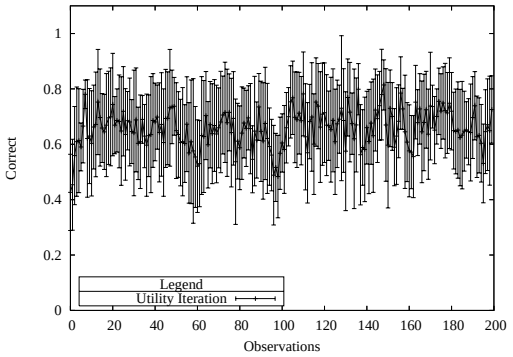


Figure D.103: Utility Iteration's chance of predicting decision D2 in scalability domain

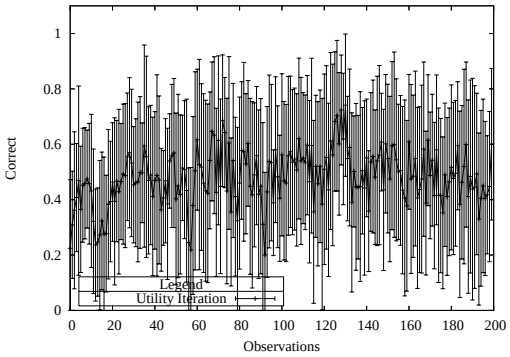


Figure D.104: Utility Iteration's chance of predicting decision LD in scalability domain

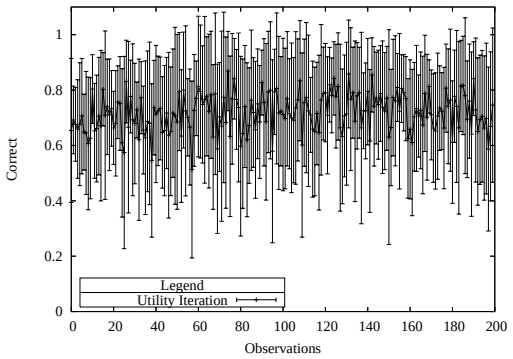


Figure D.105: Utility Iteration's chance of predicting decision RD in scalability domain

APPENDIX

E

Summery of Learning Utility Functions by Imputing

In this project, methods are developed to learn the utility function of an observed agent. The utility learning methods developed are called *Utility Iteration* and *Imputing by Comparison*. The methods are designed to handle agents in a static domains as well as agents that change behavior over time, much like humans do. The motivation for focusing on changing behavior, is that in any decision process of an agent that is to complex to model completely, any factors that are left out will still have an impact on the agent. The impact of these unmodeled factors can be interpreted as changing behavior.

Imagine attempting to model the behavior of a bus driver. Obvious factors such as traffic, the number of people riding the bus and the weather would probably be included in the model, but some factors that influence the drivers behavior may be difficult to include, such as how well he slept or the mood of his wife that morning. If, for example, the bus driver is an American football fan, and he stays up late to watch Monday night football, then this could influence his behavior on Tuesday. In other words there could be things that influence the driver that is not modeled, thus the impact of unknown factors can be interpreted as changing behavior on the drivers part.

E.1 Previous Work

The work done has been based on an earlier project, called "*FLUF* Learning Utility Function by Observing Behavior" (Hansen et al., 2004) which in turn is based on an article about estimation of utility functions, called "Learning an Agent's Utility Function by Observing Behavior" (Chajewska et al., 2001).

In the (Chajewska et al., 2001) article a method for determining the utilities in a decision tree is presented. Central to this method, *FLUF* and the methods developed in this project, is that a feasible space of utility values is maintained. This feasible space is m dimensional, where m is the number of utilities in the domain, and each point in this space assigns a value to each utility. Each point in the feasible space thereby corresponds to a utility function. Given a set of observations constraints are created, that limit the feasible space. These constraints are

in fact inequalities, such as $\alpha_1 u_1 + \alpha_2 u_2 > 0$, where u_1 and u_2 are utilities and the α values are determined by the method. After all observations have been used to create constraints, a *utility point* in the feasible space is chosen, that conforms with all the established constraints. In Chajewska et al. (2001) a distribution over possible utility functions is determined, based on a prior probability distribution over all utility functions.

FLUF is as mentioned, based on the method from Chajewska et al. (2001), and was developed to work on influence diagrams instead of decision trees, and it was developed to handle agents that, while being *rational*, changed their behavior over time. Semantically *FLUF* establishes constraints exactly as in the method from Chajewska et al. (2001), but the utility point is chosen differently. Due to agents being allowed to change behavior, the assumption of having a prior distribution over possible utility functions, was considered unlikely. Instead a method is used, that maximized a hypersphere inside the feasible space, while still conforming with all constraints. The center of this hypersphere was used as the utility point.

E.2 Assumptions

When developing *Utility Iteration* and *Imputing by Comparison* it was assumed that the probabilities and causalities in the domains, as the observed agent perceives them, are known. Furthermore, the agent is assumed to be rational, meaning it will always maximize its expected utility.

If the decision scenario being modeled includes a series of decisions, then each decision is assumed to have been observed in every observation, as well as the relevant past of these decisions.

E.3 The Developed Methods

In an attempt to develop new methods that achieved a higher accuracy than *FLUF*, an analysis on the inaccuracies on *FLUF* was done. The analysis revealed that the inaccuracies in *FLUF* was caused by relaxations of the created constraints done by *FLUF* to handle *partially observed strategies*. A partially observed strategy is when not all configurations of the domain has been observed. These relaxations were done to ensure that the utility values still allowed in the feasible space, included utility functions that allowed all possible decisions in the unobserved configurations of the domain.

