Current Tools for Assisting Intelligent Agents in Real-time Decision Making

Håkan L. Younes

Royal Institute of Technology
School of Electrical Engineering and Information Technology

December 1998

Abstract

Decision making is something with which we are all familiar. Each day we find ourselves in situations where we have more than one alternative to choose from. Most of our everyday decisions require little or no reflection. Yet, other decisions are more complex and may require thorough analysis. The field of decision analysis studies the application of decision theory to actual decision problems. Decision theory is the mathematical foundation for decision making, which combines probability theory and utility theory in order to describe what constitutes a rational decision. Today there are numerous tools for decision analysis that run on ordinary PCs. These make it easier for human decision makers to structure a decision problem, and help visualize the effects of possible outcomes. However, not only humans are decision makers. Intelligent agents are also faced with decision problems. This thesis investigates if any of the computerized tools for decision analysis can be used by intelligent agents. Special attention is given to real-time domains, where data that decisions depend on vary rapidly. These domains require quick responses from the tools. Otherwise the result of a decision analysis will be based on data that is no longer accurate. The objective of the thesis is to provide results indicating if any of the available tools for decision analysis are fast enough for use in real-time domains.

This thesis corresponds to the effort of 20 full-time working weeks.

To Barb

Contents

Chapter 1

Introduction

1.1 Background

Decision making is something with which we are all familiar. Each day we find ourselves in situations where we have more than one alternative to choose from. To act rationally, we should choose the alternative which gives the most gain according to our values. Most of our everyday decisions require little or no reflection. Yet, other decisions are more complex and may require thorough analysis. In all cases we base our decisions on the information we have gathered about the alternatives and the knowledge we have about our situation. The quality of the knowledge can vary from precise and known to imprecise and believed [Boman and Verhagen, 1998]. In a complex decision problem the quantity of data can be large, the relations between data can be intricate, and our uncertainty about the data and the relations will complicate the task of making the right decision. In domains such as health care and business, decision problems are often complex and the stakes can be very high, thus making it even more important to choose the right alternative.

The field of *decision analysis* evolved in the 1950s and 1960s, and studies the application of *decision theory* to actual decision problems [Fishburn, 1982]. Decision theory is the mathematical foundation for decision making, which combines probability theory and utility theory in order to describe what constitutes a rational decision. In the early 1980s, computers began to get a more significant role in decision making processes. At first, access to computer based tools was mainly restricted to expert users, but as ordinary desktop computers became more powerful in the 1990s, such tools became accessible to practically everybody. Today there are numerous tools for decision analysis that run on ordinary PCs, and are well integrated with other desktop applications. They make it easier for the decision maker to structure a decision problem, and help to visualize the effects of possible outcomes.

However, not only humans are decision makers. Imagine a robot sent out to collect geological samples in a volcano. It must make the right decisions in order not to fall down into the crater and be destroyed. A human could work as decision analyst, assisting the robot in making the right decisions based on data gathered by the robot's sensors. It would probably work well if the volcano was on Earth, but what if it was on Io—one of the planet Jupiter's moons?

The data from the robot would be at least 30 minutes old when it reached the person on Earth. Even if the decision is sent from Earth almost immediately afterwards, it would still take another 30 minutes before it reaches the robot. Since over an hour will have passed from the point in time when the robot scanned its environment until it can receive a suggested action based on this data from its human decision analyst, the situation might very well have changed dramatically, and the decision based on the old data can prove to be fatal in the new situation. Furthermore, it is sometimes a waste of resources if a human is always needed for supervision. In [Dorais *et al.*, 1998], an example with a Mars rover in an imagined manned mission to Mars is given. Although Mars is closer to Earth than Jupiter, the round trip time for a radio signal from Mars to Earth can be up to 40 minutes making it infeasible to tele-operate the rover from Earth. During a manned mission a crew member could tele-operate the rover, but if the rover could make decisions autonomously—in particular when travelling long distances over relatively uninteresting terrain—the crew member could attend more important matters. Still, we do not need to extend our scope to outer space, in order to find domains for non-human decision makers. On the Internet, electronic market places are rapidly emerging, where intelligent agents act as buyers and sellers on behalf of their human mandators (see e.g. [Chavez and Maes, 1996] or [Elofson and Robinson, 1998]).

