A Genetic Irrational Belief System

by

Coen Stevens

The thesis is submitted in partial fulfilment of the requirements
for the degree of Master of Science in Computer Science

Knowledge Based Systems Group
Faculty of Electical Engineering, Mathematics and Computer Science
University of Delft

In association with the Faculty of Technology
Department of Computer Science and Software Engineering
University of Portsmouth

Graduation Committee:
Dr. Drs. L.J.M. Rothkrantz
Prof. T.R. Addis
ir. F. Ververs
Dr. ir. C.A.P.G van der Mast

October 11, 2005

Abstract

The challenge set by T. R. Addis et al. is if we can construct computing based upon family resemblance rather than sets, paradigms rather than concepts, and metaphor rather than deduction? Can we devise systems that have judgement rather than decisions? To meet the challenge we need to change our approach towards computer science and start thinking with the notion of so-called Irrational sets, knowing that not everything is potentially unambiguously describable.

In this thesis we will present a model for an adaptive intelligent system that is able to handle the irrational sets and with it takes on the challenge.

The model will show how we can have a set of hypotheses to which we assign certain beliefs to create a belief profile and with it predict the state of the world. For the belief profile we can intelligently create new hypotheses to predict even better or to compensate for the irrational behavior of the world. The world changes under time and context, so the system needs to keep track of it by modeling the irrational sets and provide a mechanism for handling them.

The model is implemented to create a system for cocktail evaluation. By modeling the irrational taste of the user, the system can predict the user's taste for any cocktail combination. Hypotheses will represent possible evaluation functions that represent the user's personal evaluation. By generating new hypotheses and adjusting the belief in the old and new ones, the system keeps on searching for the best taste approximation and is able to keep track of shifts in the taste of the user. The system adapts to the feedback of the user, which are cocktail ratings.

Tests will show that the system indeed searches for the best approximation and adapts to changes made by the user.

Acknowledgements

I would like to thank my supervisors who supported me with my thesis work at the University of Portsmouth, Prof. T.R. Addis and my second informal supervisor Dr. B. Visscher for their much needed guidance, stories and support. At the TUDelft I would like to thank my supervisor Dr. Drs. L.J.M. Rothkrantz for his support.

Besides my supervisors I also like to thank my dear colleagues, Mr. Mouhammd Al-kasassbeh, Miss Antoniya Georgieva, Mr. Vikas Grover, Miss Xiaoxia Wang, Mr. Tanos Paraskelidis, Miss Hoeda and Mr Phillipe B., who have become good friends and made my stay at the University of Portsmouth a very pleasant one.

Six months away from home is one thing, but leaving your girlfriend at home is a bit harder. So I really like to thank my girlfriend, Femke van Langen, for supporting me and all the time she took for calling and visiting me.

Finally I could not have stayed 6 months in Portsmouth without the financial support of:

- My parents, Cor and Toos Stevens
- Fundatie van de Vrijvrouwe van Renswoude
- Stimuleringsfonds voor Internationale Universitaire Samenwerkingsrelaties (STIR)
- Faculteitsfonds
- Koninklijk Instituut Van Ingenieurs (KIVI)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

At this moment we see Artificial Intelligent systems around us which can help us decide on practical issues (Management, planning, stating medical diagnoses, etc). Most of the current AI research focuses on these practical engineering tasks. Designing systems that perform intelligent human tasks, where the system doesn't have to be intelligent, but only needs to show intelligent behavior. This approach maybe called 'Weak AI', a concept introduced by John R. Searle (see [12]) which states that:

> *"According to weak AI, the principal value of the computer in the study of the mind is that it gives us a very powerful tool."*

Software is a powerful tool to explore the notion of intelligence, but that doesn't mean that it has to be intelligent.

With weak AI numerous human tasks requiring intelligence can and have been automated, but there are tasks that make automation quite difficult, complex and seemingly impossible. While an expert system for Car Engine Fault detection can work very good (see [8]), a system that composes music is rather difficult. Problems like fault detection are often well defined and characterize themselves to be deterministic. Implementing a set of well defined rules is not too difficult.

Non-deterministic problems on the other hand are more difficult. Especially when we need the system to perform a creative act as in composing music. Creative solutions ask for a creative system actually being intelligent. That creativity is closely linked to intelligence is shown by T. R. Addis (see [1]) as he defines intelligence as the component of thinking that involves insight and reason. Insight entails the creative act, to find out which concepts are at work, and the insights are used through the process of reason

to come up with an answer.

The claim for a system actually being intelligent moves us from 'Weak' towards 'Strong' AI.

> *"according to strong AI, the computer is not merely a tool in the study of the mind; rather, the appropriately programmed computer really is a mind"*

Aiming for a system with human-like intelligence has been the goal of the first AI researchers(see [9]). For decades researchers have tried to come up with such a system, but never succeeded.

What is the problem here? Why can't AI researchers create a system being intelligent that can truly reason? Why is it that every time when we seem to be making progress we hit a wall again? According to T. R. Addis et al. (see [4]) the problem lies in the current paradigm of Computer Science. More computational power is not the solution, we need to change our view about the nature of the world. We need to make a paradigm leap, switching from the Rational view towards the Irrational view, assuming that not everything is potentially unambiguously describable. What we mean by the Irrational view will be explained in Chapter 2.

I will continue to explain the need for the paradigm leap and how to make that leap. For this I use the paper by T. R. Addis et al. (see [4]) which is summarized in the next paragraph. The shift in paradigms opposes a challenge which I will take on.

This thesis is meant to be a step towards the ultimate goal of strong AI, creating a system which truly is intelligent.

1.1 A Paradigm leap

The current state of Computer science as a paradigm could be described by taking Wittgenstein's Tractatus. Where the Tractatus represented a formal and logical representational schema into a descriptive form, based upon referential (denotational) semantics. It follows from the Tractatus propositions that everything is potentially unambiguously describable and all sets are rational. Where set membership is always specifiable and context independent or has an explicit context.

> Definition of a 'Rational' set: *A set where there is a finite set of rules that can include unambiguously any member of that set and unambiguously excludes any non-member of that set.*

The Tractatus gives us an extensive model of computer languages, where names in propositions do not always refer to primitive objects but are themselves referencing propositions. These propositions are complexes that finally end up as compound statements whose ultimate referent is the bit, where the bit is the mechanical equivalent of Wittgensteins referent objects.

Computer programs are based on formal descriptions and are in fact program structures of bits, where it is at the bit that the program links to the world and has meaning.

The consequence of such a formal model is that any set of names can be used in a program to represent a proposition. All that is necessary is that there is a formal definition that gives the name meaning within the program in terms of the proposition it represents.

In practice this means there is an infinite but bounded set of possible organizations of a program, where a program can only have one interpretation.

The social consequences are that rules can be constructed that can describe unambiguously any situation, so rules can bypass human judgement. In other words, there is only one correct way to see the world. Having stated that programs can have only one interpretation raises the problem of dual semantics (See figure 1.1), because programs have at least two interpretations, namely the Computer State and the Problem Domain.

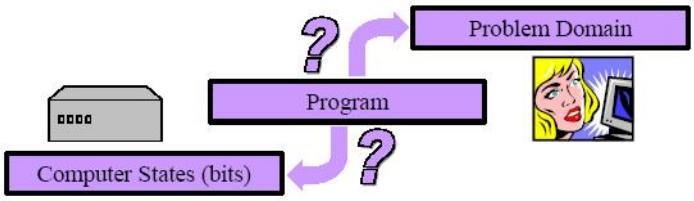

Figure 1.1: Problem of Dual semantics

Wittgenstein himself noted the flaws in his early work, that everything is not potentially unambiguously describable. An example is an attempt to define a chair, see figure 1.2). It is impossible to find a definition that will either exclude all examples that are not chairs or include all examples that are. There are also irrational sets and some sets depend upon human usage and context.

Definition of an 'Irrational' set: *A set where no finite set of rules can be constructed that can include unambiguously any member of that set and, at the same time, unambiguously exclude any non-member of that set.*

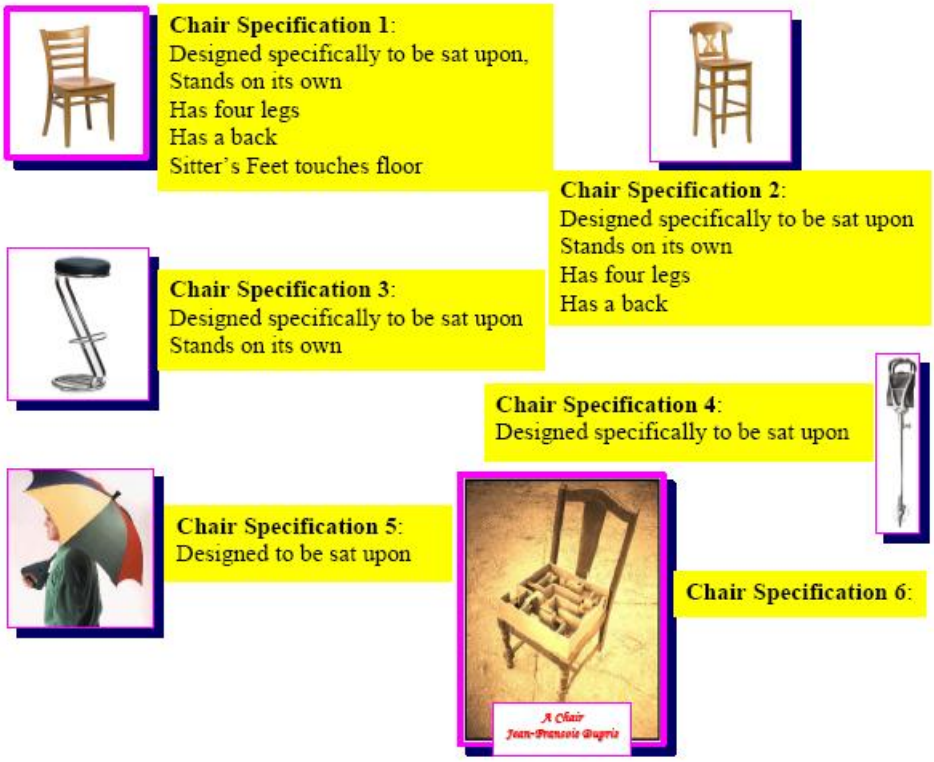

Figure 1.2: The problem of defining a chair

So Inferential semantics was brought to life to account for the exceptions. What's important to notice here is that having ambiguous rules means that there will always be a need for human judgement. The semantics of irrational sets for Wittgenstein lies in the use of family resemblance instead of sets and word usage (and structures) instead of reference. An alternative to Wittgenstein is Lakoff's use of prototypes (paradigms) and metaphor instead of reference. The challenge set by T. R. Addis et al.(see [4]) is:

> *"can we construct computing based upon family resemblance rather than sets, paradigms rather than concepts, and metaphor rather than deduction? Can we devise systems that have judgement rather than decisions?"*

They do not yet give us a solution to the problem, but clearly state the nature of the problem. They didn't invent something completely new as it was always there, but now it is defined. Having the problem defined we can change our approach towards computer science and start thinking with the introduced notion of Irrational sets.

1.2 The Challenge

I'm going to take on the challenge to construct an adaptive intelligent system knowing that not everything is potentially unambiguously describable. To meet the challenge requires a system that:

1. handles Irrational sets. Using family resemblance, paradigms and metaphors instead of referential semantics. We are going to use irrational sets to replace referential semantics.

2. makes sensible judgements. Instead of deductive and deterministic reasoning it needs a creative intelligence to be applied with an emphasis on abduction (See 2.5).

To create such a system we firstly need to find a way how to handle irrational sets as computers can only handle directly rational sets.

With irrational sets we can have ambiguous rules and this means that there will always be a need for human like judgement. What is acceptable behavior or performance is a time sensitive and socially dependent notion and therefore could change over time and context. If we want to track these changes we need to handle user feedback. The system needs to provide an interface to the user to set up a conversation. The system should be able to adapt in response to the user feedback.

Secondly in order to make sensible judgements, some form of artificial intelligence is needed, suitable in handling the irrational nature of the problem domain.

So what we want is to create an adaptive intelligent irrational system within a problem domain that requires the use of irrational sets.

1.3 Problem domain

We had to find a suitable problem domain that would ask for the kind of system supposed by the challenge. The problem domain we considered had to do with the invention of new cocktail mixes. We had in mind a system that could evaluate cocktails and make suggestions to the user what to drink. These suggestions would also include new invented unknown cocktails and should fit the user's taste. We want the system to know what the user likes and based on that act creatively recognized by inventing new cocktails.

In the creative process of invention we can distinguish two phases:

1. Idea or concept generation.

2. Evaluation.

Firstly there is the part of coming up with new concepts or ideas. For inventing cocktails it means coming up with a new combination of cocktail ingredients.

Secondly we need to evaluate the concepts or ideas. We need to see whether they are any good or not. For this there needs to be a measure of success. In case for cocktails we can measure how well they taste.

The system's key feature is to evaluate cocktails in a way that represents the user's taste to predict taste for unknown cocktails.

The reason why we considered this problem domain is that evaluating and inventing cocktails requires the kind of system supposed by the challenge, which becomes clear when we look at the system requirements for the challenge:

1. Handle irrational sets: The system needs to handle taste, where taste itself is a good example of an irrational set. It seems difficult to define a person's taste rational. And even if we can define taste, it seems to change over time and context, making the choice for an irrational set inevitable.

2. Make sensible judgements: In order to select cocktails favorable to the user, the system needs to asses the user's taste. Having acquired the user's taste, the system should be able to predict the taste for unknown cocktails. Hence new cocktail inventions can be evaluated. With sensible judgements we mean in fact an acceptable (to the user) prediction mechanism for cocktail evaluations.

1.4 Objectives and assignment

The main objective of this thesis it to take on the challenge by constructing an adaptive intelligent irrational system, which is able to assess the user's cocktail taste and predict the taste for cocktails. Therefore the system needs to:

1. Handle taste as an irrational set.
2. Make sensible judgements (evaluations).

We will focuss on the second phase of the creative process of invention: 'Evaluation'. We want a system that can make sensible judgements or in other words acceptable predictions for cocktail evaluations. The reason why is that at first we invested a lot of work on the generation of new cocktail combinations (see appendix A), but it appeared later to us that no matter what kind of sophisticated cocktail creation mechanism you design, for a creative system you will always need to have a mechanism to evaluate the creations. And as the user's taste changes under time and context, we need an evaluation system capable in tracking changes for which we need to handle irrational sets.

Therefor we 'skip' the idea or concept generation and first design and implement a system that can predict the taste for a bounded set of cocktails. As soon as we have the appropriate mechanism for the evaluation of ideas (cocktails), we may take the next step and introduce a mechanism for the generation of ideas, new cocktail combinations. A mechanism for generating new cocktails could then be as simple as creating random ingredient combinations with which we can create an infinite number of cocktails. With the evaluation system we could then choose the ones for which a nice taste is predicted.

To meet the main objective we formulated an assignment. The assignment of this thesis is to:

- design a model for the irrational set of taste, knowing that taste may change under time and context. Also as people cannot express their feeling of tastefulness in absolute statements we need to find a way to get the internal reference model of the user.
- design a model for evaluation providing a mechanism in order to make predictions for the evaluation of cocktails. The model needs to be able to handle the irrational nature of taste and therefore keep track of changes in the taste of the user.

- implement a prototype application, based on the taste and evaluation model, to create a test environment. With the application we can test whether the proposed models work or not.
- test the system for a bounded set of cocktails. See if the system can assess the taste of an a priori set of cocktails. Also test how it adapts to changes in the cocktail taste of the user.

1.5 Overview of the Thesis

Chapter two goes further on the irrational sets by explaining Irrational sets from different view points. Then the Irrational system characteristics and a model for handling irrational sets are discussed to give an indication of the sort of system we are trying to create and which mechanisms are required. Also several types of machine learning are explored to see which ones are either useful or completely inadequate in handling irrational sets.

Chapter three shows how to model the irrational set of taste, which may change under time and context. Also as people cannot express their feeling of tastefulness in absolute statements, a model is presented to get the internal reference model of the user and providing a mechanism for updating. This all is needed in order for us to keep track of the user's taste.

Chapter four presents the new genetic model for hypotheses generation and evaluation. Hypotheses represent possible evaluation functions that represent the user's personal evaluation. The model will show how we can have a set of hypotheses to which we assign certain beliefs to create a belief profile. The hypotheses are then used to predict the taste of any cocktail. For the belief profile we can intelligently create new hypotheses to predict even better and to keep track of shifts in the taste of the user. The genetic generation of hypotheses is based on ΔLogic, which will be explained, in combination with the Belief System.

The System Design for the implementation of the genetic model is presented in Chapter five, including a work flow diagram showing how the system works all together.

Chapter six presents test results of a test case showing how the system approximates the a priori ratings and how well it handles changes. The conclusion and future work are part of the final chapter, chapter seven.

Chapter 2

Irrational Sets

The definition of an Irrational set was given as a set where no finite set of rules can be constructed that can include unambiguously any member of that set and, at the same time, unambiguously exclude any non-member of that set (see [4]).

Which means that an irrational set is a type of set where memberships may change in an unpredictable way. This could create conflicts or ambiguity resulting in irrationalities and uncertainty. We need to use the irrational set to model the changes in the problem domain which in our case means modeling the user's cocktail taste.

B. Visscher (see [17]) presents a new approach to software engineering to handle irrational sets. I'll use one of his suggested mechanisms for the continuing updating of irrational sets in the problem domain, using the rational sets in the programming domain.

Irrational sets can be explained from different view points, so first I want to make clear how we should see irrational sets (§ 2.1). Then we can move on with the actual characteristics of mechanisms for dealing with these irrational sets (§ 2.3). Based on the needed mechanism an architecture for an irrational system is proposed (§ 2.4). The introduction of irrational sets in our model has some severe consequences for inference mechanisms for which we need a solution (§ 2.5).

The system characteristics for handling irrational sets will show that not every type of Machine learning is suitable for handling irrational sets. I'll discuss a list of most commonly known AI techniques to see which ones are either useful or completely inadequate (§ 2.6).

2.1 The Irrational view

There are two optional views that can be taken about the nature of the world from which irrational sets are explained:

- The Rational view, assumes that everything can be described in terms of rational sets, which is currently adopted by many scientists who adopted require formal models in their field, statistics economics... and represents the current paradigm of Computer Science. Rational sets would only appear irrational merely because at the moment we are dealing with incomplete and therefore imprecise information. This viewpoint represents the essence of Wittgenstein's Tractatus. Irrationality is considered subordinate to rationality and considered a temporary state of affairs that will be solved.

- The Irrational view, assumes that not everything can ever by potentially unambiguously describable. Sets within the world are not always rational. They could be, but there are irrational sets as well. We do not need to force a rational or irrational set onto the world.

The rational point of view is consistent and seems to give a complete explanation of the irrational sets, but there still are some limitations. If we would have an algorithm that couldn't explain a certain event due to the irrationality of a set, then it can always be said that we just didn't have enough data for a valid solution, our input wasn't complete. The problem is that this kind of argument can always be given. In this way we can avoid criticism by saying that we just need more data. The question how much data is enough? can never be answered. This could in fact lead to very expensive algorithms for simple problems.

A second limitation is the difficulty in dealing with human views and conceptualizations. Given the rational view the whole world consists of rational sets, but people are limited in the amount of data they can collect. People just can't see the world completely rationally as we can't know everything. For people the world appears to be irrational and they just try to deal with the irrationalities they encounter. Human communalization, understanding and conceptualizations are based on irrational sets.

So if we want a system to deal with human views and conceptualizations, then we also need to be able to handle irrationality and not force a rational set onto it. For this we need mechanisms to deal with irrationality as the behavior of irrational sets is distinctly different from that of rational sets.

Having a world that is completely rational according to the rational view doesn't mean that we can always create a model that gives a completely rational description of the world. Working with irrational concepts might be more practical.

To overcome the limitations of the rational view, we need to adopt the irrational view, but this has some other consequences as well.

With the irrational view we can now have irrational sets to describe the world, which means that some sets can't be described perfectly. The consequence of this is that we can't predict the world perfectly. Instead of 'true' facts we can have in some cases only mere 'believed' facts about the state of the world. Beliefs may change over time and context and because of this rules that depend on true facts cannot replace human judgement. Fixed rules are unable to follow the changes of the irrational sets, while a person seems to possess the ability to track these shifts.

Still it's not always the case that we should work with irrational sets. Some area's like mathematics and physics that deal with mostly rational sets are often better of with the rational view, while Psychology and history are with irrational sets. A system acquiring the taste of a person certainly would benefit from the Irrational view as it contains sources for irrationality (§ 2.2) and needs to track the irrational belief of the user for which fixed rules are impossible.