So to remove this source of inaccuracy, the relaxations were replaced by imputations, in the sense that by imputing so called *virtual* observations for the configurations that were unobserved, the strategy became a *fully observed strategy*, meaning that relaxations would no longer be necessary. The imputed observations are called virtual, because they have never really occurred, and a strategy is called fully observed when all configurations of the domain are observed.

Both utility learning methods start by creating constraints for the last decision node in the temporal order. The reason for examining the last decision first, is that the later in the temporal order decisions are, the less imputations will be necessary, e.g. after the last decision there are no decisions that have not been observed. In Imputing by Comparison the order actually has no impact, but in Utility Iteration it does, as described below. After evaluating decision number n, both imputing methods impute observations to ensure that decision n is fully observed, i.e. ensuring that all configurations of the decisions relevant past has a corresponding decision. When decision n have been made fully observed, it can be replaced by a chance node

that encodes the policy of the decision node. This enables the imputing methods to create constraints based on observed choices in decision node $n-1$ without relaxations, since this has become the last in the temporal order, and so on. This means that the only difference between the two imputing methods is the way in which imputations are done.

E.3.1 Utility Iteration

In Utility Iteration virtual observations are chosen based on temporary utility functions. The initial temporary utility function is found by creating constraint for the last decision in all observations, because evaluating this decision does not require any imputations and therefore no temporary utility function. After adding constraints for decision number n in all observations, the center of the largest possible hypersphere, conforming with these constraints, is used as the initial temporary utility function.

Using this temporary utility function in the agents influence diagram, a policy, i.e. mapping between relevant past configurations and choices, can be obtained for any decision node. The initial temporary utility function is used in this way to obtain a policy for decision node n.

With the needed virtual observations imputed to make decision n fully observed, constraints can be added for the observed choices in decision $n-1$. However, constraints are only added for one of the observed decisions. This is because after adding the constraints from one observation of decision $n-1$, then a new temporary utility function can be found, and the newly added constraints together with the constraints created already will yield a more reliable utility function.

So after the initial temporary utility function has been determined, then choices are evaluated one at the time, in the order described above, each time refining the utility function. The temporary utility function will continually be refined, until all decisions have been evaluated in all observations. Now the final utility function, is the estimation done by Utility Iteration.

E.3.2 Imputing by Comparison

Imputing by Comparison finds the decisions in the unobserved configurations of the relevant pasts that should be imputed by comparing probability distributions. Some notation is needed to describe imputations in Imputing by Comparison. The hypothesis variables of a decision, is the parents of all utility descendants of that decision. These hypothesis variables can include chance as well as decision nodes. The utility descendants of some decision node, is the utility nodes that can be reached from that decision node by following a directed path through the influence diagram.

To impute a virtual observation for some relevant past of a decision node, Imputing by Comparison calculates the joint distribution over the hypothesis variable. This joint distribution is calculated for all the *true observations*, i.e. observations that have actually been made, by instantiation the past of the decision node as observed in each true observations, and treating decision nodes as deterministic chance nodes. By comparing the *Euclidean distance* between each true observation and each possible virtual observation, with respect to the joint distribution of the hypothesis variable, then the virtual observation with the smallest Euclidean distance is chosen.

E.4 Dynamic Domains

As the agent is allowed to change behavior over time, then observations can be made that conflict with each other, i.e. only the *trivial utility function* can allow both observations to occur. The trivial utility function is the utility function that attributes the same expected utility to all decisions, by having all local utility functions yield the same utility no matter the configuration of the influence diagram.

Since *conflicting observations* can occur, policies were developed that Utility Iteration and Imputing by Comparison could use to handle such conflicts. These policies were targeted on specific kinds of dynamic behavior, namely *drift*, *fluctuation* and *noise*. Drift is when the strategy of the agent gradually changes, fluctuation is when it suddenly changes and noise is when a single observations faulty, i.e. single variables or decisions have changed state from what they should have been in the observation.

E.5 Experimental Results and Conclusion

After conducting a series of experiments, it can be concluded that the utility learning methods based on imputing virtual observations, instead of relaxing constraints, will generally predict decisions more accurately. With regards to complexity, the experiments supported the complexity analysis, that indicated that Utility Iteration and Imputing by Comparison would have better scalability, when the domain grew, compared to *FLUF*.

INDEX

U

BIBLIOGRAPHY

Antonio Aguilera and Richardo Peréz-Aguila. General n-Dimensional Rotations. In *WSCG SHORT Communication papers proceedings*. UNION Agency - Science Press, 2004.

Hugin Expert A/S. Hugin Researcher,
@ `http://www.hugin.com/Products_Services/Products/Academic/Researcher/` 2004.

Michel Berkelaar *et al.* LP Solve,
@ `http://groups.yahoo.com/group/lp_solve/` 2005.

Urzula Chajewska, Daphne Koller, and Dirk Ormoneit. Learning an Agent's Utility Function by Observing Behavior. In *Proceedings of the 18th International Conference on Machine Learning (ICML '01)*, pages 35–42, 2001.

Fraleigh and Beauregard. *Linear Algebra*. Prentice Hall, second edition, 2003.

Anders Hansen, Nicolaj Lock, and Peter Poulsen. *FLUF Learning Utility Function*. Master's thesis, Aalborg University, 2004.

Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

R. Shachter. Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams,
@ `citeseer.ist.psu.edu/shachter98bayesball.html` 1998.

R. Shachter. Efficient Value of Information Computation. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 594–601, San Francisco, CA, 1999. Morgan Kaufmann Publishers.