In view of these examples, we may ask the following question: Can intelligent agents benefit from using current tools for decision analysis? First we must of course convince ourselves that any of the current tools at all can be used by intelligent agents. This is by no means obvious. Existing tools for decision analysis focus on human users. While humans benefit from extensive graphical user interfaces, intelligent agents require other means to communicate with the decision analysis tool. Whereas this might place practical obstacles in the way for intelligent agents to make use of the tools, it is of little principal interest. There are several effective techniques for adding an interface for intelligent agents to a tool without altering its main functionality (see [Genesereth and Ketchpel, 1994] or [Fou, 1997]).

Assuming that the communication issue can be solved (cf. [Boman, in press]), it seems reasonable to believe that current tools can be useful for intelligent agents—at least in situations illustrated by the last of the three examples above. However, the first two examples illustrate situations very much unlike decision situations for which most (if not to say all) current decision analysis tools where designed to be used—situations which require real-time decision making. Whether any of the current tools are useful in such situations must be empirically tested.

1.2 Objective

The main objective of this thesis is to investigate if any of the currently available tools for decision analysis can be used for assisting intelligent agents in real-time decision making. In order to achieve this, some of the available tools (see Chapter 3) will be selected and a separate *pronouncer*[1] will be built based on

[1]This is a term introduced in [Boman and Verhagen, 1998], where it is also further described and motivated. In e.g. [Boman, in press], decision module is used to denote the same type of entity. See also, Chapter 4.

each of the selected tools.

The primary measure for the pronouncers will be *speed*. The speed is important because the focus is on real-time applications. What will be measured is time passed from when an agent sends a decision problem to a pronouncer to when the agent receives an answer. By running tests on decision problems of varying complexity, it can be determined which of the available tools for decision analysis is the fastest. Depending on what is understood by "real-time" (which can vary between domains), it can then be determined whether any of the tools are fast enough for use in real-time domains. In this work, the RoboCup domain[2] is used as a reference domain.

Not only speed is important in evaluating the performance of a pronouncer. Finding a good utilization of pronouncers is equally important. This, however, is a whole research project by itself. Even if a pronouncer is found to be fast enough for a specific domain, the correct decision models must be created, and it must be decided when to make calls to the pronouncer. Without thorough analysis of the decision problems involved, using the pronouncer can be a burden instead of a gain. Some limited qualitative tests in the RoboCup domain were intended to be performed as part of this work with the UBU team [Boman *et al.*, 1998] developed by members of the DECIDE research group. Unfortunately, the UBU team has not yet reached a state where pronouncer calls can easily be incorporated. The main deficiency is the lack of explicit modeling of the uncertainty in the domain. Currently the uncertainty is handled in an ad hoc manner, which is highly inappropriate if pronouncers are to be used. Due to this, qualitative tests falls outside the scope of this work. Thus, the objective of the thesis, more clearly stated, is to provide results indicating if any of the available tools for decision analysis are fast enough for use in real-time domains—not how to make the best use of them.

1.3 Related Work

The usefulness of decision theory and decision analysis to software agents is investigated in [Bovin and Hammarberg, 1998]. The authors present a framework for using decision analysis in decision support systems for agents, but lack tools for implementing many of the features they propose. This work can fill that gap if any of the investigated tools are found useful.

In [Bovin and Hammarberg, 1998] real-time domains are not specifically considered. The idea of utilizing decision analysis for real-time decision problems can be found in [Boman, in press]. There, two provisos are made.