A final important note when adopting the irrational view are the following differences. With the rational view any set boundary is static and could be arbitrarily closely approached by refining the input, assuming that additional input is available and that it contains useful information with which the boundary could be defined. These assumptions are contrary to what the irrational view assumes, where input is limited and that we could come to a point where the input can no longer be refined. For the irrational view this means that we can no longer rely on more data means better approximations. We can no longer make the excuses that we need more data as it could just be that due to the irrational nature of the set a better approximation is impossible. Within the irrational view the set boundaries aren't static but dynamic.

2.2 Sources for Irrationality

B. Visscher (see [17]) speaks of 7 sources that show why a set may appear as irrational, which gives us an indication of the kind of problems we need to solve:

- **Noise**; Imprecise input results in imprecise membership for the set. We can't overlook wrong cocktail ratings given by a user.
- **Ambiguity**; It is impossible to make all set classifications. What if I like a cocktail as much as I dislike it?
- **Generalization**; If I like a cocktail with coke does it mean that I like every cocktail containing coke?
- **Specialization**; If I like a cocktail with coke, does it mean that I like coke?
- **Shifts**; What I like today doesn't mean I'll still like it tomorrow morning.
- **Disunity**; Introducing new sets when shifting.
- **Unity**; I can't choose a cocktail, I like them all.

If we provide a mechanism for each source to handle the irrationality, we can truly create an irrational system (§ 2.4), but first let us see the kind of characteristics of the mechanisms for handling irrational sets.

2.3 Characteristics of Mechanisms Handling Irrational Sets

As computers can only handle rational sets, how should we then handle irrational sets? A summery of the rational-set and irrational-set characteristics shows that we need a radically different mechanism. See Table 2.1 for some of these characteristics.

Table 2.1: Summary of rational-set and irrational-set characteristics

Set features	rational set	irrational set
Boundary	Static	Dynamic
Number of boundaries	Single	Multiple
Constraints for Algorithm	Consistent	Consistent / Contradictory
Learning	Always Converges	Does not converge, either stabilizes or diverges
Learning period	Finite	Never ending
Decision making	Deterministic	Non-deterministic
Mechanism	Contextualisation	Adjustment / Insight

Based on these characteristics creating a system that is able to handle shifting (irrational) sets, would need to have the following three characteristics:

1. Dependent on time. As shifts occur over time and cannot be predicted, the shifts would be reflected in the order.

2. The learning period never ends. The system must continue evaluating and learning when presented with new ideas and situations.

3. Showing some form of randomness. While in a rational system, every decision can be made completely deterministic to get the optimum response, within an irrational system, no such optimum can be found as the sets change.

With the characteristics of mechanisms handling irrational sets presented and knowing the irrational sources that need to be handled (§ 2.2) we can present the architecture for an irrational system.

2.4 Irrational system

An architecture for an irrational system is given by B.Visscher (see [17]) in a general overview of the different components (See Figure 2.1) and the information flow from one process to another. It shows how an irrational system could work and interact with its environment.

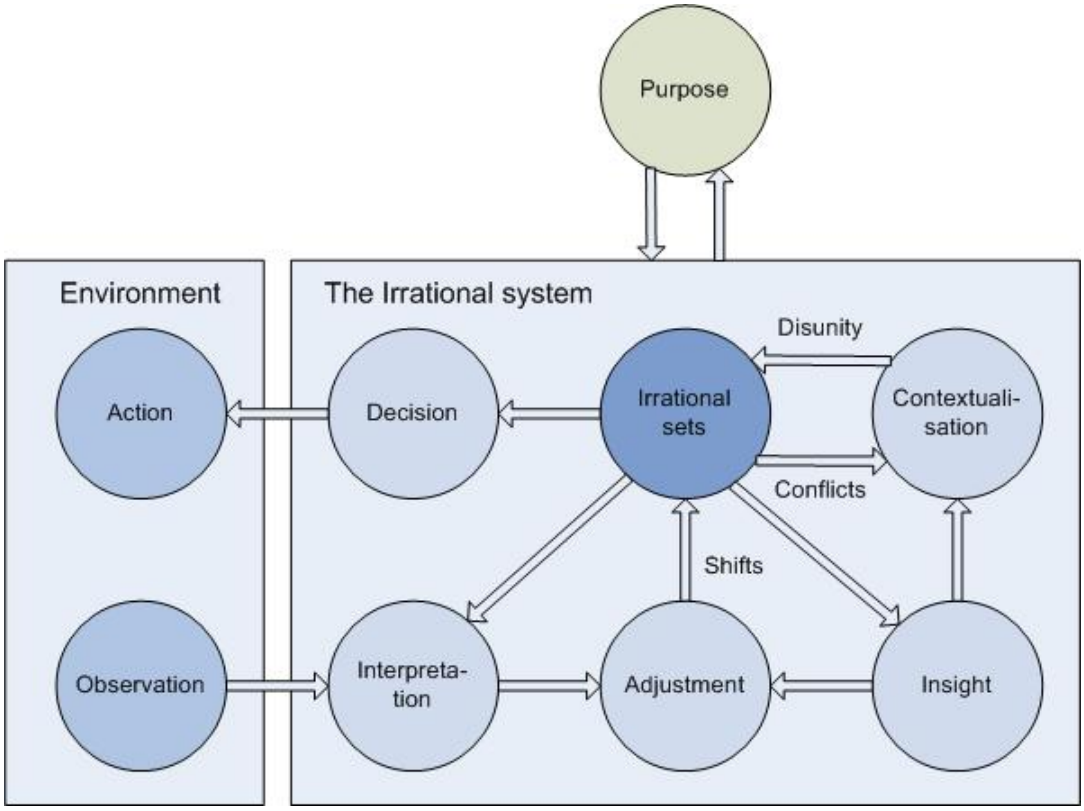

Figure 2.1: The Irrational system

Some of the components provide a mechanism for handling the different sources of irrationalities. The next four components handle the following sources:

- **Interpretation**; A mechanism to handle ambiguity. A set of contexts is chosen that create a consistent meaning.
- **Contextualization**; A mechanism to handle conflicts. Conflicts can be handled to create a new context allowing a consistent interpretation.

- **Adjustment**; Handling Generalization, Specialization, Unity and Shifts. As set membership is not static we need to be able to adjust the boundary of a set.
- **Insight**; Forcing Set Unity or Shifts. Needs to clean up disconnected conceptualizations.

So contextualization creates new contexts which solve a conflict at hand allowing a consistent interpretation, but when we are having multiple contexts of the same set, then we also need to know which set to use in a given situation. For this we need the system to evaluate the contexts against the data in the given situation and choose the appropriate context.

An example of a system that uses multiple contexts to make predictions about the result of experiments is the Belief System by T. R. Addis and D. C. Gooding (see [2]). Hypotheses represent the contexts, where belief in them can be adjusted based on the outcome of the experiments. The Belief System incorporates the Abductive model to reason with irrational sets (see § 2.5). The abductive model also represents the mechanism of Insight, (belief) Adjustment and Interpretation (deduction).

The Belief system may look like the ultimate irrational system already, but the thing missing is the ability to create new hypotheses, the system only works with a priori contexts. Providing the Belief System with a mechanism for hypotheses creation would make the Belief System truly an irrational system.

After discussing the abduction model I'll have a look at other AI techniques to find out if there are besides the Belief System any other types of machine learning suitable for handling irrational sets.

2.5 Abduction model

The introduction of the irrational set has some consequences for the inference mechanism. In case we would only work with rational sets, we can use the deductive inference. Deduction is purely syntactic as long as the coherence of the rational set is kept at all times. Irrational sets destroy the coherence of the sets, which mean that we can no longer use the deductive mechanism. Instead of deduction we may use the abductive inference loop modeled by T. R. Addis et al. (see [4]) and shown in figure 2.2.

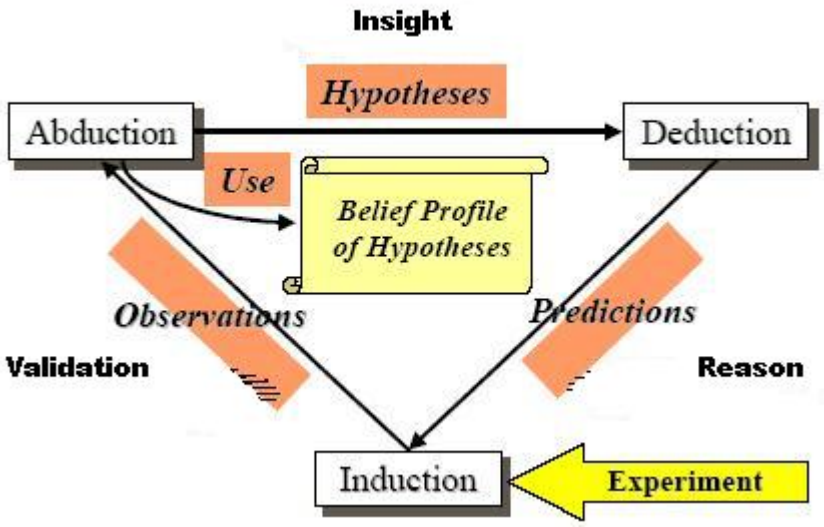

Figure 2.2: The Abductive Inference loop

In this model truth is only part of the deductive element. Outside deduction truth is replaced by belief. The system beliefs in a set of hypotheses which make up the formal part of the system used for deductive reasoning. In this way we can make predictions about the world. These predictions can be validated by observations in the world. Validation provides a mechanism through which conclusions can be justified. The process of validation adjusts the beliefs and this means we adjust the formal model.

What is important to notice here is the need for purpose. Without purpose there is no criteria for success and therefore no mechanism for validation (see [1]).

2.6 Machine learning

For an intelligent system we need an AI technique capable of dealing with irrational sets. The Irrational set characteristics in previous table 2.1 together with the three system characteristics (Time dependent, never ending learning period and Randomness) demand certain requirements on the type of machine learning. For the next five most commonly known AI techniques is discussed which ones are either useful or completely inadequate:

1. **Knowledge Based Systems (KBS)**, problem solving by a general-purpose search mechanism trying to string together elementary reasoning steps with a reference to a database of knowledge on a particular subject to find complete solutions (see [14]). Expert systems and Case-Based-Reasoning are good examples of Knowledge Based Systems. The typical expert system consisted of a set of facts, a set of rules, and an inference engine. The inference engine applies a sequence of rules to the set of facts, thereby producing new facts.

2. **Neural networks**. Artificial Neural Networks are distributed, adaptive, generally nonlinear learning machines built from many different processing elements (see [11]).

3. **Genetic algorithms (Machine Evolution)**. These are Global search procedures, proposed by John Holland, that search the performance surface, concentrating on the areas that provide better solutions. They use generations of search points computed from the previous search points using the operators of crossover and mutation (see [11]).

4. **Bayesian Methods and Learning**. Examples are Bayesian belief networks, a directed acyclic graph of nodes representing variables and arcs representing dependence relations among the variables(see [10]) nodes represent random variables, which are connected through edges that represent causal relations. When new evidence is presented, probabilities are propagated through the network in a consistent way. Another example is the Belief System (see [2]).

5. **Fuzzy Systems**. Fuzzy logic is an extension of Boolean logic dealing with the concept of partial truth. Whereas classical logic holds that everything can be expressed in binary terms (0 or 1, black or white, yes or no), fuzzy logic replaces boolean truth values with degrees of truth (see [10]).

(1)Expert systems analyze information by a set of rational rules and according to their given relations they fire other rules recommending a course of user action. Knowledge Based Systems are in this way deductive systems. Knowledge Based Systems work well when the problem domain is static and well defined or in other words it works well with rational sets. As soon as we need to work with a problem domain with irrational sets we get into trouble. For instance when we try to model or imitate human inference as done by the medical expert system MYCIN (see [13]). To compensate for the irrational nature of human inference, E. H. Shortliffe introduced so called certainty factors, which were intended both as an engineering solution and as a model of human judgment under uncertainty. Although some expert systems of this kind work quite well, the problem in other cases is that certainty factors tend to produce contradictions. With certainty factors we cannot handle irrational sets, because the assignment of values is arbitrary, but ones it is set they will remain fixed. There is no introduction of new possible concepts, or the option to assess a member to a new set. This makes it unable to track shifts.

Further more we do NOT see any Non-deterministic decisions in an expert system, because this would make the whole process of reasoning invalid. So besides the troubles when handling uncertainty, it also doesn't show any form of randomness.

Case-Based-Reasoning suffers from the same problem as with Neural Networks, discussed next.

(2)While Neural Networks are most commonly used for adaptive learning systems, they are not suitable handling irrational sets. This because they fail on the second system characteristic that the learning period should never end. Keeping track of irrational sets, by shifting a rational set, must continue forever. The problem with a neural network is that it suffers from so called overtraining of the network. If we want the Neural network to generalize correctly, then there is a certain time after which we should stop training the network, because the performance on the test set will otherwise deteriorate.

If we want to use a Neural Network to track changes, then we need to keep on feeding the network with new training samples, which leads to overtraining. The reason why the performance deteriorates can be explained through the notion of irrational sets. The more samples we keep on using for training, the more noise we introduce in the system. We can have more shifts and as a consequence of that more variation in the boundaries. When there is to much variation, then we can't make any good predictions.

The problem is that we keep the noise of every training sample we feed to the network. The effect of the training sample when adjusting the network will last forever. Every sample has just as much effect on the present state of the network. This means that 'old' training samples keep on effecting the present even when they are no longer valid due to a shift in the problem domain.

It's not only Neural Networks that have this problem, also Case-based-reasoning and Baysian belief networks.

(3)Genetic Algorithms have the property that allows them to keep track of irrational sets, as their solution may constantly evolve. The purpose of the system is set by the 'world' and can change at any time and under any other context.

Time (order) dependence is present, where systems evolve in time. Randomness is introduced with the breeding of genes where crossover and mutation provide for the necessary variation.

(4)As described by T. R. Addis and D. C. Gooding (see [2]), the belief system is an ideal learning system. It invokes Bayes Rule (this behaves like a three layer neural net) but has the added advantage of being adaptive in a way that reflects irrational sets. It doesn't use Bayes rule for belief revision, but instead it uses a different updating rule based on responsiveness to new evidence (See § 4.18).

This in contrary to Baysian Belief Networks which use Bayes Rule for learning. Bayes Rule assumes a constant and unchanging world and that the order of the events is irrelevant, therefore we could not use it to handle irrational sets as time order dependence is crucial. With the non-Basysion belief-revision update rule in the Belief System it has the characteristic of time (order) dependence.

Further more the Belief System has the ability to learn forever and shows randomness by involving game theory for choosing experiments. The only shortcoming of the system is that it is not yet able to act creatively. It still needs some form of hypotheses generation where it now only works with an a priori set of hypotheses. This means that it will eventually break down.

(5)For Fuzzy Systems the Fuzzy sets are often being mistaken for irrational sets, but one should be reminded that fuzzy sets remain in the rational domain.

"We can extend the classical set (Chair=1, not-Chair=0) by

> assigning an intermediate value to membership (e.g. 0.5) as an attempt at providing a rational description for irrational sets. Examples include fuzzy and probabilistic membership assignments. However, fuzzy sets are rational in that members are assigned a membership value that is unchanging. The value is represented in any implementation by a finite number. Such assignments can therefore be expressed by a finite set of rules and are therefore within the domain of rational sets."(see [4])

The next table gives an overview of the different types of machine learning and their compatibility with Irrational sets.

Table 2.2: Machine learning Compatibility with Irrational sets

Machine Learning type	Able?	Comments
Knowledge Based Systems	No	Ambiguous rules are not allowed, No form of randomness
Neural networks	No	Unable to learn forever, due to overtraining
Genetic Algorithms	Yes	Constantly evolving and randomness is present
Belief system	Yes	Time dependent, could learn forever, uses Game Theory
Fuzzy systems	No	A Fuzzy set is not an irrational set

The best bet still seems to be on the belief system, but with the addition of a genetic algorithm to create new hypotheses. Chapter 4 proposes a model for this combination, but first we show a model for the irrational set of taste in the next chapter.

Chapter 3

Initial model

A system that invents the kind of cocktails needs to have some sense of what the person likes. The system should reflect the user's taste in the way it evaluates cocktails. To do this the user's taste needs to be assessed. There are two problems here. First taste tends to be irrational. In time and under different contexts taste changes.

The second problem is that people cannot express their feeling of tastefulness in absolute statements. To like something is an emotional feeling with only internal and private references. When two persons both rate a cocktail with an 8, then it doesn't mean that they both have the same taste experience. Taste can only be put on a relative scale, where only greater or less than comparisons can be made. I can't exactly know what another person feels when eating a banana as I don't know his private reference scale, but I can ask whether it tastes better or worse than apples.

The question is how we can model the internal references of a person and account for irrational shifts. T. R. Addis and D. Billinge give a possible solution (§ 3.1).

3.1 Abstract model

D. Billinge and T. R. Addis (see [15]) present an abstract model providing a means of resolving cocktail taste into numeric measures and program labels (see figure 3.1).

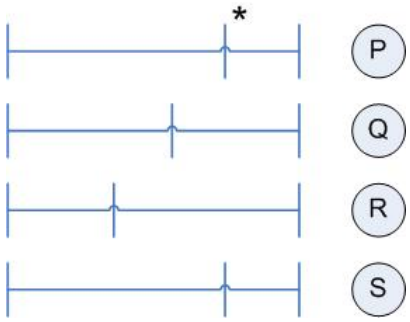

Figure 3.1: Abstract model of internal reference

For one person we represent his taste, where the Cocktails are represented as P to S. The lines represent a simplified single-emotional reaction to Cocktails P, Q, R and S, namely tastefulness. The extent to which Cocktail P person A likes is shown by the position of the bisecting vertical line at *, the horizontal line representing the continuum between totally-dislike on the left and fully-like to the right.

We create a functional model which implements the internal model of continua to represent strength of taste. The strengths can be adjust in conversations with the user.

3.2 Functional model

The continua of strength of taste is represented by a 7-part scale, splitting the dimension into 7 different hypotheses. Each hypothesis is treated as a value drawn from a set of competing hypotheses. The 7 hypotheses represent an ordered set of values, where we use 0 to represent dislike and 6 to represent absolutely liking it:

$$0\ (\text{dislike}) < 1 < 2 < 3(\text{neutral}) < 4 < 5 < 6(\text{like})$$

The amount of belief in each of the hypotheses represents the rating distribution for a particular cocktail. Since belief in the hypotheses must never exceed unity; the maximum value of 1 is distributed over a 7-part scale. If the person hasn't gained any knowledge about the taste of the cocktail, then the maximum value of 1 is distributed evenly over the 7-part scale:

$$[\ 0.143 \mid 0.143 \mid 0.143 \mid 0.143 \mid 0.143 \mid 0.143 \mid 0.143\ ]$$

Choosing the number of hypotheses (values) is quite arbitrary, because the values are relative values. This means that value 3 and 7 mean the same in respectively a 3-part and 7-part scale. As long as we have more than one hypothesis, which is necessary to discriminate, we can have any number of hypotheses. The choice for an ordered set of 7 values is because we wanted a set with a center value. A center value represents a neutral position saying I like it as much as I don't like it. While every odd number will give us our center value, we chose 7 values, because it seems not too limited and not too vast.

3.3 Updating

After experience of a single cocktail the person can only make 7 absolute statements, while internally (inside the person's mind) the cocktail will have triggered a degree of likeliness, but which cannot be expressed. In our model we will represent this personal experience as a belief of 1 for one of the seven positions (hypotheses) on the continuum. For example rating 6 is:

[0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | **1.0**]

By giving a cocktail one of the 7 absolute ratings, the user can update the taste of the cocktail. The internal model is changed by adjusting the belief in the 7 rating hypotheses. The given absolute rating is compared with the predicted rating by the hypotheses, where the predicted rating is calculated as follows:

$$E(Cocktail) = \sum_{H} E(H) * H(Value) \tag{3.1}$$

Because the ratings are represented by a relative scale, only greater or less than comparisons can be made between two rating scales. The belief adjustment is made by comparing the predicted value with the given absolute rating by the user and based on the result shifting the hypotheses belief distribution to the left or the right.

Based on whether the absolute rating is greater or less than the predicted value, its belief is equalized over respectively the hypotheses above and below the rating. This is done because if for example rating 4 is greater than the predicted value, then so are rating 5 and 6. Hypotheses 5 and 6 should then get an increase in belief as well. Table 3.1 shows both examples for an absolute rating of 4.