- Agents act in accordance with advice obtained from their decision module, with which they can communicate.
- The decision module contains algorithms for efficiently evaluating supersoft decision data.

The first proviso mainly concerns the communication issue which was briefly discussed at the end of Section 1.1, and is not very controversial. The second proviso, however, may be too restrictive since it rules out all but one of the available tools for decision analysis. Consequently, this work will not uphold this proviso.

[2]See [Kitano, 1998] for a comprehensive introduction to the RoboCup domain.

1.4 Methodology

The initial phase of the thesis work was partly dedicated to studying relevant literature on decision analysis and decision theory, in particular work concerning intelligent agents and multi-agent systems. In addition, a substantial part consisted in gathering information about available tools for decision analysis. This was first done by compiling a list of known tools. Later, information on each tool was extracted from the Internet, and each of the companies and academic research groups responsible for a tool were inquired for information of specific interest to this thesis.

After the initial phase followed a more thorough study of the theoretical foundations of involved techniques for decision analysis. This was accomplished by further literary studies, as well as practical training with some of the available tools.

When the preparatory and theoretical part of the work was done, some tools were selected for basing pronouncers on. Pronouncers were then implemented using a general architecture. Some of the tools had accompanying tutorials which made the pronouncer implementation easy. Help with the tools was also acquired directly from people responsible for development of the tools.

In the final phase, the pronouncers were tested. Suitable test cases were designed in co-operation with members of the DECIDE research group involved with RoboCup.

1.5 Structure of Thesis

In Chapter 2, the fundamentals of decision analysis needed to understand this work is presented. Those already familiar with the field of decision analysis could either skim through this chapter just to capture the terminology used throughout the thesis, or skip the chapter altogether. Chapter 3 gives an overview of current tools for decision analysis, and provides a brief statement for each of the tools in what way (if at all) they could be useful to intelligent agents. Reading this chapter should be sufficient for people only interested in getting a picture of what is currently available, and some preliminary recommendation for which tools may be interesting to consider basing a pronouncer on. Chapter 4 deals with implementation issues for pronouncers. It elaborates on matters briefly touched upon in Chapter 3. Basic functionality of a pronouncer is defined here, which is later used when actually implementing pronouncers. Chapter 5 describes the implementation of three pronouncer in detail. A general framework for pronouncer implementation is presented here. This is followed by Chapter 6, which describes the tests that have been performed with the implemented pronouncers. The final chapter contains a discussion on what has been achieved with this work, as well as what can be done in the future.

Chapter 2

Decision Analysis

2.1 Introduction

Decision analysis is the application of decision theory to actual decision problems. This chapter introduces the elements of decision analysis as well as decision theory referred to and used in subsequent chapters. Various techniques for modeling and solving decision problems are presented. Methods for handling vagueness in descriptions of decision situations are briefly discussed. A section about sensitivity analysis is also included, although this is a subject that will be given limited attention in this work.

2.2 Modeling Decision Problems

The first step in analysing a decision problem is to create an explicit model of the problem. There are several techniques for doing this, and those techniques of relevance to this thesis are presented here. First, the following concepts should be introduced:

Probability Uncertainty in a decision problem can be modeled with probabilities. If A is an event, then the probability of A occurring is denoted by $P(A)$. This is a value between 0 and 1 when it is uncertain whether A will occur or not, it is 1 if A for certain will occur, and it is 0 if A for certain will not occur. Probabilities can be inherent in the domain of the decision situation, as in lotteries where the probability of winning is the number of winning tickets divided by the total number of lottery tickets. Probabilities can also be subjective, and are then only manifestations of the decision maker's beliefs. The only restriction placed on the decision maker when assigning subjective probabilities is that they must conform to the axioms of probability (see e.g. [Russell and Norvig, 1995]). *Conditional probabilities* are used when information about the situation has been acquired. If A and B are events, then $P(A|B)$ denotes the probability of A occurring given that B has occurred. Conditional probabilities are also called *posterior probabilities*—then in contrast to *prior probabilities*, which is an alternative name for unconditional probabilities.

Utility In a decision situation there are several possible final outcomes. In order to concretize how desirable an outcome is to the decision maker, a value is assigned to each outcome. This value is referred to as the utility of an outcome. Sometimes utilities represent values in the real world. For example, the utility of winning $1,000 on a lottery ticket bought for $10 could be set to 990, while the utility of not winning could be set to -10. However, in many situations there is no straightforward measure to adopt from the domain of the decision situation, and a subjective value must be used for the utility of an outcome. Even in cases where objective measures do exist, it can be necessary to account for subjective judgments when assigning utilities. Imagine a person with severe toothache. If dental service costs $100, then having $100 would enable the person to cure the toothache, while having $90 would not be of much help to the person.

2.2.1 Decision Trees

In a *decision tree*, a decision problem is modeled chronologically. It starts with a *decision node* (depicted as a square) in the root of the tree. A decision node denotes a point in time where a decision has to be made. From a decision node, two or more branches emanate—each branch representing an alternative available to the decision maker. Each of the branches lead to a new node. These nodes can be additional decision nodes. They can also be either *chance nodes* (circles) or *result nodes* (triangles). A chance node stands for a probabilistic event in time, the outcome of which is not under the influence of the decision maker. Branches that emanate from a chance node represent possible outcomes of the event. Finally, a result node denotes an outcome of the entire decision problem. Consequently, no branches emanate from a result node, and all leaf nodes in a decision tree must be result nodes. A path from the root of the decision tree to one of the result nodes is often referred to as a *scenario*.

This only gives the structure of the tree. In order to make a complete decision tree, certain values must also be added. First, probabilities have to be assigned to each of the branches emanating from chance nodes. Probabilities for branches emanating from the same chance node must sum to 1. This is because all possible outcomes of a probabilistic event should be included in the model, and the outcomes should be mutually exclusive. Second, utilities must

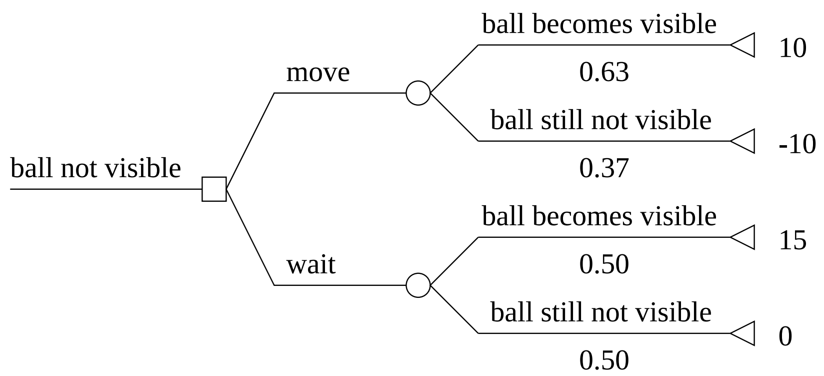


Figure 2.1: A sample decision tree.

be assigned to all result nodes. Once these two types of values have been added, the decision tree is complete.

Figure 2.1 shows a complete decision tree. The decision maker—in this case a soccer playing intelligent agent that cannot see the ball—has two alternatives: move, and wait. Both alternatives lead to a chance node with two possible outcomes—either the ball becomes visible to the agent, or it remains out of sight. The subjective probabilities for each of the outcomes are given in the tree. For example, the probability of the ball becoming visible if the agent makes a move has been estimated by the agent to be 0.63. Utilities are given at each result node.

2.2.2 Belief Networks

A *belief network*[1] is used to represent dependencies between random variables. Each node in the network denotes a random variable, and a directed link between two nodes represents dependencies between the two random variables denoted by the involved nodes. The node at the starting-point of a link is called a *parent* of the node at the end-point of the link. A conditional probability table is associated with each node. The table quantifies the effects that the parents have on the node. The network may not contain cycles.