Table 3.1: Greater or less than

Rating	Greater than	Result
4	TRUE	[0|0|0|0|1|0|0] → [0.0|0.0|0.0|0.0|0.33|0.33|0.33]
4	FALSE	[0|0|0|0|1|0|0] → [0.2|0.2|0.2|0.2|0.2|0.0|0.0]

The hypotheses belief distribution will be updated with the adjusted absolute rating distribution using the following equation (See § 4.18 where the same update rule is used to adjust beliefs):

$$E(H) = \frac{(N-1)E_{n-1}(H) + E_{n-1}(H/R_e)}{N} \tag{3.2}$$

Here Flexibility is defined as $\frac{1}{N}$ and reflects responsiveness to new results. So with a high flexibility, the belief modification "listens" more to the given rating.

Table 3.2 and 3.3 give two examples of updating the following belief distribution with a predicted value of 3, with respectively a higher rating (4) and a lower rating (1). The Flexibility is set at 0.35.

[0.107 | 0.107 | 0.107 | 0.357 | 0.107 | 0.107 | 0.107]

Table 3.2: More

Step	Description	Result
1	Create a 7-part scale	4 → [0|0|0|0|1|0|0]
2	Accumr	[0|0|0|0|1|0|0] → [0|0|0|0|1|1|1]
3	Normalize the distribution	[0|0|0|0|1|1|1] → [0|0|0|0|0.333|0.333|0.333]
4	Change belief value, given flexibility The resulted distribution is clearly shifted to the right	[(1-F)*0.107 + F*0 | (1-F)*0.107 + F*0 | (1-F)*0.107 + F*0 | (1-F)*0.357 + F*0 | (1-F)*0.107 + F*0.333 | (1-F)*0.107 + F*0.333 | (1-F)*0.107 + F*0.333] → [0.070|0.070|0.07|0.232|0.186|0.186|0.186]

Table 3.3: Less

Step	Description	Result
1	Create a 7-part scale	1 → [0|1|0|0|0|0|0]
2	Reverse	[0|1|0|0|0|0|0] → [0|0|0|0|0|1|0]
3	Accumr	[0|0|0|0|0|1|0] → [0|0|0|0|0|1|1]
4	Reverse	[0|0|0|0|0|1|1] → [1|1|0|0|0|0|0]
5	Normalize the distribution	[1|1|0|0|0|0|0] → [0.5|0.5|0|0|0|0|0]
6	Change belief value, given flexibility The resulted distribution is clearly shifted to the left	[(1-F)*0.107 + F*0.5 | (1-F)*0.107 + F*0.5 | (1-F)*0.107 + F*0 | (1-F)*0.357 + F*0 | (1-F)*0.107 + F*0 | (1-F)*0.107 + F*0 | (1-F)*0.107 + F*0] → [0.245|0.245|0.070|0.232|0.070|0.070|0.070]

3.4 Creating new cocktails

We now have an internal model of the users taste, but there is no mechanism yet to create a belief model for an unknown cocktail. There is no mechanism for evaluation, no measure for success. We always need to ask the user for a first evaluation, but we want the system to be able to evaluate for itself not having to ask the user each time, avoiding an endless conversation with the user. If the system has hypotheses how to predict, we need less information.

So besides getting the internal models of known cocktails, we need to have a mechanism to generalize and predict models for unknown cocktails. The system should learn the ever changing internal models of some cocktails in order to make predictions in the evaluation of new cocktails. Here we should note the dynamical aspect of the learning set as taste tends to be irrational. In time and under different context taste changes, meaning different internal models.

As soon as we have a mechanism for evaluation we can design an algorithm for creating cocktails and evaluate before hand whether the user will like it or not.

The next chapter will discuss a model for cocktail evaluation based on a combination of the Belief System and a genetic algorithm.

Chapter 4

Genetic Model

In this Chapter I will discuss the model I designed for an irrational cocktail evaluation system.

The system needs a mechanism for evaluation in order to make predictions in the evaluation of cocktails. In our case evaluation means that given a list of cocktail features and ingredients a person's internal taste model is returned, a belief distribution over the 7-part scale of rating hypotheses. What we are looking for is the relation between each combination of ingredients and its corresponding rating. If we can define a function that represents this relation, then we could apply it onto any cocktail combination and we have our needed evaluation function. An evaluation function given its input parameters (cocktail features) returns the appropriate rating, where R is a rating distribution over the 7 hypotheses:

$$F(f1, \ldots, fn) \rightarrow R$$

A system with evaluation functions doesn't limit us to work only with cocktails, but can be used in all sorts of cases where evaluation is needed.

Perhaps one or more evaluation functions can be found that predict the user's rating and thereby approximate his or her evaluation behavior. But before we get to this point we need to answer two other questions first:

- How do we define a cocktail evaluation function?
- How can we get the optimal evaluation function?

The answer to the first question is related to the second one, so let me first answer that one.

To get the optimal evaluation function I came up with a completely new model, the Genetic Belief System model presented in figure 4.1. A Genetic system that creates evaluations functions in order to predict the user's taste.

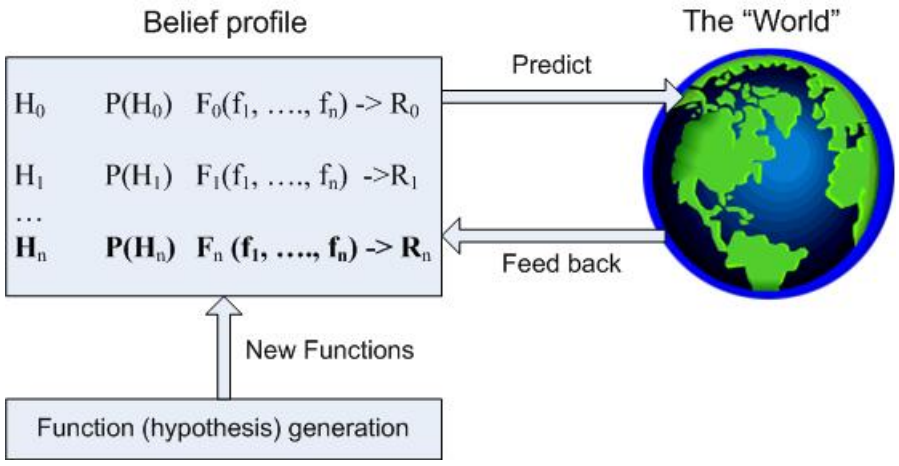

Figure 4.1: Genetic model

A set of evaluation functions will form the belief profile, where each evaluation function is in fact an hypothesis for the Belief System. The Belief System will figure out which evaluation functions predict the best.

The current Belief System works only with an a priori fixed set of hypotheses. The new model introduces the creation of new hypotheses by generating new evaluation functions with a genetic algorithm.

It is important to notice that the hypotheses for the Belief System are evaluation functions and not the hypotheses which represent the rating distribution for a particular cocktail which were discussed in previous paragraph 3.2. The (seven) hypotheses for the rating scale represent the output of an evaluation function, where the evaluation function is a hypothesis of the Belief System.

I will show how I made a genetic algorithm for evaluation functions. For this we need to define the cocktail evaluation functions with ΔLogic, which answers the question how we define the evaluation functions. Evaluation functions defined with ΔLogic make it possible to transform evaluation functions into genes. ΔLogic is discussed in the next section.

4.1 ΔLogic

4.1.1 Introduction

ΔLogic is an alternative type of logic, introduced by B. Visscher(see [16]), that works with detecting differences between input signals. B. Visscher tries to increase the understanding of the brain by introducing a more natural logic to model the excitation/inhibition behaviour of biological neurons with than the traditional Boolean Logic. He introduces Δneurons sharing some resemblance with the behaviour of biological neurons. Using only these Δneurons in a network it is possible to create any Boolean function of any number of inputs. These networks of Δneurons can be represented by so called Δprograms.

It is with these Δprograms that I designed my genetic algorithm by mapping Δprograms onto genes. The idea is to build Δneuron networks for our cocktail evaluation functions and with their Δprogram representation map them onto genes (see figure 4.2). These genes could then be used by genetic algorithm I designed to generate new genes, meaning new Δprograms, new networks, new cocktail evaluation functions.

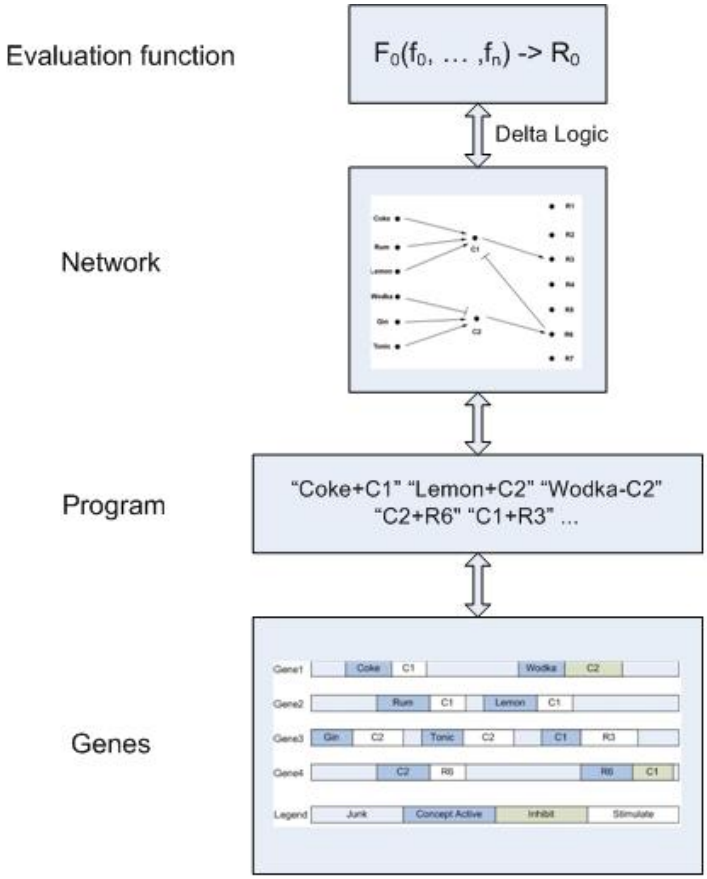

Figure 4.2: Mapping Evaluation functions onto Genes

To explain how this all works in detail, let's start at the beginning and explore the basics of ΔLogic.

4.1.2 Basics

ΔLogic consists out of five basic components, an environment, Δsignals, connections isolated from the environment to transport a Δsignal, the inhibitor and the combiner that combine two Δsignals in a specific way.

For the Δsignals there are two signal elements. The reason for this is that it is not possible to create a useful signal on its own, because a signal can only be used if it is detectable and it is only detectable if it differs from another signal. One of the two is called the environment signal, denoted by θ. The second one is the Δsignal that can have either the value θ or 1, where θ means the Δsignal is the same as the environment and can therefore not be detected using only the environmental signal. 1 meaning it differs from the environmental signal, which can be detected and used to send either θ or 1 on the output.

The ΔLogic functions are made with the following three components:

- Connection. The connection element is used to connect the other two elements together. A ΔIn signal is always connected to one ΔOut signal and a ΔOut signal is connected to 0 or more ΔIn signals.

Table 4.1: Connection

ΔIn	ΔOut
θ	θ
1	1

- Combiner. Combines the inputs and puts out 1 when one of the inputs is set to 1.

Table 4.2: Combiner

ΔIn1	ΔIn2	ΔOut
θ	θ	θ
θ	1	1
1	θ	1
1	1	1

- Inhibitor. The behaviour can best be described where a 1 on ln1 inhibits ln2 to be outputted.

Table 4.3: Inhibitor

ΔIn1	ΔIn2	ΔOut
θ	θ	θ
θ	1	1
1	θ	θ
1	1	θ

With the previous described components we can't yet create all possible Boolean functions. Within ΔLogic it is not possible to detect and react to the θ state using only the environment. Because of this, it is never possible to create the inverse of the θ state of a Δsignal within ΔLogic. It isn't possible to say, where NOT ΔIn = θ:

IF NOT ΔIn THEN ...

There are two ways to overcome this problem:

1. Introducing a second constant to the system that has always the value 1 to be distinguishable from the environment.
2. Single concept representation

(1)The first option is more the traditional solution. Here we only need the Inhibitor where the second input is connected to the 1 value. In this way we have created an invert gate for the first input. (See figure 4.3)

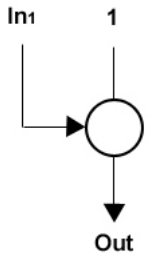

Figure 4.3: Invert gate with Inhibitor and constant

(2)The second option is the use of single concept representation. Because we only have the ability to detect and react to the 1 state and not to the θ

state, there is an asymmetry between the θ and 1 state of a Δsignal. It has one usable state and can therefore only represent one concept. This in contrary to Boolean logic where we assign two concepts to one signal, namely True and False. So now to represent one Boolean signal, two Δsignals are needed, one for True and the other for False. This approach makes inverting a signal trivial as we only need to exchange the True signal with False and visa versa.

Both solutions work and it comes down to the personnel choice of the programmer. The second solution, single concept representation stays close to the way in which biological neurons in the brain work. The Brain does not appear to have any constant signal other than the environment. But the problem is that with single concept representation the input space is huge, it is a very inefficient solution as we need to double the number of input concepts. So I will continue using the first solution, a constant value.

4.1.3 Networks

The Δneuron is an additional element that can replace both the Inhibitor and Combiner. It is a more general element and simplifies the job of network creation. In fact it is a combination of the Inhibitor and Stimulator shown in figure 4.4.

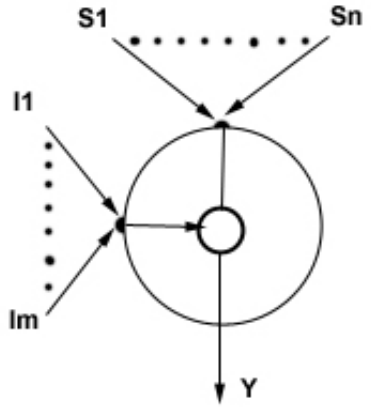

Figure 4.4: Δneuron with N stimulation and M inhibiting ΔInputs

In the network a Δneuron can be the Stimulator or Inhibitor of another Δneuron (DN). These two relations are defined as follows:

- Stimulating, a DN can stimulate another DN. This means that whenever the first DN puts out a stimulation (put to 1), than it will stimulate the second DN (putting it to 1). This is a many-to-many relationship. DN's can stimulate multiple DN's and DN's can be stimulated by multiple DN's.
- Inhibiting, a DN can inhibit another DN. This means that whenever the first DN puts out a stimulation (put to 1), than it will inhibit the second DN (putting it to 0). This is a many-to-many relationship. DN's can inhibit multiple DN's and DN's can be inhibited by multiple DN's. There is only one inhibit necessary to inhibit the DN, no matter how many stimulaters it may have.

An example of a network structure is displayed next. Its an example referring to cocktails where we have a list of 6 input cocktail features (ingredients), 2 intermediate states or concepts representing cocktail combinations, and 7 outputs representing possible ratings.

There are two different concept relation arrows to show whether a concept stimulates or inhibits another concept. These are –> and –|, which respectively stand for stimulates and inhibits.

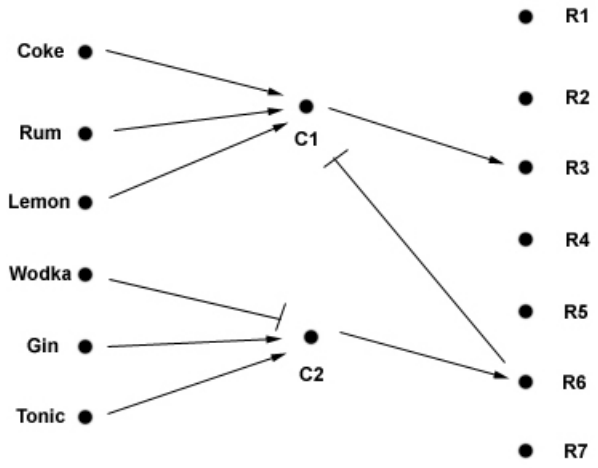

Figure 4.5: Network build with Δneurons (Concepts)

It is quite important to note the differences between this type of Δneuron network and an Artificial Neural Network (ANN) based on the McCulloch-Pitts neurons. As to what seems to be quite similar, there are still two major differences:

- First of all the Δnetwork is not learning anything. In contrary to the ANN it has no weights to update, meaning that it will not adapt. Its only there to represent a function, where the ANN can be trained to adapt its function.
- Secondly the ANN is computed in a one way direction, having layers of neurons only effecting the next layer, but to compute the output for a given input on a Δnetwork we need to proceed differently. As there is no sequential ordering in the stimulating and inhibiting of Δneurons, we need to compute in parallel. The Δnetwork may also contain loops or Δneurons stimulating/inhibiting previous Δneurons. In this way we can have a never ending progress through the network. One way to overcome this problem is to set a time limit. When the time is up we count the number of times each output concept has been activated, which gives us a probability distribution over the output concepts.

4.1.4 ΔPrograms

Since every active concept (Δneuron) that comes from either an input or from within the network can only stimulate or inhibit other concepts (Δneurons), the whole network can therefore be described in the following IF-THEN terms:

IF 'concept'=1 THEN Stimulate OR Inhibit 'concept'

This representation is shortened to two pattern influence rules or π's, one for stimulation and the other for inhibiting:

- π+ 'concept A' + 'concept B' 'concept A' stimulates 'concept B'
- π- 'concept A' - 'concept B' 'concept A' inhibits 'concept B'

We can now represent the entire network with these pattern influence rules, resulting in what we call the Δprogram. Take for example the Δprogram of the network displayed in the previously discussed figure 4.5:

'Coke'+'C1' 'Rum'+'C1' 'Lemon'+'C1' 'C1'+'R3' 'Wodka'-'C2'
'Gin'+'C2' 'Tonic'+'C2' 'C2'+'R5' 'R5'-'C1'

The Δprogram is now our evaluation program. As we can run the program to evaluate a given input where it will return a probability distribution over the outputs. At first it may seem strange how we can suddenly get a probability distribution over the outputs as we can only stimulate or inhibit an output concept. This will become clear as we take a look at the way in which I designed the program to run.

Running the program will work as described in the following steps:

1. First a time limit is set, because loops might be present.
2. The given input concepts will be set active and they may stimulate or inhibit other concepts (or themselves) defined by the pattern influence rules. This provides us with a new list of active concepts, without the previous active concepts that are not stimulated again.
3. The new active concepts will on their turn again stimulate or inhibit other concepts (or themselves), creating again a new list of active concepts. As long as the list with active concepts isn't empty we will repeat this step. In the mean time we keep counting the number of times each output is activated.

4. When there are no longer active concepts or when the time limit is reached, then the program has come to an end. The probability distribution is then calculated by normalizing the number of activations for each output.

The next table gives a simple example of running the previous Δprogram (represented by the network in figure 4.5). The program evaluates an input of four features: Coke, Rum, Lemon and Tonic.

Table 4.4: Δprogram running example

Step	Active concepts	Pattern influence rules
2	Coke, Rum, Lemon, Tonic	Input
3	C1 and C2	'Coke'+'C1' 'Rum'+'C1' 'Lemon'+'C1' 'Tonic'+'C2'
3	R3 and R6	'C1'+'R3' 'C2'+'R6'
4	empty	-

At the point where the program reaches an empty list of active concepts only the R3 and R6 output have been activated once. So the resulting probability distribution over the outputs (R0...R6) is after the normalization of counts:

(0 0 0 0.5 0 0 0.5)

With the Δprogram we now have a program in a formal language to define our cocktail evaluation function. As the ΔLogic is complete it enables us to create any function possible. Out of all possible functions we need to find the best evaluation function, meaning we need to construct the best fitting Δprogram. One way to create the best Δprogram is by using an evolutionary genetic algorithm. The genetic algorithm should evolve better and better solutions by breeding offspring and creating new populations of evaluation functions. So we want to breed new Δprograms out of (initial) existing ones, by using the biological inspired methods of crossover and mutation. In order to employ a genetic algorithm we first need to create its building blocks, Genes.

4.2 Genes

With the use of ΔLogic I'm able to design genes to construct my genetic algorithm, by mapping ΔPrograms onto genes. I'll show how the genes are designed and coded.

4.2.1 Gene Design

To create the necessary genes for the genetic algorithm the first step is to somehow map the Δprogram onto genes, meaning we need to get the pattern influence rules onto genetic strands. One way to do this is presented in the next figure 4.6.

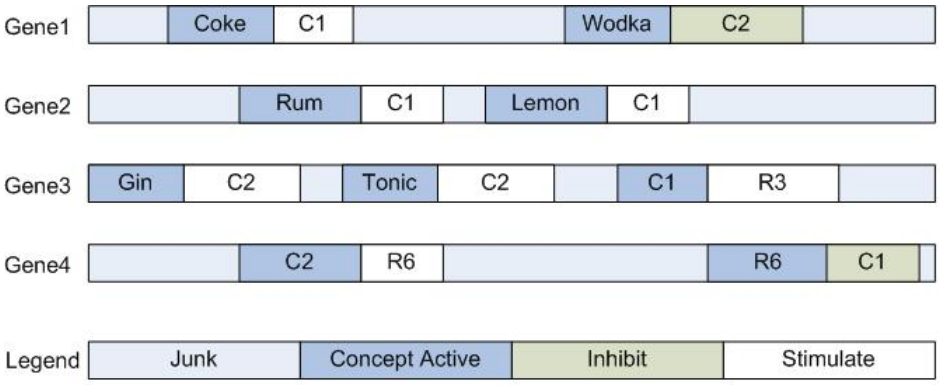

Figure 4.6: Example of a Δprogram mapped onto genes

Genes are in this way composed with valid π's and Junk, where Junk is what fills up the genes with sequences for which no function has been identified. The reason for introducing Junk is inspired by what can be seen in nature, referred to by "Junk DNA"(see [7]). Having junk should give the following advantages:

- Much junk gives point mutation a smaller change to have any effect. In this way junk acts as a protective buffer against genetic damage (errors) and negative point mutations.
- Junk could provide a reservoir of sequences from which potentially advantageous new π's and concepts can emerge.
- Functions switched off by mutation, remain present in the genes and could be switched on again by future mutations. In this way we can have dormant gene parts.

The concept of junk in combination with a genetic algorithm is quite novel and the use of junk raises some questions as well. For instance:

1. How big is the junk proportion of a gene?
2. Does junk DNA really exist or does it just have a function that we can't detect?

(1)A possible answer to the first question is to take the example of junk DNA presented by nature, where 97% of the human genome has been designated as junk (see [7]).

(2)The second question seems to be very important, because if it would turn out that even junk DNA has a function, then the whole concept of junk is useless. We are stuck with an empirical problem, because we can't prove (yet) that junk DNA has no function. If we can't detect a function doesn't mean it isn't there. So for now we can't give a solid answer to the question whether junk really exist or not, but acknowledging the presents of junk is in line with what has been seen and explored by present science. Also the given advantages seem to make junk very plausible.

To map a Δprogram onto genes by composing genes with valid π's and Junk will be explored further, discussing one way to implement this.

4.2.2 Gene Coding

One way to implement the genes is to create genetic strands of bit strings. In this way a gene is represented by a string filled with zeros and ones. The bit string can be read from left to right in order to find the encoded π's between the junk. To explain how this works let us start with the following example presented in figure 4.7.

Figure 4.7: Gene Example

If we want to identify valid π's along the bit string, then we need the use of bit code Identifiers. Take for example identifier '00000', whenever we find bit string '00000' inside the gene it means that it is followed by a valid π. In our example this means that the two π's in the string are preceded by '00000'.

'00000' 'Coke'+'C1' and '00000' 'Wokda'-'C2'

The next step is where we need each concept to be represented by a unique bit code. When we build the genes for the first time, then we start with the list of known concepts which are present in the network and Δprogram (input ,output and intermediate concepts). But in time we want new concepts to emerge from junk or by concept mutation. These emerged concepts should have their own unique bit code. Paragraph 4.2.3 describes the mechanism for the needed progressive bit state coding. For the example we could have the following unique bit codes:

'00000' '0101'+'1000' and '00000' '1100'-'1110'

Because there are only two types of π's (+ and -) we can replace the + (stimulator) by a 1 and the - (inhibitor) by a 0.

'00000' '0101' 1 '1000' and '00000' '1100' 0 '1110'

With all this every part of the Delta-Program can now be mapped onto a string of bits. The only thing missing now is Junk. Junk will be represented by a sequence of random numbers of zeros and ones. Junk cannot contain the '00000' sequence, because this infers that we have found an identifier after which follows a valid π. Adding some junk to our example results in the following possible sequence of bits for Gene 1. (The | is added at some places to increase readability, where they separate junk, identifiers and patterns)

0100110101|**00000**|**0101**|**1**|**1000**|11010101010|**00000**|**1100**|**0**|**1110**|0110

It is not necessary for a Δprogram to be mapped onto only one gene, the valid π's may be spread over multiple genes. For now there are no guidelines or rules of thumb for the number of genes, but the genes themselves should consist for about 97 percent out of junk. Genes don't have a fixed length as the amount of junk and the number and length of π's may vary. The fact that also the length of concepts may vary is due to the used concept coding which is discussed in the next paragraph.

4.2.3 Flexible bit state coding

As stated earlier, each concept needs to be represented by a unique bit code to distinguish between concepts. This would have been easy if the list with concepts was a fixed set, but in time we want new concepts to emerge from junk or by concept mutation. The reason why we want new concepts is because they could be potentially advantageous in forming the best evaluation function. Only there isn't really an ultimate everlasting evaluation function to be found (or approximated), due to the irrational nature of it. People will change their opinion on taste in time and context. So we need to be able to form new concepts to track the user's taste.

Another point is being dynamical. For instance when we would like to add new concepts (ingredients) to our input space we don't want to encode the genes all over again. Introducing new concepts asks for flexible coding.

What we don't want is to start by stating a fixed number of bits to represent the concepts, say for example a number of 8, because then we always have a limit to the number of concepts, in this case 8bits = 256 states. We don't want this fixed limit, when we reach the limit we want to expand the number of bits representing states. If we would state a fixed number of bits, then we are limited in our flexibility. Flexible bit state coding means that the length of concepts may vary. This makes it a bit more difficult when we start reading the gene bit string from left to right decoding the concepts.

Reading the gene bit string from left to right we should be able to find the π's not knowing before hand which concepts we will find. The only thing we can find in the first place is the π identifier. We start looking for the first identifier sequence, which is "00000" in the next gene bit string example.

0100110101|**00000**|00101000101011101010100000010

Once we have found the identifier sequence we know that a valid π will follow, defining one concept that will inhibit or stimulate another concept. Because we don't know the length (number of bits) of the concepts following the identifier, this needs to become clear from the bit pattern.

One way to do this is to define the length of the concept by the number of zeros at the beginning of the bit pattern. In practise it works as follows. When we start reading the gene bit string after the identifier, we count the number of following up zeros. As long as the length of the zero bit pattern is less than the Z-Pattern the length is defined by the Depth, where the Z-Pattern and Depth are arbitrary initialized at 2 and 4 respectively. When

the zero bit pattern equals the Z-Pattern, both the Z-Pattern and Depth are doubled. The concept length (Depth) is thereby defined with the following rules:

Z-Pattern = 2
Depth = 4
IF number of zeros < Z-Pattern THEN Depth
ELSE Z-Pattern = 2 * Z-Pattern AND Depth = 2 * Depth

Using these rules will result in what is shown by table 4.5, the Concept lengths for some of the Zero bit pattern lengths.

Table 4.5: Zero bit pattern length and Concept length relation

Zero bit pattern length	Concept length
0-1	4
2-3	8
4-7	16
8-15	32
16-31	64
...	...

The previous rules are not the one and only way to implement this flexible behaviour. Both the initialization of the Z-Pattern and Depth is arbitrary and could have been any positive value as long as the Depth > Z-Pattern. Also doubling the Z-Pattern and Depth is not the only possibility to increase the limits.

But in any case flexible coding in this way means that not all the possible bit combinations are used. For instance out of the 16 possible states with 4 bits, only 12 states are valid as shown in figure 4.8.

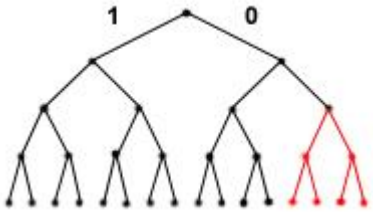

Figure 4.8: Bit codes length 4

The states '0011', '0010', '0001' and '0000' aren't valid as their zero bit pattern defines a different length.

What goes for the length of 4 bits goes for the other lengths as well. The next figure gives an illustration of the part of the bit pattern tree that will be used.

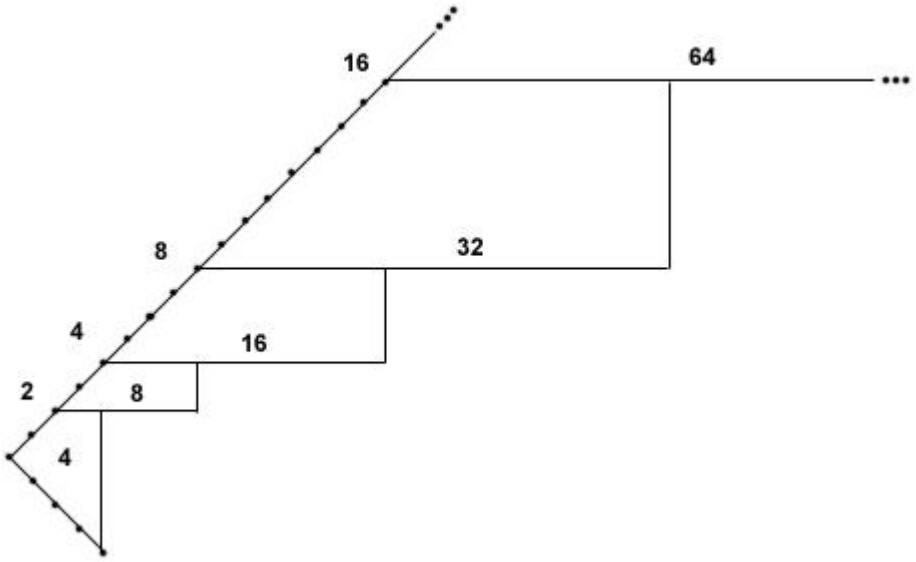

Figure 4.9: Flexible bit coding

Using only these parts of the tree results in the following number of possible valid states presented in table 4.6.

Table 4.6: Number of States

Number of bits	Number of states
4	12
8	48
16	3.840
32	16.711.680
64	$> 10^{15}$
...	...

Looking back at the gene bit string example we should now be able to read out the π's. After the '00000' identifier we remain with the next sequence of bits.

00101000101011101010100000 10

Before we find the first 1 in the string, we already count 2 zeros which define a length of 8 bits (See table 4.5). So the first concept bit code will be '00101000'. The next bit defines the relation, which is in this case a 1 for stimulating. The final step is to read out the second concept and because its head is only one zero we know it will have length 4 given us a sequence of '0101'.

00101000|**1**|**0101**|110101010000010 $\longrightarrow$ 00101000 stimulates 0101

In the remaining string of bits we can find another identifier. When we start reading after this identifier we run out of bits.

11010101|**00000**|10...

In this case we break off and consider the end of the gene to be junk. For the next gene we start all over again by searching for an identifier.

As we now fully understand how to design and code the genes we can move on to find out how to manage the process of breeding new genes.

4.3 Breeding

In order to create new and potentially better evaluation functions I want to use the Δprogram genes to breed new genes, meaning new Δprograms, new cocktail evaluation functions. To do so I need to start from a population of Δprograms mapped onto genes. The evolution of the Δprograms will then happen in generations, where in each generation we want to take the genes from the two 'best' Δprograms to create new offspring for the next population. For breeding we always need the genes from two different Δprograms (parents) and create two new Δprograms (children). To select the optimal Δprogram parents we need to evaluate Δprograms and assign to each of them a Probability representing its fitness. This part will be handled by the Belief System discussed later. First let us have a closer look at the mechanism for manipulating the genes to generate new offspring.

The breeding of new genes involves two techniques inspired by evolutionary biology: Crossover and Mutation. In addition to these techniques I suppose a crossover variation, gene breaking and disappearing as extra means to control the number of genes and their length.

4.3.1 Crossover

After having picked two evaluation functions for breeding we take their genes and randomly select a number of crossovers. This is limited by the evaluation function with the smallest amount of genes, because a gene can only function as a parent once during each breeding cycle. So we can't have more crossovers than there are genes.

For each crossover we randomly take a gene from each parent function, like for instance in figure 4.10, where we take gene 1 from the first function and gene 3 from the second function. The parent selection for Crossover is not restricted to parents with the same gene number. There are two reasons for this:

- First because there is no underlying gene structure. The location of the genes are insignificant.
- Secondly because there is not a fixed number of genes, meaning that otherwise for functions with relatively a lot of genes some of the (high numbered) genes could never be selected for Crossovers.

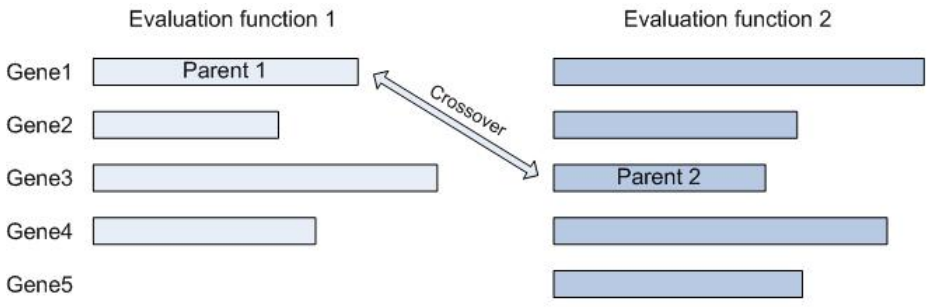

Figure 4.10: Evaluation function Genes

What we do with crossover is splitting each parent gene at a random point and swap strings beyond that point as illustrated in the next figure.

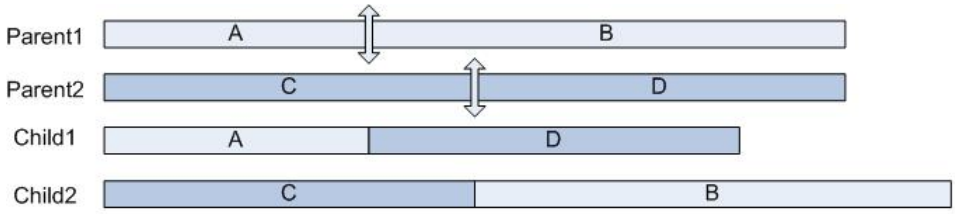

Figure 4.11: Crossover

We end up with two children, one for each new evaluation function. After a random number of crossovers we end up with two new evaluation functions based on the unaffected genes and the used parent genes are replaced by their children. Figure 4.12 gives an example of a potential result after one crossover.

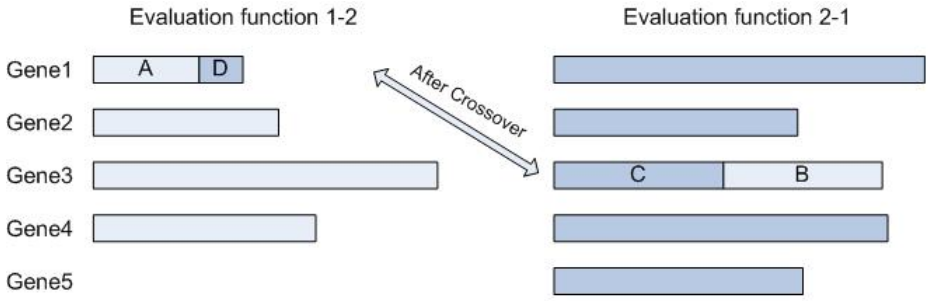

Figure 4.12: After Crossover

To provide even more variation both the children of a crossover action may end up in the same new evaluation function. This asymmetric type of crossover may occur occasionally based on a set Probability. An example of an asymmetric crossover result is shown in figure 4.13, where both children are placed in evaluation function 1.

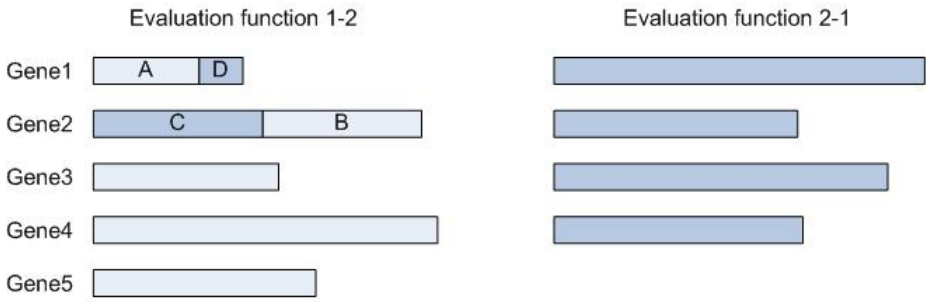

Figure 4.13: Asymmetric Crossover variation

4.3.2 Mutation

A mutation involves a probability that an arbitrary bit in the gene bit sequence will be changed from its original state. A 0 becomes a 1 and a 1 turns into a 0. Figure 4.14 shows an example of two mutations within one gene bit sequence.

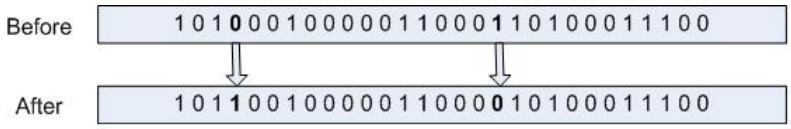

Figure 4.14: Mutation

The probability to which mutation occurs is usually set to 0.01.

4.3.3 Breaking and Disappearing

One way to control the length of the genes is to introduce gene breaking. Breaking involves a probability that a gene will split at a random point into two genes as shown in figure 4.15. The chance whether a gene should break or not is based on the length of the gene.

$$(1 - P(break))^{length}$$

The longer the gene the bigger the chance that a break will occur. $P(break)$ is usually set very small to allow long genes.

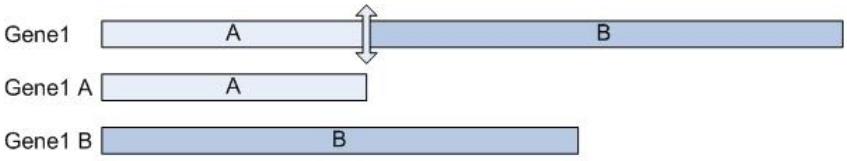

Figure 4.15: Breaking genes

Breaking genes increases the number of genes, but if we want to control the number of genes we also need a mechanism for decreasing the number as well. One way is to reverse breaking and combine genes, another way is to make genes disappear. Disappearing remains to be done with care, where if you would remove to much you might end up with an empty gene pool. Once all genes have disappeared we can't breed anymore.

4.3.4 Valid networks

It is important to notice that no matter what kind of crossover or mutation we always end up with genes that represent a valid Δprogram. This is truly the power of Δlogic in combination with flexible bit coding. Each genetically generated evaluation function is valid and now only needs to be tested and evaluated, in order for us to pick the optimal evaluation functions. For this we are going to use the Belief System, which is discussed in the next section. The Belief System will figure out which evaluation functions predict the best.

4.4 Belief System Evaluation

Every initial or generated evaluation function needs to be tested to see if it is any good. This to know which evaluation functions would make the future a less surprising place for us and to select the optimal evaluation functions for the breeding process. The evaluation of the functions will be performed by the Belief System. The reason for using the Belief System was already explained in § 2.4 as the Belief System incorporates the Abductive model to reason with irrational sets (see § 2.5) and has the three system characteristics for handling irrational sets:

- The Belief System is dependent on time.
- The learning period never ends
- It shows a form of randomness.

The Belief System will be an altered version of the system presented by T. R. Addis and D. C. Gooding(see [2]). We want to model the belief profile for only one actor (the user) instead of creating a number of interacting agents. The Belief System originally is used for a simulation that represents learning as a social process of belief revision by a number of interacting agents, but I will use and modify it to fit it to an evaluation system. Besides combining it with my designed evaluation functions I introduce the additional feature of generating and using new hypotheses (evaluation functions).

The model defines a range of beliefs about cocktail evaluation functions. The beliefs are represented by a combination of an evaluation function and belief values between 0 and 1. The evaluation functions will be the system's hypotheses. The set of these hypotheses is mutually exclusive and the sum of the beliefs will be taken as 1.

The Belief profile states the belief (also called confidence) in each of a range of hypotheses (Evaluation functions), like in figure 4.16.

To evaluate the hypotheses we need to run the evaluation functions on cocktail test samples. Knowing the rating distributions for the test samples we can compare it with the results of the evaluation function. According to how well each hypothesis predicts the outcome in comparison with the other evaluation functions, we can adjust the beliefs by changing the probability of each hypothesis. The test samples will be referred to as experiments. What the Belief System will do is constantly running experiments to update the

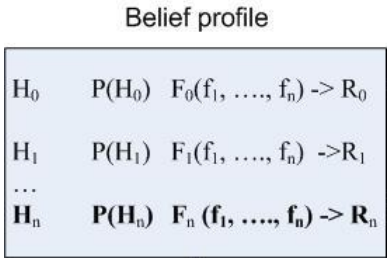

Figure 4.16: Belief profile

beliefs. In order to do so we need an experiment setup with several cocktails and a priori probabilities of each rating result. An example is given in table 4.7.

Table 4.7: Example of Experiments with their Probability of each result

Experiment	R1	R2	R3	R4	R5	R6	R7
Martini	0.0	0.1	0.9	0.0	0.0	0.0	0.0
Adonis	0.0	0.0	0.0	0.5	0.4	0.1	0.0
Bloody Mary	0.0	0.1	0.9	0.0	0.0	0.0	0.0
Manhattan	0.0	0.3	0.0	0.0	0.0	0.7	0.0

The ultimate evaluation function would correctly return the right rating distribution for each of the experiments. So what we are doing with the system is searching for a belief profile of evaluation functions which approximates the a priori experiment probabilities. In a way these experiments are what the system perceives as the real world. For the real world we can't model a person's taste once and that's it, because the user's taste may change under time and context. So the a priori probabilities my change in time, whenever the user decides that an experiment result (cocktail rating) is incorrect and adjusts it. This feedback from the user is necessary to keep track of his or her taste.

Running my Genetic Belief System means going through an infinite number of cycles of the following three steps to find the best predicting evaluation functions:

1. Select an experiment (See §4.4.4)
2. Do experiment and adjust beliefs (See §4.4.1)

3. Introduce new hypotheses (See §4.4.5)

As the system runs continuously, the user may give feedback by rating the cocktails (experiments) and we can ask the system to evaluate known and unknown cocktails (See §4.4.6).

4.4.1 Belief adjustment

Performing an experiment returns a result. The result happens according to the a priori probability distribution over the results. If we take for instance experiment example "Manhattan" from table 4.7, then we see that in 3 out of 10 cases it will probably return Rating 2 ($P(R_2) = 0.3$) and for the other 7 cases Result 6 ($P(R_6) = 0.7$). There is no way that this experiment could result in Result 1, 3, 4, 5 or 7.

Say we run the "Manhattan" experiment and it returns Rating 6. Given this result we can determine the expected hypothesis with the following equation:

$$E_{n-1}(H/R_e) = \frac{E_{n-1}(R_e/H)}{E_{n-1}(R_e)} \tag{4.1}$$

Here R_e is the result of an experiment e and H is an hypothesis. The expected result $E_{n-1}(R_e/H)$ for any experiment will depend upon the belief of each evaluation function and the probability of a result for the evaluation function supposing it is true. To get the probability of a result for an evaluation function means evaluating the experiment.

$E_{n-1}(R_e)$ is the expected result for all hypotheses. Defined by:

$$E_{n-1}(R_e) = \Sigma_H E_{n-1}(H) * E_{n-1}(R_e/H) \tag{4.2}$$

As Rating 6 resulted from our "Manhattan" experiment e then the belief probabilities (confidences)for each hypothesis, can be modified to $E_n(H)$ by adapting the above equation to the following:

$$E_n(H) = \frac{(N-1)E_{n-1}(H) + E_{n-1}(H/R_e)}{N} \tag{4.3}$$

Flexibility is defined as $\frac{1}{N}$ and reflects responsiveness to new results. The Flexibility value creates a time window for belief in previous evidence. Figure 4.17 shows how long the influence of a result is held on to previous evidence for three different Flexibility values. For example with a Flexibility of $\frac{1}{3}$ the impact of evidence after 6 updates is less than 10%.

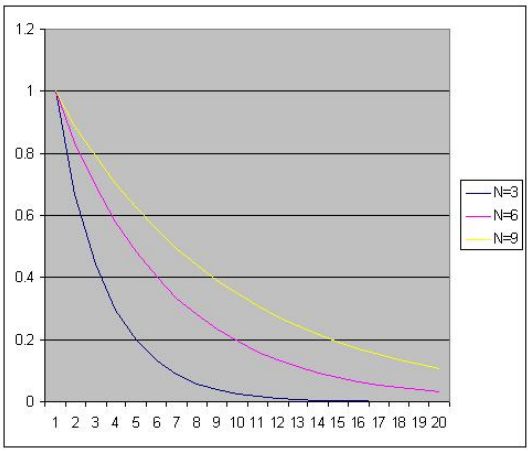

Figure 4.17: Flexibility time window

When we start to adapt the confidences for each hypothesis in response to the result Rating 6 of the 'Manhattan' experiment, we need to calculate the expected result for the experiment of each hypotheses, $E_{n-1}(R_e/H)$, and therefor we needed to see how each of the evaluation functions predicts the results for the 'Manhattan' experiment. Table 4.8 gives an example of the results from each evaluation function for the 'Manhattan' experiment.

Table 4.8: Evaluation functions with their Probability of each result

Hypothesis	Belief	R1	R2	R3	R4	R5	**R6**	R7	Sum
H_1	0.6	0.0	**0.0**	0.6	0.0	0.0	**0.4**	0.0	1.0
H_2	0.3	0.0	**0.0**	0.9	0.0	0.0	**0.1**	0.0	1.0
H_3	0.1	0.0	**0.0**	0.0	0.4	0.0	**0.0**	0.6	1.0
Sum	1.0								

Based on these evaluation function results we can now calculate the the expected result for the experiment, applying equation 4.2:

$$E(R_6) = (0.6 * 0.4) + (0.3 * 0.1) + (0.1 * 0.0) = 0.27$$

With equation 4.1 we calculate the probability for each hypotheses that is to be expected:

$$E(H_1/R_6) = \frac{0.6*0.4}{0.27} = 0.89$$

$$E(H_2/R_6) = \frac{0.3*0.1}{0.27} = 0.11$$
$$E(H_3/R_6) = \frac{0.1*0.0}{0.27} = 0.0$$

Finally we can apply equation 4.3 to adjust the beliefs. For the Flexibility we took the arbitrary value of $\frac{1}{3}$:

$$E(H_1) = \frac{2*0.6+0.89}{3} = 0.69$$
$$E(H_2) = \frac{2*0.3+0.06}{3} = 0.24$$
$$E(H_3) = \frac{2*0.1+0.0}{3} = 0.07$$

So in the end the 'Manhattan' experiment resulted in an increase of belief in hypothesis 1 ($0.6 \rightarrow 0.69$) and a decrease in both hypothesis 2 ($0.3 \rightarrow 0.24$) and 3 ($0.1 \rightarrow 0.07$).

Figure 4.18 gives a nice overview of the process of belief adjustment.

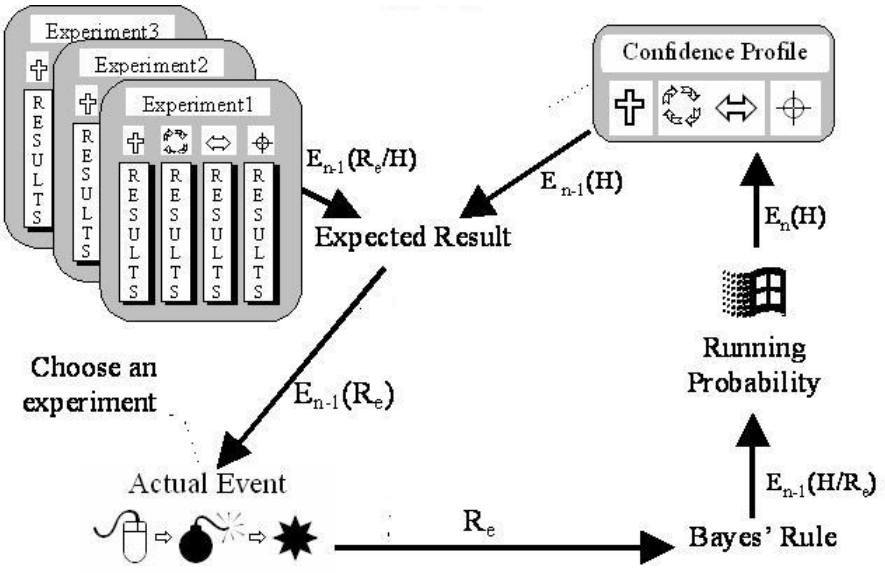

Figure 4.18: Belief adjustment

4.4.2 NULL-hypothesis

Say we would again run the 'Manhattan' experiment and this time it returns Rating 2 instead of Rating 6. If we again take a look at the evaluation function results for the 'Manhattan' experiment (see table 4.8), then we see that none of the hypotheses predicts Rating 2: $E(H_1/R_2) = E(H_2/R_2) = E(H_3/R_2) = 0.0$

Actually they are not even 0.0, in fact they are undetermined. This because each of them results in a $\frac{0}{0}$ fraction.

If we would again apply equation 4.3, with a Flexibility of $\frac{1}{3}$ and determine $\frac{0}{0}$ to be 0.0, to modify the beliefs, then we get the following:

$$E(1) = \frac{2*0.6+0.0}{3} = 0.4$$
$$E(2) = \frac{2*0.3+0.0}{3} = 0.2$$
$$E(3) = \frac{2*0.1+0.0}{3} = 0.07$$

We now have a problem that the sum of all confidences is no longer one: $\Sigma_H E(H) = 0.4 + 0.2 + 0.07 = 0.67$. To solve this problems we could normalize the confidences to 1 resulting in the same beliefs as before we started the experiment. Apparently it makes now difference whether we do the experiment or not. So why bother to do the experiment at all? This feels unsatisfied, because knowing that each hypothesis fails to predict then it makes sense to loose a bit of faith in all of them.

To overcome this problem of the Belief System I chose for another more satisfying option where I introduce what will be the NULL-hypothesis, resembling the 'it could be anything'-hypothesis. The NULL-hypothesis is an indicator for the system's ability to predict. When the NULL-hypothesis enjoys a high confidence, then we know we are heading towards a random system, where every result is equally probable. It is not always a bad thing believing a lot in the NULL-hypothesis, where in some cases the world indeed behaves at random. Sometimes there is no possible evaluation function to be found.

The NULL-hypothesis is presented in the next table in addition to table 4.8 with the results from each evaluation function for the 'Manhattan' experiment.

What is characteristic for the NULL-hypothesis evaluation function is that each probability for each result of any experiment is equal. Another point is that the belief in the NULL-hypothesis is not allowed to drop to zero. When this happens the hypothesis's confidence is reintroduced at a very small insignificant probability $\epsilon > 0$, like 0.00001 in the example.

Table 4.9: NULL Hypothesis

Hypothesis	Belief	R1	R2	R3	R4	R5	R6	R7	Sum
NULL	0.00001	0.143	0.143	0.143	0.143	0.143	0.143	0.143	1.0
H_1	0.6	0.0	0.0	0.6	0.0	0.0	**0.4**	0.0	1.0
H_2	0.3	0.0	0.0	0.9	0.0	0.0	**0.1**	0.0	1.0
H_3	0.1	0.0	0.0	0.0	0.4	0.0	**0.0**	0.6	1.0
Sum	1.00001								

The reason for this lies in the fact that while in theory beliefs will never become exactly zero, in practise we can't represent beliefs that small. And once any hypothesis's belief has reached zero, then it will always remain zero. This is an implication of using belief adjustment equation 4.3, where $E_{n-1}(H/R_e)$ will always stay zero once $E_{n-1}(H)$ is zero. In the original Belief System this was not a problem as belief could be increased through a method of consultation with other actors, but the Genetic Belief System no longer provides consultation.

If the confidence in the NULL-hypothesis would reach zero and stay zero, then we keep having the problem the NULL-hypothesis is trying to solve in the first place. We could get a decrease in belief in all hypotheses meaning that their beliefs will no longer sum up to 1. Therefor we need to keep the NULL-hypothesis in to play by reintroducing it (as an alternative to consultation). When other hypotheses then the NULL-hypothesis reach zero, then they are not reintroduced at the ϵ probability. Instead they will be replaced by a new generated hypothesis which will be explained later in § 4.4.5.

The introduction of the NULL-hypothesis seems like too much of extra work just to feel more satisfied and so there is even more to it. The NULL-hypothesis has some other additional features which will come in handy for gene initialization and breeding, which will be discussed later § 4.5.2.

4.4.3 Indifference threshold

Having a set of hypotheses we want the system to have a general confidence in his evaluation functions and its ability to predict. Entropy is an expected measure of the log of a certainty. We use this as a general measure calculated by the following equation.

$$Entropy_n = -a \sum_H E_n(H) * Log2(E_n(H)) \tag{4.4}$$

From this we can obtain an inverse of the entropy which gives an expected value for $E_n(H)$. This is denoted by $I_n(H)$ and is called the Indifference Threshold.

$$IndifferenceThreshold = \log_2^{-1}(Entropy_n) = I_n(H) \tag{4.5}$$

The indifference threshold sets the confidence level. Only hypotheses with a higher probability than the confidence level are believed or in other words under consideration. The higher the confidence level, the higher the confidence of the system that it is able to predict correctly.

4.4.4 Selecting Experiments

Having stated the confidence level of the system, the system tries to increase this level to gain greater overall confidence. So instead of selecting experiments randomly, we want to select the experiment that would make the future a less surprising place one that would increase the overall confidence. To increase the overall confidence we need an experiment that discriminates between hypotheses. So the choice of experiment is derived from its effectiveness in discriminating between hypotheses and its stability of a result given a hypothesis. To make the choice we use an Entropy measure for each experiment to describe the confidence of an hypothesis, the expected certainty of a result given a hypothesis:

$$Entropy_e(Experiment_e) = \sum_R \sum_H E_{n-1}(H/R_e) * Log2(E_{n-1}(H/R_e)) \tag{4.6}$$

The experiment with the minimum entropy will give the most decisive results for supporting or turning down each of the hypotheses in the Belief system. As the experiment with the minimum entropy seems like the best option to take, it will not always be chosen.

We will use a mixed strategy approach suggested by Game Theory to select an experiment. The experiment's entropy represents the expected probability of the experiment, so for all experiments we can set up a probability distribution on which we can base our decision. The decision mechanism deploys expected confidence as a probability to select.

Because we are dealing with the case that a hypothesis is either True or False we can take pairs of expected losses (say E_n, E_{n+1}) that are equal (with experiments E_n, E_{n+1} with indifference values of P_n, P_{n+1}):

$$(1 - P_1) * f_1 = (1 - P_2) * f_2$$
$$(1 - P_1) * f_1 - (1 - P_2) * f_2 = 0$$

In matrix form for solving equations we have:

Table 4.10: Four equations for four unknowns

	f1 for E1	f2 for E2	f3 for E3	f4 for E4	
Sum of fn = 1	1	1	1	1	-1
E1 - E2	(1-P1)	-(1-P2)	0	0	0
E2 - E3	0	(1-P2)	-(1-P3)	0	0
E3 - E4	0	0	(1-P3)	-(1-P4)	0

The payoff should invoke a maximum security level from the entropy pairings. In this way the experiment can be selected which will gain greater overall confidence.

The big advantage of using the approach suggested by Game Theory is that we can really cut down on the number of experiments we need to run. We can pick experiments that really make a difference.

4.4.5 Generation of new Hypotheses

Sofar we have seen how to select and do experiments to update the beliefs in the evaluation functions. With running the experiments we can find one or more favorable hypotheses in the belief profile, where the belief profile is setup a priori with a fixed number of hypotheses (See §4.5). As these hypotheses start of with a certain approximation of the result probabilities of the experiments we want to improve the approximation by introducing new hypotheses which are potentially better in predicting each experiment's outcome.

This Chapter already provided us with the genetic mechanism for hypothesis generation. We now only have to decide on the following:

1. With which two hypotheses do we want to breed?
2. Which hypotheses do we want to replace?

3. When should we start breeding?

(1)So first we need to select two hypotheses, which need to be two different ones. The selection is based on the belief probability, where the highest belief has the biggest chance of being chosen. The reason for not always picking the ones with the highest belief is that it not guarantees to get the best offspring.

(2) Next because we have a fixed number of hypotheses we need to select two of them to be replaced by two children, the offspring of the two selected hypotheses for breeding.

- First of all the hypotheses to be replaced ought to be different then the ones already used as parents.
- Secondly the NULL-hypothesis can never be replaced.
- Thirdly if there are any two hypotheses with zero confidence they will be selected immediately as they will always remain zero (See §4.4.2). Otherwise the selection is again based on the belief probability $P(H)$, but only with the difference that we now want the lowest belief to have the biggest chance of being chosen. To do so we first modify the probability $P(H)$ to $(1 - P(H))$.

To put the new hypothesis under consideration we introduce them both at the indifference level. Because this means the beliefs will no longer sum to 1 anymore (except for the case where all the hypotheses were already at the indifference level), we need to normalize.

(3) Every time we perform an experiment we will check afterwards to see if there are any two hypothesis with zero confidence. If so we generate two new ones.

Every x-number of cycles (say 50) we will perform a generation any way. After the introduction of new hypotheses we need a couple of cycles for the belief system to settle down again.

4.4.6 Cocktail Evaluation

Because I introduce a system for the evaluation of cocktails I need to explain how to use the system to evaluate a cocktail. The evaluation of known and unknown cocktail combinations is based on all evaluation functions. Each hypothesis (evaluation function) evaluates the ingredients of the cocktail and based on all their predictions we calculate the cocktail rating distribution:

$$E(Cocktail_c) = \sum_R \sum_H E(R_c/H) \tag{4.7}$$

Take for example the possible evaluation results of each hypothesis for a particular cocktail presented in Table 4.11.

Table 4.11: Evaluating cocktail

Hypothesis	Belief	R1	R2	R3	R4	R5	R6	R7	Sum
NULL	0.05	0.143	0.143	0.143	0.143	0.143	0.143	0.143	1.0
1	0.4	0.0	0.0	0.6	0.0	0.0	0.4	0.0	1.0
2	0.3	0.0	0.0	0.9	0.0	0.0	0.1	0.0	1.0
3	0.25	0.0	0.0	0.0	0.4	0.0	0.0	0.6	1.0
Sum	1.0								

For each result of each hypothesis we calculate $E(R_c/H)$, meaning that each result probability of each hypothesis is multiplied by its belief in it (for example $E(R_1/NULL) = 0.05 * 0.007$). This results in the following table:

Table 4.12: Evaluating cocktail

Hypothesis	Belief	R1	R2	R3	R4	R5	R6	R7	Sum
NULL	0.05	0.007	0.007	0.007	0.007	0.007	0.007	0.007	1.0
1	0.4	0.0	0.0	0.24	0.0	0.0	0.16	0.0	1.0
2	0.3	0.0	0.0	0.27	0.0	0.0	0.03	0.0	1.0
3	0.25	0.0	0.0	0.0	0.1	0.0	0.0	0.15	1.0
Sum	1.0	0.007	0.007	0.517	0.107	0.007	0.197	0.157	1.0

As shown in the table, adding up $E(R_c/H)$ for each result of each hypothesis results in the following probability distribution:

$$(0.007 \mid 0.007 \mid 0.517 \mid 0.107 \mid 0.007 \mid 0.197 \mid 0.157)$$

4.4.7 Performance

Now we know how to evaluate cocktails we can evaluate the experiments to find out how well the system performs, by comparing the a priori experiment probability distributions with the evaluated probability distributions. In this way we have another measure next to the confidence level, where the confidence level only states how firm the system beliefs in its hypotheses, but shows no indication of how well it predicts cocktail ratings. This was not a feature of the original Belief System.

To calculate the distance between two probability distributions, let us consider the following. When P and Q represent two probability distributions, and p_R and q_R represent the places (Ratings) in those distributions respectively, and D represents the distance between the two probability distributions, then D can be calculated with the Euclidian distance:

$$D = \sqrt{\sum_{R}(p_R - q_R)^2)} \tag{4.8}$$

We can now calculate the euclidian distance for each experiment probability distribution and its predicted probability distribution. Taking the mean of all these distances returns the system performance as a value between 0 and $\sqrt{2}$:

$$Performance = \frac{\sum_e D_e}{e} \tag{4.9}$$

Performance is the euclidian distance between the predicted results and the optimal results set by the a priori experiments.

4.5 Initialization

To initialize the Genetic Belief System I had to think of how to create the concepts (bit codes), set up experiments and initialize the hypotheses.

4.5.1 Concepts

For the system we need to create the initial input, output and possible intermediate concepts. For the output concepts, the number of output states needs to be selected and a concept is added for each output. The input concepts are created by the user as he or she defines a list of ingredients representing the input concepts. Besides the input and output concepts the user may also add an arbitrary number of intermediate concepts. The reason for letting the user construct the input, output and intermediate concepts by him or herself was to give the user total freedom. The advanced user could now create any Δnetwork possible for the initial creation of the hypotheses (see § 4.5.2).

The concept bit codes are generated by the system depending on the number of concepts. All the initial concepts (input, output and intermediates) will start with bit codes of equal lengths. In this way each concept has the same probabilities of emerging and disappearing during breeding and on the effect of crossover and mutation.

With the defined ingredient input concepts the user can construct cocktail combinations. In this way each cocktail represents a list of input concepts. These cocktails are used to set up experiments. For this the user only needs to give an a priori rating to each of the cocktails. With the experiments set up it is time to initialize the hypotheses with which we try to approximate the rating distributions of the experiments.

4.5.2 Hypotheses

For the Genetic Belief System we need to create the initial hypotheses. First we select the number of hypotheses for the belief profile. The number of hypotheses is fixed and we can only replace hypotheses. The reason for having a fixed number is that we would otherwise run into computational problems, we just can't have an unlimited number of hypotheses due to the belief calculations. One other option is to start with a small number of hypotheses and increase the number of hypotheses to a certain limit, but we like to start off with the maximum number of hypotheses to have more

variation.

There are two options for creating the fixed number of initial hypotheses, either the user creates them manually or we let the system create random hypotheses.

The first option gives the user the opportunity to built in some of his a priori knowledge about the ratings as he or she can construct any possible Δnetwork. In this way the user can bias the system and perhaps give it a head start in searching for the hypotheses giving the 'optimal' approximation. The only problem is that we cannot say that the bias is always a good thing. Further more it could also be a lot of work defining multiple complex Δnetworks. The system should eventually come with a solution out of a completely randomly created genes as well. So if the initial time is not an issue, then we just let the system run a bit longer.

Instead of creating hypotheses with completely random genes of random bit sequences we suggest to create the initial hypotheses out of the NULL-hypothesis, where we take the NULL-hypothesis to be a fully connected network. There are several possible fully connected NULL-hypotheses, take for example the two in figure 4.19.

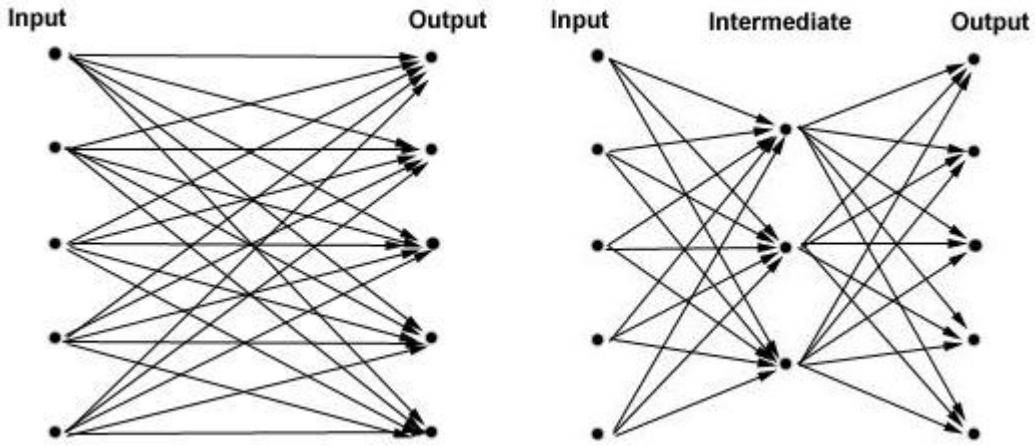

Figure 4.19: Fully connected NULL-hypotheses

That a fully connected network indeed functions as the NULL-hypothesis is because for each combination of inputs it activates each output equally. This results in an evenly distributed output saying that each output is equally probable.

An important feature of the fully connected network of the NULL-

hypothesis is that it preserves the input and output concepts. During breeding also the NULL-hypothesis genes can be used and this could help us reintroduce the input and output concepts whenever they are lost in the other hypotheses's genes. Of course they could also emerge from junk by mutation and crossover, but this could take time.

The fully connected network is create out of a list of stimulator patterns. To create other hypotheses we take a random number of the stimulator patterns and create a new network in this way, which results in a part of the NULL-hypothesis network. An example is given in figure 4.20 where we took the NULL-hypothesis from figure 4.19 without the intermediates to select from.

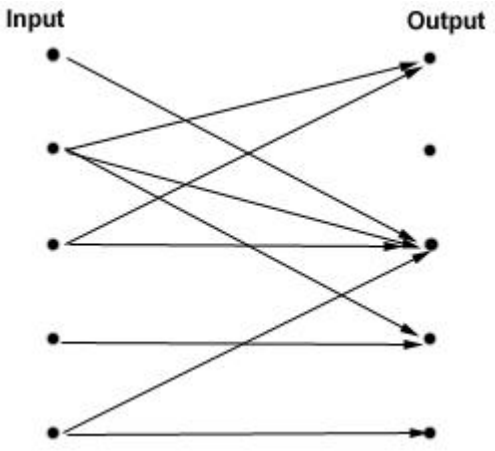

Figure 4.20: Part of the NULL-hypothesis

The advantage of using parts of the NULL-hypothesis would be that we are sure of starting of with a number of stimulator patterns and input concepts, instead of being completely random.

With the list of stimulator patterns and the network they represent we still need to create genes as we want bit string sequences containing the stimulator patterns and junk. For this we need to define the following parameters:

- The pattern identifier length
- The percentage of junk
- Insert probability; P(insert)

- Break probability; P(break)

Based on the parameters we will create one big string in which we insert the stimulator patterns, junk and bits with the letter 'b'. The string will be broken on the places assigned with the letter 'b', resulting in multiple bit sequences, genes. How this is done exactly is discussed next.

First we calculate the number of bits which will be signed as junk. The calculation is based on the number of bits required to represent the stimulator patterns and the percentage of junk. Say we have for example only the following three stimulator patterns with their bit code sequence and we want 97% of junk:

Coke stimulates Result1 → **0101|1|1000**
Coke stimulates Result2 → **0101|1|0100**
Rum stimulates Result2 → **0111|1|0100**

We have $3 * 9 = 27$ bits representing the stimulator patterns. 97% of junk means that the 27 bits are only 3%, so we need $\frac{27*0.97}{0.03} = 873$ bits of junk.

The next step is to start a loop wherein we build a sequence of bits which includes the 873 bits of junk and the 3 stimulator patterns preceded by an identifier. The loop is repeat for the number of junk bits and every step we add either a junk bit or one of the stimulator patterns based on P(insert). Each stimulator pattern can only be added once. Based on P(break) we may add 'b' as well:

```
For(int i = 0; i<bits of junk; ++i){

if(P(break)> random value ) insert 'b';

if(P(insert)> random value & Number of patterns > 0 ) {
insert stimulator pattern;
Number of patterns-1;
}
else insert random junk value;
}
```

When the loop is finished and we still have remaining stimulator patterns, then they are added at the end of the sequence. Afterwards we split the sequence at the 'b'-points and we have our genes.

4.5.3 System parameters

When the experiments are set and the hypotheses are initialized, then before we can start running the Genetic Belief System, we first need to set the following parameters for the breeding process and belief adjustment:

- Breeding probability; P(breed). The probability that the system starts breeding with two hypotheses during a cycle. (default=0.02)
- Mutation probability; P(mutation). The probability to which mutation occurs. (default=0.01)
- Flexibility. The responsiveness to new results.(default=0.30)
- Asymmetric breeding probability; P(asym). The probability to which asymmetric crossover occurs. (default=0.2)
- Gene break probability; P(gene break). The chance whether a gene should break or not. (default=0.0)
- Gene disappear probability; P(disappear). The chance whether a gene should disappear or not. (default=0.00)

Now with all that is needed initialized, the system model is ready to run. In order to run and test our model we implemented an application based on the model, which is presented in the next chapter.

Chapter 5

Implementation

The Genetic model presented in the previous Chapter is embedded in an application providing an easy tool with which we could create the ingredients, cocktails, experiments and the hypotheses needed to run and test the irrational adaptive system. The Belief System by is modified and extended with the additional feature of hypotheses generation which will be performed by a Dynamic Link Library or DLL (see § 5.2.2).

Because the tool used for implementing the Belief System, Clarity[1](see appendix B), doesn't provide a suitable option for building a graphical user interface (GUI), we also needed to build a C++ MFC interface on top of that (see § 5.2.3).

Section 5.1 provides the systems functionality from a users perspective. Section 5.2 presents the system design describing the structure of the system.

Finally in section 5.3 we show the flow chart of the process of running the Genetic Belief System.

[1]Clarity is developed by T.R. Addis, J.J. Townsend Addis and D.C. Gooding at the University of Portsmouth and the University of Bath

5.1 UML

UML stands for Unified Modeling Language and is a system of diagrams that can specify how systems work. The systems functionality from a users perspective is specified in the next functional model by a use-case-diagram.

Use cases focus on an external view of the system. A use case describes a function provided by the system that yields visible results for an actor. An actor describes any entity that interacts with the system (e.g. a user, another system, the systems physical environment) (See [3]). In the use-case-diagram in figure 5.1 the use case for the system user is shown. The system provides all the functionality to setup and initialize a test case for the Genetic model. Therefor the user is able to:

- Add ingredients by specifying a name and type.
- Add Cocktails as combinations of the added ingredients.
- Besides the input concepts represented by the ingredients, the user can add intermediate and output concepts, which are used in constructing the ΔNetworks for the evaluation functions.
- Add experiments based on the cocktails. The user can select which cocktails are going to function as experiments and are given an initial rating distribution. These ratings are the a priori experiment results.
- In order to run the Genetic Belief System the user needs to generate the initial hypotheses. Several parameters can be set to influence the hypothesis generation (see § 4.5.2).
- After the hypothesis generation the user may initialize the Genetic Belief System with another set of parameters (see § 4.5.3)
- After the initialization of the Genetic Belief System the user can start (and stop) running the system in order to find a belief profile which predicts a right approximation of the a priori experiment results.
- The user may check up on how well the system approximates the a priori experiment results. The system returns a list with the predicted results for each experiment cocktail. Also the performance level is showed to the user.
- Finally the user can save hypotheses.

The System design in the next paragraph describes the structure of the system.

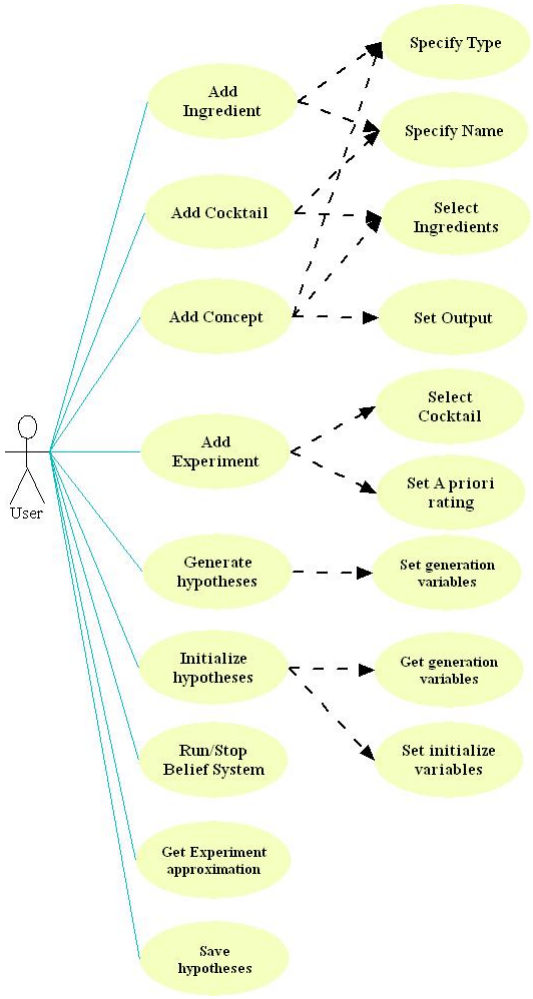

Figure 5.1: Use Case diagram

5.2 System design

The System is based on the Belief System which is written in a functional language called 'Faith' which is generated by and designed in Clarity (see appendix B). I modified and extended it with the additional feature of hypotheses generation which is performed by my designed Dynamic Link Library (DLL), which I named GIBS (Genetic Irrational Belief System). The DLL performs the generation, evaluation and breeding of genes. On top of the Belief System I designed a Graphical User Interface in C++ to control the Genetic Belief System by providing all the functionalities presented in the previous UML Use Case Diagram (figure 5.1).

So the system basically consists of three main components:

- The Clarity Belief System (see § 5.2.1)
- GIBS DLL (see § 5.2.2)
- C++ MFC Interface (see § 5.2.3)

Figure 5.2 illustrates how these three parts are related to one another.

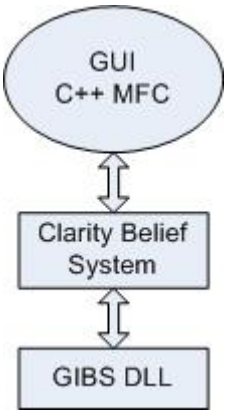

Figure 5.2: System

The same system overview is presented in figure 5.3 on the next page, only now showing the components in more detail.

In the next three subsections each main component is described in further detail.

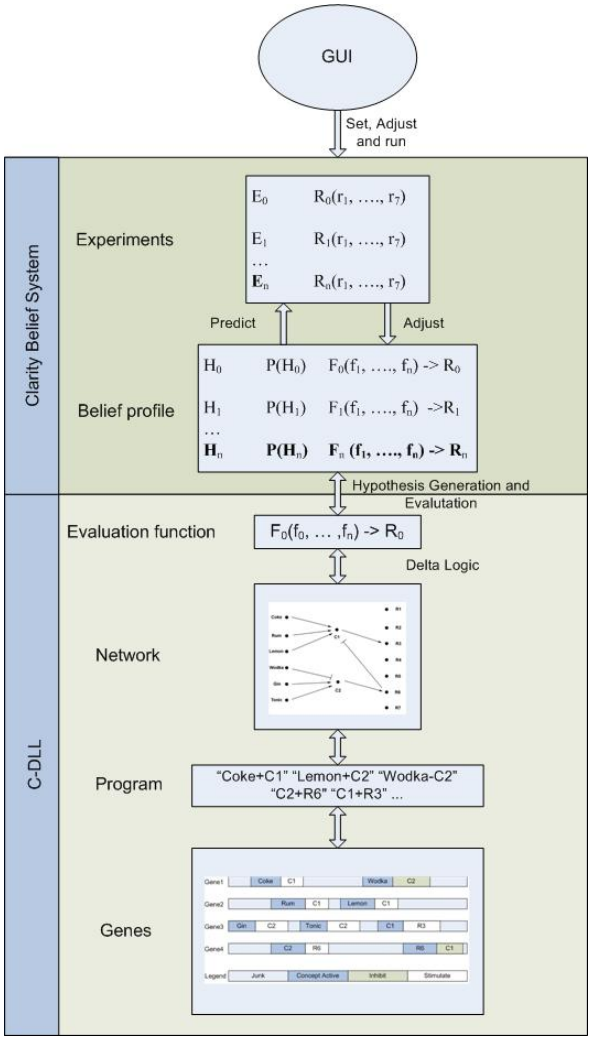

Figure 5.3: System overview

5.2.1 Clarity Belief System

The Belief System can be described by the following two abstract components, which are also illustrated in Figure 5.4:

- The Belief profile, which states the belief in each of the hypotheses. (Evaluation functions).
- The Experiments, which are test samples with an a priori rating distribution.

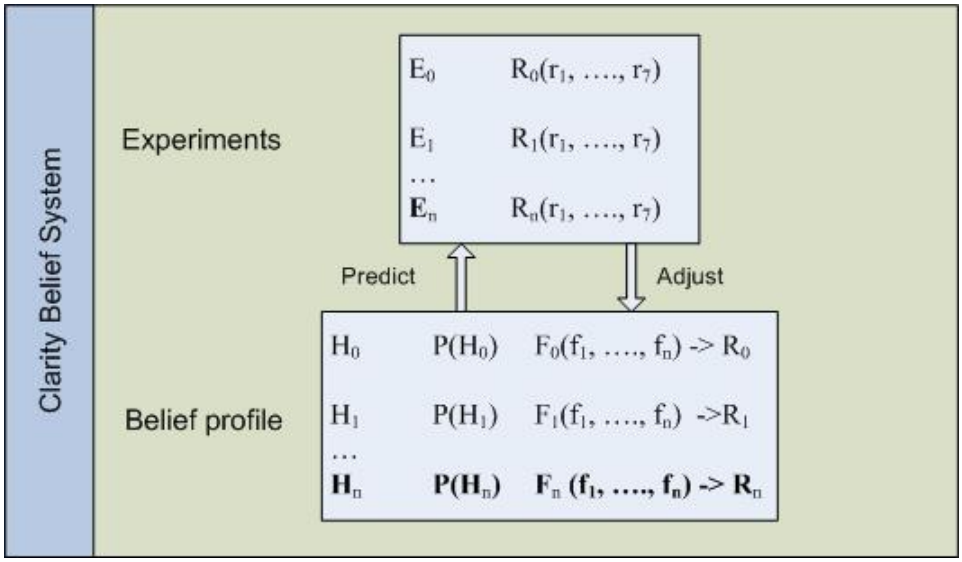

Figure 5.4: Belief System components

The system is searching for a belief profile of evaluation functions which approximates the a priori experiment result probabilities. The evaluation functions are predicting the results.

Every time when the system performs an experiment the beliefs in the Belief profile are adjusted according to experiment results.

5.2.2 GIBS-Dll

I implemented a Dynamic Link Library (DLL) called GIBS to perform all the gene manipulation in the system. The DLL is used for:

- the breeding of new genes.
- the evaluation of genes.
- the creation of new genes.

The GIBS-DLL is implemented in C and contains a number of functions which can be called from within Clarity. In order to make this work the created DLL is based on the following C file template:

```
#include <windows.h>
#define EXPORT __declspec(dllexport)
EXPORT char* CALLBACK function_name(char* values[],int count);

int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason,
PVOID pvReserved)
{
return TRUE;
}

EXPORT char* CALLBACK function_name(char* values[],int count)
{
function code which returns a string ...
}
```

There are five EXPORT functions in GIBS-DLL, namely:

- eval; Given an hypothesis and a list of ingredients the function returns a string with a probability distribution over the outputs.
- hypogeneration; Given two parent and two child hypotheses the function will breed new genes to replace the genes of the two child hypotheses. Returns a string stating “TRUE” or “FALSE”.
- init; Initializes the genes. Returns a string stating “TRUE” or “FALSE”.

- set_globals_breed; Sets the parameters for the breeding process. Returns a string stating “TRUE” or “FALSE”.
- set_globals_create; Sets the parameters for the gene creation process. Returns a string stating “TRUE” or “FALSE”.
- genes_creation; Creates new genes. Returns a string stating “TRUE” or “FALSE”.

Figure 5.5 gives an example of the use of the GIBS-DLL by illustrating how we can call for the function ‘eval’ in the GIBS-DLL by using the built-in function ‘user’.

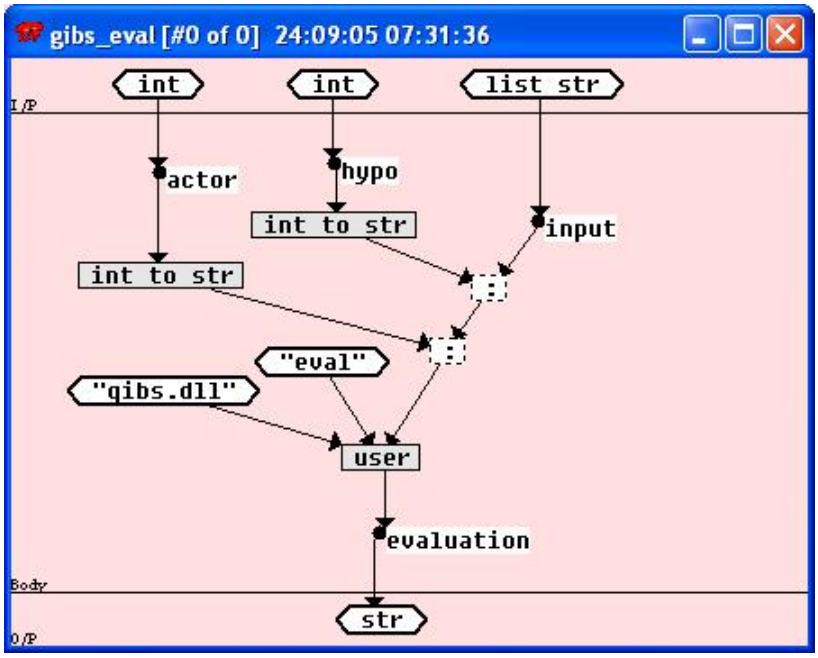

Figure 5.5: Use of DLL

To manipulate genes the DLL needs to read and write gene bit strings from and to a text file. Every hypotheses has its own text file with genes. All files use the following template:

```
Line 0: Number of genes
Line 1: gene 1 length
Line 2: gene 1 string
Line 3: gene 2 length
Line 4: gene 2 string
Line 5: gene 3 ...
etc.
```

Take for example the following small gene file:

```
3
23
10001010011010101000110
5
10010
46
1000101001101010100011010001010011010101000110
```

When the genes are initialized, the system stores the strings in memory in a char array. The functions 'eval' and 'hypogeneration' manipulate these arrays.

5.2.3 C++ MFC interface and Faith-DLL

On top of the Belief System we designed a Graphical User Interface (GUI) in C++ to control the Belief System by providing all the functionalities presented in the previous UML Use Case Diagram (Section 5.1).

The reason why we didn't builded the GUI with Clarity was because it cannot be used to create a windows (dialogs) based interface. So instead we chose for the C++ Microsoft Foundation Class Library. We picked a C++ based interface, because there was already a DLL available that could provide the communication between C++ and Clarity, called faith.dll.

The next piece of C++ code for the Dialog frame in Figure 5.6 shows an example on how to use the faith.dll. The interface allows the user to evaluate a Clarity query, returning the answer in the Reply window.

```
extern "C" __declspec(dllexport)
void ask_query(char *query, char *reply, HWND hMsge);
void CQueryDlg::OnEvaluate() {

static HWND hwnd = NULL;
char query[1000];
char reply[5000];

UpdateData(TRUE);
if(hwnd == NULL)
GetDlgItem(IDC_MESSAGES, &hwnd);

strcpy(query, (LPCTSTR)m_strQuery);
ask_query(query, reply, hwnd);

m_strReply = _T(reply);

UpdateData(FALSE);
}
```

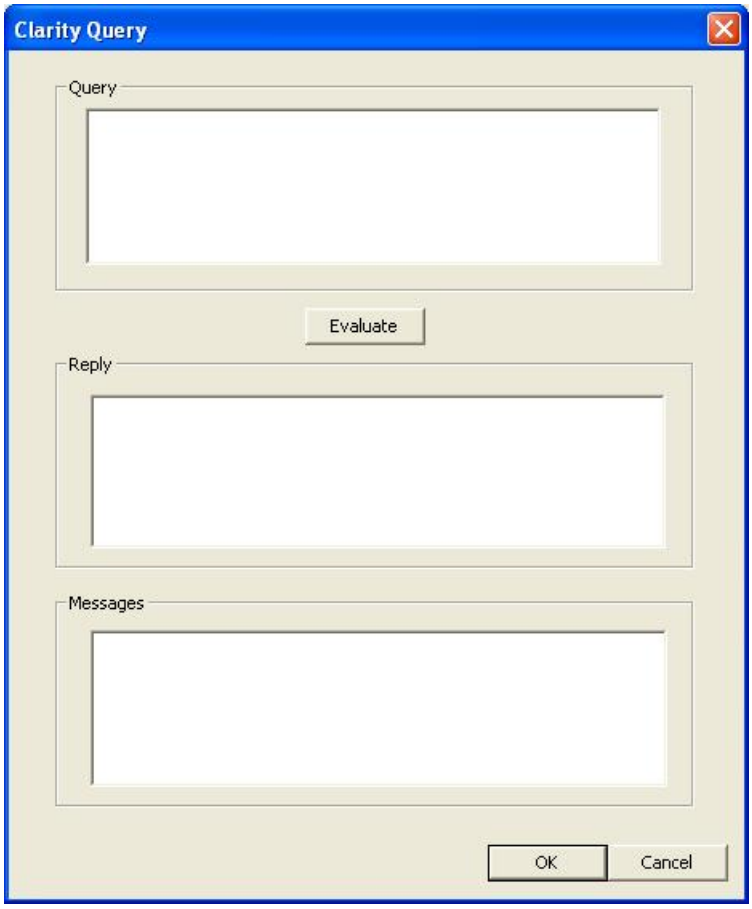

Figure 5.6: Clarity queries

The application is dialog based and the main dialog frame gives the user an overview of all the different dialogs as shown in Figure 5.7.

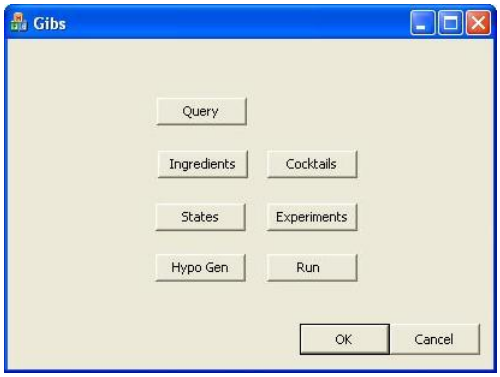

Figure 5.7: Dialog based application

The user could press for instance the 'Hypo gen' button to open the dialog shown in the following Figure, where the user can set the parameters for gene creation and generate the initial genes.

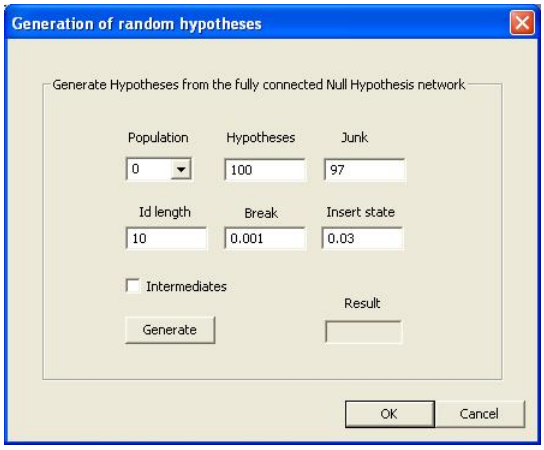

Figure 5.8: Dialog for Generating Hypotheses

5.3 Running Thread

If we want to track an irrational set we need to run the system forever. When we run the Genetic Belief System we start a thread which will keep on running until the user stops it or the application is terminated. The Thread is a looping cycle wherein it performs experiments to update the beliefs and breeds new hypotheses. The flow chart of this process is presented in the next figure.

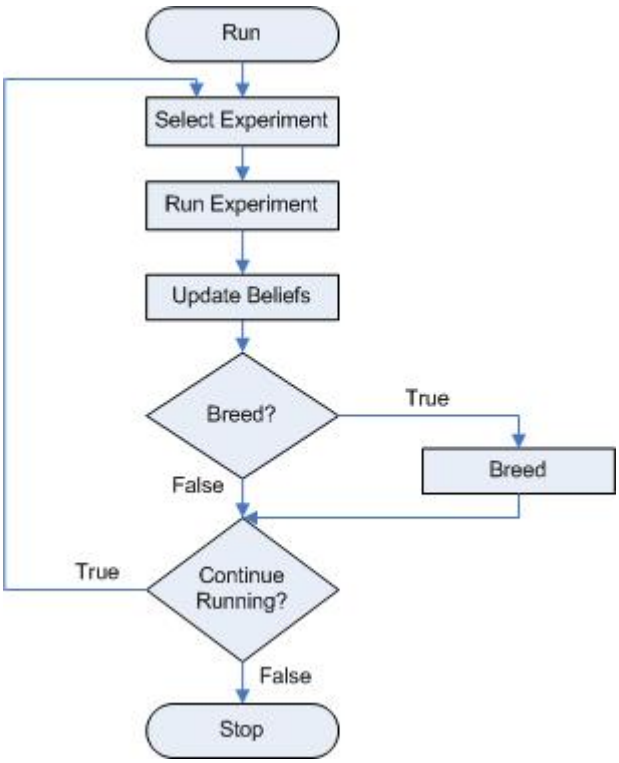

Figure 5.9: Thread

Chapter 6

System evaluation

In this chapter we will try to find out whether the system indeed behaves like an irrational system. For this we are going to test the performance of the system on two things:

1. Approximation of a priori experiment results
2. Tracking changes

The system should continuously search for potentially better predicting hypotheses. We will see if the performance of evaluation indeed increases over time, where performance is the euclidian distance between the predicted results and the optimal results set by the a priori experiments (see § 4.4.7). The shorter the distance, the better the approximation and performance.

So we will look at how well the belief profile with evaluation functions approximates the a priori experiment results. These results, which represent the users taste could change in time. So when the a priori experiment results are changed, the system should start approximating the new results. In other words it needs to track the changes of the user.

Section 6.1 presents the test case for which we are going to determine the systems performance of evaluation. Section 6.2 shows the results for the test case.

6.1 Test Case Highballs

For the test set we took nine Highball cocktails. When the Highball drink rose to prominence in the 1920s many people claimed authorship. It took a special investigation by the New York Times to establish beyond reasonable

doubt that it was created around 1895 by Mr Patrick Duffy. He used one liquor, one mixer (soda or ginger ale) and no more than one garnish, usually a twist of lemon, or none at all. Up to two mixers can be used, one of which should be sparkling, the inclusion of more than one base liquor is to be avoided (See [6]). Table 6.1 presents the nine Highballs with their ingredients.

Table 6.1: Cocktails ingredients

Cocktail	Ingredients
Brandy H	brandy, ginger ale, muddler, twist of lemon
Apple Brandy H	apple brandy, ginger ale, muddler, twist of lemon
Bourbon H	bourbon, ginger ale, muddler, twist of lemon
Gin H	gin, ginger ale, muddler, twist of lemon
Scotch H	ginger ale, muddler, scotch, twist of lemon
Carpano H	lemonade, punt e mes, twist of orange
Mile H	kirsch, midori, sparkling bitter lemon, twist of lemon
Sky H	blue curacao, pineapple juice, scotch, twist of lemon
Seville H	mandarine napoleon, pernod, sparkling bitter lemon, twist of orange

The first five Highballs are known as Brandy Highballs. According to Robert Cross are the Brandy Highballs less tasteful than the rest of the Highballs. Based on this we set up the Highball experiments with the rating results presented in table 6.2.

Table 6.2: Highball ratings

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**1.0**	0.0	0.0	0.0	0.0	0.0	0.0
Apple Brandy H	0.0	**1.0**	0.0	0.0	0.0	0.0	0.0
Bourbon H	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
Gin H	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0
Scotch H	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
Carpano H	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0
Mile H	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0
Sky H	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**
Seville H	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0

The system should approximate these a priori experiment results.

6.2 Results and Discussion

In order to run the system we need to generate initial hypotheses given the generation parameters (see § 4.5.2). The parameter values for the generation of hypotheses are set with the values from Table 6.3.

Table 6.3: Generation Parameters

Parameter	Value
Hypotheses	10
Identifier length	10
Junk percentage	97
P(insert)	0.03
P(break)	0.001

We like to start with a small number of hypotheses (10) as this is good enough to illustrate whether it works or not. The other parameters are default values, meaning we get 97% of junk and genes have on average a length of 1000 bits.

The initial genes are created out of the NULL-hypothesis without intermediate concepts. All input concepts are connected with all output concepts (see § 4.5.2).

With the hypotheses created we are ready to initialize the Belief System (see § 4.5.3). The parameter values used for initialization are shown in Table 6.4.

Table 6.4: Initialization Parameters

Parameter	Value
P(breed)	0.02
P(mutation)	0.01
Flexibility	0.30
P(asym)	0.0
P(gene break)	0.0
P(disappear)	0.0

These are all the default values, only the asymmetric breeding is switched off. First we like to get the picture of the simple case.

When we run the genetic Belief system for 30000 cycles, we get the performance level presented in Figure 6.1.

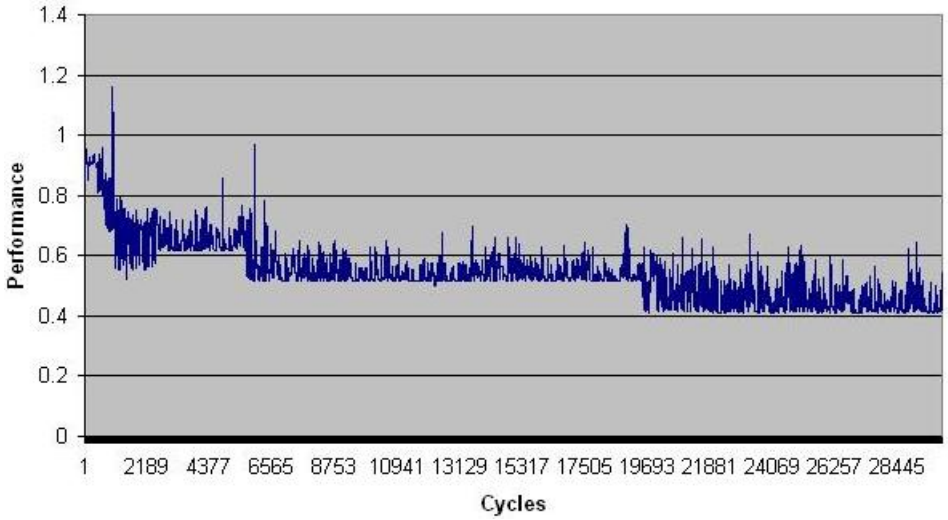

Figure 6.1: Results 30000 cycles and 10 hypotheses

The result is promising as the graph shows just the kind of behavior we would expect, namely a stepwise increase in performance. When a hypothesis is generated and as always introduced with a belief at the indifference level (meaning a relatively strong belief, see § 4.4.3, then there are three possibilities:

- The hypothesis predicts far more worse than the other ones, which results in a decrease in performance, a peak in the graph (increase in Euclidian distance).
- The hypothesis predicts on average the same as the others. Then there won't be a significant change.
- The hypothesis predicts a lot better than the other ones, suddenly it can approximate the result of an experiment closely for which the previous hypotheses didn't have a clue. At these points in the graph we will see a relatively large decrease in euclidian distance, meaning an increase in performance.

We would expect to see a lot of hypothesis creations which predict worse due to the random factor in breeding. Picking two good hypotheses to breed with is no guarantee for generating good hypotheses. It's simply waiting for the moment to get lucky.

Table 6.5 gives an overview of the cocktail predictions after the 30000 cycles. The **bold** values are the ones that should in the ideal situation be 1 (see Table 6.2).

Table 6.5: Estimation, 30000 cycles and 10 hypotheses

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**0.822**	0.018	0.018	0.018	0.018	0.018	0.086
Apple Brandy H	0.143	**0.143**	0.143	0.143	0.143	0.143	0.143
Bourbon H	0.143	0.143	**0.143**	0.143	0.143	0.143	0.143
Gin H	0.016	0.016	0.016	**0.904**	0.016	0.016	0.016
Scotch H	0.143	0.143	**0.143**	0.143	0.143	0.143	0.143
Carpano H	0.143	0.143	0.143	0.143	**0.143**	0.143	0.143
Mile H	0.043	0.043	0.043	0.043	0.043	**0.743**	0.043
Sky H	0.022	0.022	0.022	0.022	0.022	0.022	**0.775**
Seville H	0.043	0.043	0.043	0.043	0.043	**0.743**	0.043

In the next run we put in a bit more variation during breeding, by setting the probability for asymmetric crossover to 0.2. The performance level during 15000 cycles is shown in Figure 6.2.

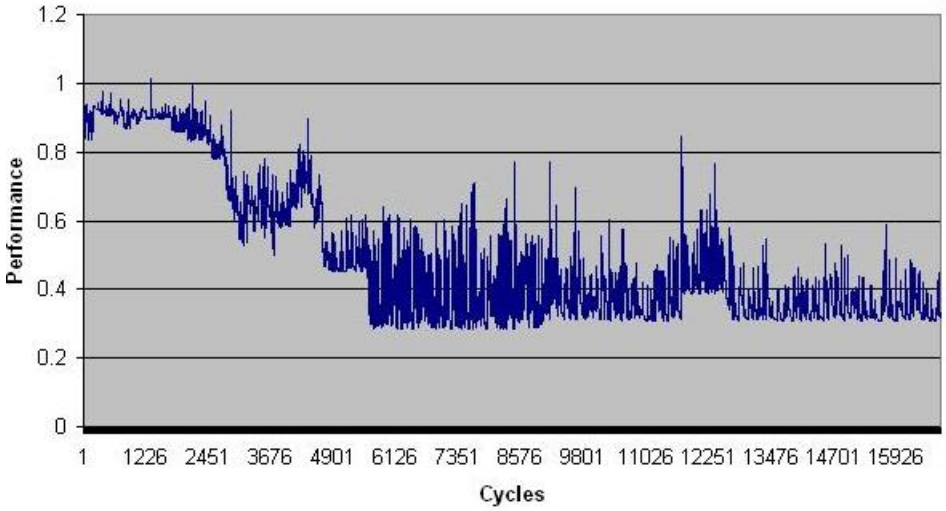

Figure 6.2: Results 15000 cycles, 10 hypotheses, asymmetric crossover

The graph shows the same kind of behavior as before, only now we see that we get to a higher performance much quicker and at the end of the

15000 cycles we already have a better predicting system (see Table 6.6)

An interesting point in the graph is around cycle 12000, here we notice a significant decrease in performance (increase in the graph). We seem to have lost better predicting hypotheses and got worse ones in return. This problem could occur because every hypothesis has a chance of being replaced during a breeding cycle, so also the best hypothesis with a relatively small chance. If we increase the number of hypotheses, then this problem would occur less often. We also preserve more variation with more hypotheses.

Table 6.6: Estimation 15000 cycles, 10 hypotheses, asymmetric crossover

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**0.140**	0.140	0.140	0.140	0.140	0.140	0.160
Apple Brandy H	0.000	**0.862**	0.000	0.000	0.000	0.000	0.138
Bourbon H	0.140	0.140	**0.140**	0.140	0.140	0.140	0.160
Gin H	0.027	0.139	0.027	**0.708**	0.027	0.027	0.047
Scotch H	0.132	0.132	**0.132**	0.132	0.132	0.132	0.211
Carpano H	0.000	0.000	0.000	0.000	**0.929**	0.000	0.071
Mile H	0.032	0.032	0.032	0.032	0.032	**0.601**	0.238
Sky H	0.002	0.002	0.002	0.002	0.002	0.002	**0.986**
Seville H	0.026	0.026	0.026	0.026	0.026	**0.661**	0.211

Next we continue running the belief system with twice as much hypotheses, namely 20. Figure 6.3 shows the level of performance over 65000 cycles.

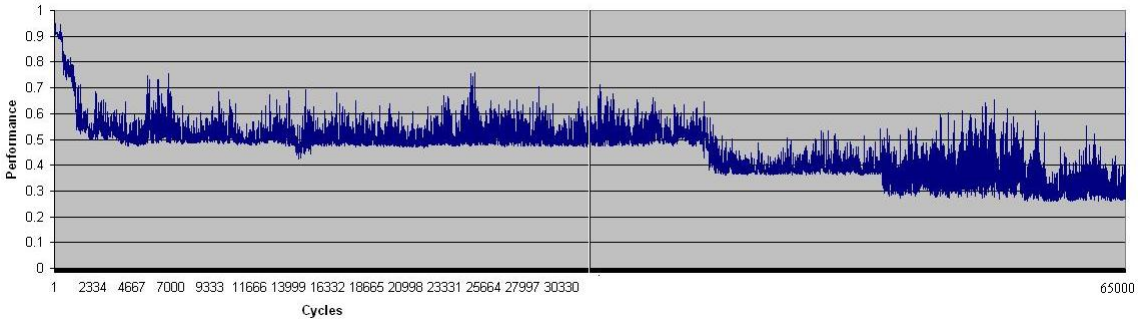

Figure 6.3: Results 65000 cycles, 20 hypotheses, asymmetric crossover

Again the result is in line with our expectations. Increasing the number of hypotheses seems to have an effect on the time it takes to increase the performance.

Table 6.7 shows that the system isn't sure about the Gin Highball and for the Scotch and Sky Highball only in approximately 50% of the cases.

Table 6.7: Estimation 65000 cycles, 20 hypotheses, asymmetric crossover

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**0.986**	0.006	0.004	0.000	0.000	0.000	0.004
Apple Brandy H	0.000	**0.997**	0.001	0.001	0.000	0.000	0.000
Bourbon H	0.000	0.000	**0.962**	0.000	0.038	0.000	0.000
Gin H	0.143	0.143	0.143	**0.143**	0.143	0.143	0.143
Scotch H	0.000	0.000	**0.533**	0.000	0.000	0.000	0.467
Carpano H	0.001	0.001	0.001	0.001	**0.996**	0.001	0.001
Mile H	0.001	0.000	0.000	0.000	0.001	**0.998**	0.000
Sky H	0.001	0.000	0.531	0.000	0.000	0.000	**0.467**
Seville H	0.000	0.000	0.000	0.000	0.000	**1.000**	0.000

The system seems to approximate the a priori experiment results quite reasonable in the previous case, but now we want to see what happens when we adjust the a priori results after cycle 65000. The test is to see whether the system indeed is able to track changes. Instead of the experiment results from Table 6.2 we have shifted the rating distribution for Apple Brandy H to the right and for Mile H to left as shown in Table 6.8.

Table 6.8: Cocktails ratings

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**1.0**	0.0	0.0	0.0	0.0	0.0	0.0
Apple Brandy H	0.0	**0.1**	0.0	0.0	0.0	**0.45**	**0.45**
Bourbon H	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
Gin H	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0
Scotch H	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
Carpano H	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0
Mile H	**0.9**	0.0	0.0	0.0	0.0	**0.1**	0.0
Sky H	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**
Seville H	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0

When we continue running with the new experiment results we get the graph of performance presented in Figure 6.4.

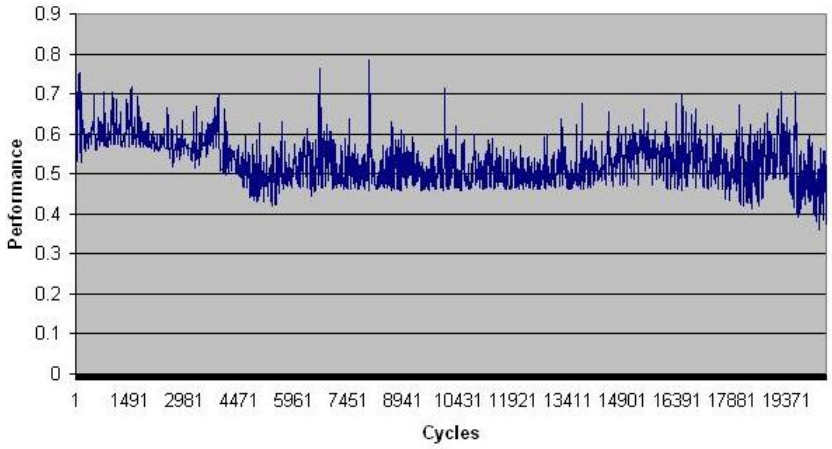

Figure 6.4: Results 20000 cycles after change

The beginning of the 20000 cycles after the previous 65000 cycles starts with a decrease in performance as the system is now more off with its predictions for Apple Brandy H and Mile H.

While its not going fast, the system indeed seems to adapt itself to the new ratings. Table 6.9 shows the predictions for the new experiment results.

Table 6.9: Estimation after change

Cocktail	Res. 0	Res. 1	Res. 2	Res. 3	Res. 4	Res. 5	Res. 6
Brandy H	**0.742**	0.043	0.043	0.043	0.043	0.043	0.043
Apple Brandy H	0.143	**0.143**	0.143	0.143	0.143	**0.143**	**0.143**
Bourbon H	0.001	0.001	**0.996**	0.001	0.001	0.001	0.001
Gin H	0.143	0.143	0.143	**0.143**	0.143	0.143	0.143
Scotch H	0.007	0.007	**0.484**	0.007	0.007	0.007	0.481
Carpano H	0.086	0.001	0.001	0.001	**0.910**	0.001	0.001
Mile H	**0.451**	0.021	0.021	0.021	0.044	**0.423**	0.021
Sky H	0.000	0.000	0.239	0.000	0.000	0.000	**0.761**
Seville H	0.021	0.021	0.021	0.021	0.021	**0.876**	0.021

Apparently it found out that the rating for Mile H is now more towards rating 0, but for Apple Brandy H it doesn't have a clue.

Chapter 7

Conclusion and Future Research

7.1 Conclusion

With this thesis we have taken on the challenge set by T. R. Addis et al. (see [4]):

"can we construct computing based upon family resemblance rather than sets, paradigms rather than concepts, and metaphor rather than deduction? Can we devise systems that have judgement rather than decisions?"

The thesis presents a model for an adaptive intelligent irrational system, which is able to assess the users cocktail taste and predict the user's taste for cocktails. For this we needed a system that could:

1. handle taste as an irrational set.
2. make sensible judgements (evaluations).

In order to achieve the main objective to create such an adaptive intelligent irrational system, we set the following assignment in our introduction (Chapter 1) which is repeated below:

1. design a model for the irrational set of taste, knowing that taste may change under time and context. Also as people cannot express their feeling of tastefulness in absolute statements we need to find a way to get the internal reference model of the user.

2. design a model for evaluation providing a mechanism in order to make predictions for the evaluation of cocktails. The model needs to be able to handle the irrational nature of taste and therefore keep track of changes in the taste of the user.

3. implement a prototype application, based on the taste and evaluation model, to create a test environment. With the application we can test whether the proposed models work or not.

4. test the system for a bounded set of cocktails. See if the system can assess the taste of an a priori set of cocktails. Also test how it adapts to changes in the cocktail taste of the user.

The model to handle with the irrational nature of taste and to get the internal references of a person was presented in Chapter 3. By modeling taste with a set of seven rating hypotheses and a belief distribution, we can approximate the internal reference model of the user.

With only the internal reference model of the users taste, there was no mechanism yet to create a belief model for an unknown cocktail. There was no mechanism for evaluation, no measure for success. Chapter 4 presented the Genetic model which I designed and implemented for the evaluation of cocktails providing a mechanism for prediction. A set of evaluation functions forms the belief profile, where each evaluation function is in fact an hypothesis for the Genetic Belief System. By adjusting beliefs we find out which evaluation functions predict the best.

The mechanism I designed for generating new hypotheses for the modified Belief System, enables us to handle the irrational nature of taste and thereby keep track of changes in the taste of the user. The generation of new hypotheses is made possible by using ΔLogic to describe cocktail evaluation functions in terms of genes. I designed and implemented genes which with we can breed new genes and thereby new evaluation functions. The model truly represents an Irrational system as it has the following three characteristics, stated in Chapter 2, needed to handle irrational sets:

1. It is dependent on time. As shifts occur over time and cannot be predicted, the shifts would be reflected in the order.

2. The learning period never ends. The system must continue evaluating and learning when presented with new ideas and situations.

3. Shows some form of randomness. While in a rational system, every decision can be made completely deterministic to get the optimum response, within an irrational system, no such optimum can be found as the sets change.

Chapter 5 presents the system design and my implementation for an application based on the Genetic model. I modified the Belief System which was written with Clarity, to make it suitable for handling evaluation functions. The Belief System can be seen as the evaluation part of my genetic algorithm.

Besides modifications I wrote a Dynamic Link Library (dll) written in C to perform the Genetic part. It does all the heavy computational breeding tasks, gene evaluation and gene creation.

I also builded a GUI written in C++ (MFC) on top of the Genetic Belief System

In Chapter 6 the model was tested with the implemented application and the results were promising. With the bounded set of cocktails from the Highball test case we showed that the system in time indeed would increase its performance on predicting the taste of the a priori experiments. The approximation of the experiment results is relatively poor, but still better than random.

When we adjust the a priori results, the system adapts to the new rating results, so it also seems to be able to track changes. In general we can say that the Genetic model works, but we have to point out that it is very impractical for two reasons:

- Firstly of all it takes a lot of cycles for the system to increase performance. In practise we need the system to adapt a whole lot faster to changes.
- Secondly, which relates to the first point, we need better performance, even after a long while the approximation of the experiment results is poor.

We can conclude by saying that with this thesis we made the first step towards a practical irrational system. The Genetic model presented in the thesis has all the features you would need to handle irrational sets, but the model and implementation need to be optimized if we want to make it useful in practise.

If this can be done then we really are on our way in creating intelligent creative systems, which can truly make sensible human-like judgments and thereby fulfilling the Challenge.

With this thesis I can conclude that it is possible to design and implement a working irrational system, which has the great advantage over the rational system that it is (forever) continuously adapting to its environment. In a rational system every case would have to be pre-designed or pre-determined, something which is impossible to do as the rational system should therefor account for all the possible future situations. Concepts will change under time and social context and we now have a system that can deal with it.

7.2 Future Research

For future research the following items must be recommended:

- Exploring the use of multiple populations for breeding. Instead of having only one belief profile with hypotheses, we should have multiple profiles and exchange and breed with genes of hypotheses between different profiles. In this way we can have more variation.

- Another field that needs to be explored is the use of parallel computing. With this we should be able to make some optimizations in the time it takes to find good predicting hypotheses. Especially if it is possible to compute for multiple belief profiles in parallel. This is certainly an important point as we need to improve a lot on speed to get a practical system. As soon as we have improved on the relatively small bounded test case, we can start testing for larger sets with more hypotheses and experiments.

- Optimizing the breeding process by finding better ways to select genes to breed with. Picking two good hypotheses to breed with is no guarantee for generating good hypotheses. There is no underlying gene structure for the genes of a hypothesis and therefor the location of the genes are insignificant. For crossover we now pick genes randomly as there is no information available on how 'good' or 'useful' a single gene (bit string) is. From an engineering point of view we should design a mechanism to assess genes. One could think of some kind of fitness function like we see in standard Genetic algorithms. If we can evaluate the genes themselves we could select the optimal ones. Or perhaps it is not a case of selecting optimal genes, but optimal gene matches.

Perhaps for this we should call upon nature and have a look at sexual attraction. If we could somehow calculate the sexual attraction between genes, then we could see which gene pairs are more suitable for crossover.

- Do more extensive testing to find out the optimal setting for the system's parameters. Or perhaps even find a mechanism for the system to change the parameters by itself dynamically.

Bibliography

[1] T. R. Addis. Stone soup: Identifying intellingce though construction. *Kybernetics*, 29:849–870, 2000.

[2] T. R. Addis and D. C. Gooding. Learning as collective belief-revision: Simulating reasoning about disparate phenomena. Edinburgh, 1999. AISB Symposium.

[3] Allen H. Dutoit Bernd Bruegge. *Object-oriented software engineering*. Prentice Hall, New-Jersey, 2000.

[4] T. R. Addis B. Visscher D. Billinge and D.C.Gooding. Socially sensitive computing, *a necessary paradigm shift for computer science*. The Grand Challenge in Non-Classical Computation `http://www.cs.york.ac.uk/nature/workshop/papers.htm`.

[5] Clarity. `http://www.claritysupport.co.uk`.

[6] Robert Cross. *The classic 1000 cocktails*. Foulsham, Berkshire England, 2003.

[7] Junk DNA. `http://en.wikipedia.org/wiki/Junk_DNA`.

[8] Hirpa L. Gelgele and Kesheng Wang. An expert system for engine fault diagnosis: development and application. *Journal of Intelligent Manufacturing*, 9:539 – 545, December 1998.

[9] Artificial Intelligence. `http://en.wikipedia.org/wiki/Artificial_intelligence`.

[10] Intelligence(trait). `http://en.wikipedia.org/wiki/Intelligence_%28trait%29`.

[11] W. C. Lefebvre J. C. Principe, N. R. Euliano. *Neural and Adaptive Systems, fundamentals through simulations.* John Wiley and Sons, New-York, 2000.

[12] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences 3*, pages 417–457, 1980.

[13] E. H. Shortliffe. Computer-based medical consultations: Mycin. *Elsevier/North-Holland*, 1976.

[14] Peter Norvig Stuart J. Russel. *Aritificial Intelligence A Modern Approach.* Prentice Hall, New-Jersey, 2003.

[15] D. Billinge T. R. Addis. The functioning of tropic communication: A mechanism for consistent figurative descriptions of artistic effect. AISB Symposium on AI and Creativity in Arts and Science, 2003.

[16] B. Visscher. Deltalogic. 2002.

[17] B. Visscher. *Exploring Complexity in Software Systems. From an irrational model of software evolution to a theory of psychological complexity and cognitive limitations based on empirical evidence.* PhD thesis, 2005.

Appendix A

Inventing Cocktails

I discuss two ways to create new cocktail combinations besides random cocktail generation:

1. Invention based on single replacements
2. Invention based on genetics

A.1 Invention based on single replacements

Inventing cocktails isn't something that has never been done before. It is not that new cocktails suddenly fall from the sky in order for us to decipher its components. We people create them by mixing known combinations of ingredients together based up on our previous experience with the ingredients, or based up on the experience of someone else. So to come up with an algorithm for inventing cocktails, we just looked at the way we solve the problem in the real world by asking our selves the question: "How should I invent a new cocktail?" The idea is basically that we take a cocktail we already like and then try to substitute one or more ingredients. It seems not the smartest thing to do by just plunge in all ingredients we like to make it great. For instance when I really like coca cola, strawberries, honey, chocolate and gin, then a mix of all this doesn't seem so tasteful anymore. We like to base the substitutions we make on our previous experience and general usage. So we will try to substitute one or more ingredients with ingredients that are used in similar combinations. Besides combinations also flavour seems quite useful. Flavour can help us ruling out certain combinations. If an ingredient is found in a cocktail with similar flavours, then this substitute should be in favor instead of the ones that are not. Finally we should

probably pick ingredients we like. We are not saying you should never mix with ingredients you don't like, because the combination with other ingredients may turn out to be quite nice. But in general it seems wise to favor ingredients you already like.

The idea of combinations is when we have for instance an ingredient "dry vermouth" that is often mixed with "gin". In other cases "dry vermouth" is sometimes mixed with "Canadian whisky". Knowing that both "gin" and "Canadian whisky" are mixable with "dry vermouth", perhaps then we can say that "gin" can be substituted for "Canadian whisky" and "Canadian whisky" substituted for "gin".

A.2 Breeding cocktails

Create paradigms of cocktails, cocktails that seem to be related showing some family resemblance.

Ingredients, additions and garnishes themselves can be grouped into the following 9 groups: Spirits (S), Liqueurs (L), Wines (W) Flavourings (F), Juices (J), Mixers (M), Syrups (R), Garnishes (G) and Cream (C).

Each paradigm is represented by a cocktail template, which can be seen as the genetic structure of the paradigm. The structure for a group could be: S S L L F J M C G G G G. This means that this paradigm could contain two spirits, two liqueurs, one juice, one cream and three types of garnish.

Genes can be filled or left empty. The value of a gene represents the part of the cocktail. For instance if S = 0.2, then it means that 20 percent of the cocktail consists of this Spirit. These ratio values aren't used for Garnish, they have value 0 or 1 (Present or absent). The total volume of a cocktail in calculated in advance.

For instance Rum Sour (where x represents empty):
Structure: S S x x x J x C G x x x
Ingredients: white rum, golden rum, x, x, x, lemon juice, x, syrup, spiral of lemon, x, x, x
Values: 0.4 0.15 0 0 0 0.3 0 0.15 1 0 0

Or Scotch Melon Sour:
Structure: S x L x x J x C G G G x
Ingredients: scotch, x, midori, x, x, lemon juice, syrup, lemon twist, slice of orange, cherry, x
Values: 0.25 0 0.25 0 0 0.25 0 0.25 1 1 1 0

Breeding involves two structures and crossing over a random number of ingredients:

Example:
Locations: 1 2 3 4 5 6 7 8 9 10 11 12
Rum Sour: S S x x x J x C G x x x
Scotch Melon Sour: S x L x x J x C G G G x

It is only possible to swap between the same gene types, so it's aloud to swap an S for an S ingredient. Take for instance swapping Rum sour ingredient 2 with Scotch Melon Sour ingredient 1, resulting in:

Rum sour2: white rum, scotch, x, x, x, lemon juice, x, syrup, spiral of lemon, x, x,x
Values 0.4 0.25 0 0 0 0.3 0 0.15 1 0 0

Scotch Melon Sour2: golden rum, x, midori, x, x, lemon juice, syrup, lemon twist, slice of orange, cherry, x
Values 0.15 0 0.25 0 0 0.25 0 0.25 1 1 1 0

Both list of gene values need to be normalized:
Rum sour2: 0.35 0.21 0 0 0 0.22 0 0 0.12 1 0 0
Scotch m s2: 0.19 0 0.27 0 0 0.27 0 0.27 1 1 1 0

So now we have two new cocktails.

Appendix B

Clarity

The tool used for implementing the Belief System is Clarity. Clarity is a program environment that allows you to draw your programs and then run them. Behind the scenes it converts your design drawing into a functional language (called Faith) that is then interpreted(see [5]). An example of a design drawing is presented in figure B.1, where it illustrates the implementation of the following function:

$$Example?0 ::= *(*(3)(2))(+(/(*(6)(+(2)?0))(4))(7))$$

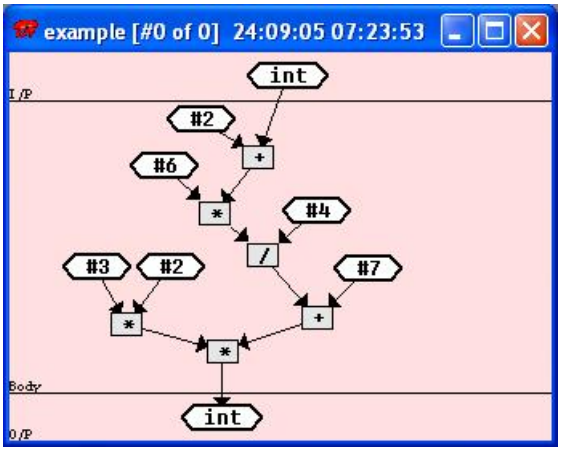

Figure B.1: Clarity function example

It takes some effort to find out what is being calculated, when we only look at the function declaration in the faith code. By drawing the same

function instead of declaring the function in the functional language, makes it a lot easier as we maintain a good overview.

Clarity is easy to work with and can be learned quickly. Programming in a functional language has never been this easy. The controls are quite intuitive and the schematics give you a good overview of the code, helping to manage the complexity. Errors you make are easily spotted or pointed out by Clarity, decreasing the time for debugging tremendously. With good error handling, the power of a functional language and pattern matching it's possible to write quite complex programs in a short amount of time.

The only problems were the lack of building interfaces, a bit too slow in heavy computational problems and difficulties in handling very large data sets. The first two problems can be solved by combining Clarity with for instance C++, C or any other language. A DLL is available to provide the necessary interface between the different programming languages. The option for parallel computation is an answer to the second problem, handling very large data sets, where parts can be distributed over a network of PC's.

Functional programming is a style that is tremendously powerful and capable of programming very complex ideas simply. The simplicity of functional programming is that there is really only one idea and that is the function. Programs are functions. So are data, procedures and sub-routines. In a pure functional language there are no global variables and places for storing data. So how do we store data for the Belief System, knowing that the Belief System is only defined by functions? For this we use the function definitions. Take for instance the case where we want to store the beliefs ($E(H)$) in each of the hypotheses. The system has a function called 'confidence', for which its schematic is presented in Figure B.2.

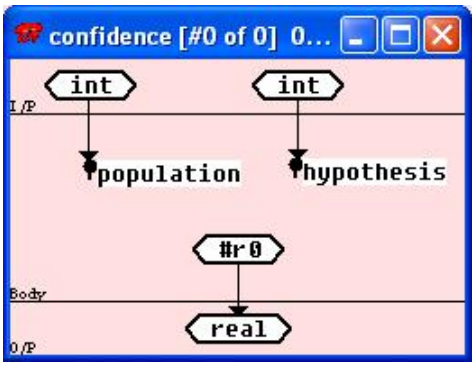

Figure B.2: Confidence

The schematic declares and defines the function respectively as:

```
confidence ::= int→int→real;
confidence ?0 ?1 ::= #r0.000000 ;
```

The two inputs of the function are the population number and the hypothesis number and for any input it returns the real number 0.000000. We can assert 'confidence' function definitions and thereby storing beliefs for hypotheses. Take for instance the assertion of the following definition:

```
confidence #0 #0 ::= #r0.250000 ;
```

Clarity uses pattern matching and is therefor able to return 0.25 when we evaluate function 'confidence' with an input of two zeros (H_0 from $Population_0$). Thus a function can be used as a dynamical array with the benefit that sparse arrays are only kept to only those components of the array that have significant results. These arrays are now accessible through patterns instead of simple integer indexes. Or Access them according through structures.