As before, imagine a soccer playing intelligent agent that cannot see the ball. It wants to find out the probability of the ball entering its field of vision, given its beliefs about the current movement of the ball—whether it is moving towards or away from the agent. The belief network given in Figure 2.2 represents this situation. Probabilities, representing the agent's beliefs, are given in the tables associated with each of the nodes in the network.

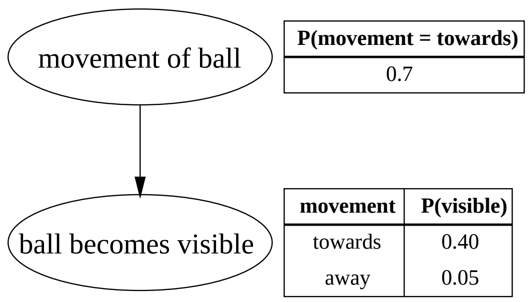


Figure 2.2: A sample belief network.

Belief networks are not primarily intended for modeling decision problems, since they do not provide explicit means to model alternatives and utilities. Numerically, a belief network is just a representation of the joint probability distribution among the involved random variables [Russell and Norvig, 1995]. An extension of belief networks that is better suited for modeling decision problems is influence diagrams.

[1] Other names for this structure that often occur are *Bayesian network*, *probabilistic network*, and *causal network*.

2.2.3 Influence Diagrams

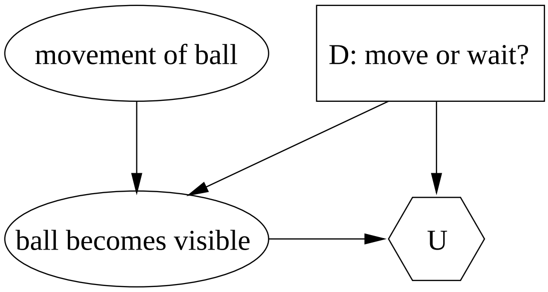
In addition to nodes for representing random variables provided by belief networks, *influence diagrams* also provide node types for modeling alternatives and utilities. *Chance nodes* (depicted as ovals) denote random variables, and correspond to the only node type available in belief networks. Alternatives are modeled with *decision nodes* (rectangles). A decision node indicates a decision facing the decision maker—similar to decision nodes in decision trees—and contains all alternatives available to the decision maker at that point. The third and final node type provided by influence diagrams is *utility nodes* (diamonds). Nodes of this type represent the utility function of the decision maker. In utility nodes, utilities are associated with each of the possible outcomes of the decision problem modeled by the influence diagram.

Directed links between nodes represent influences. Links between two chance nodes have the same semantics as in belief networks. Other links in an influence diagram may also represent a temporal relation between the nodes involved. For example, a link from a decision node to a utility node not only indicates that the choice of action influences the utility, but also that the decision precedes the outcome in time.

Influence diagrams are useful in bringing out the structure of a decision problem. While for example decision trees are more effective at presenting the details of a decision problem, influence diagrams more clearly show factors that influence a decision. This makes influence diagrams a fairly compact—but nevertheless very powerful—way of modeling decision problems.

In Figure 2.3, the belief network from Figure 2.2 has been extended to a influence diagram by the addition of a decision node and a utility node. As in the example from Section 2.2.1, the decision to make is whether to make a move or to wait[2]. The tables in the figure show probability distributions for the random variables, and the utility function for the utility node.

P(movement = towards)
0.7



movement	D	P(visible)
towards	move	0.50
towards	wait	0.40
away	move	0.20
away	wait	0.05

visible	D	U
T	move	10
T	wait	15
F	move	-20
F	wait	0

Figure 2.3: A sample influence diagram.

[2]Without the chance node "movement of ball", the two decision problems would be structurally equivalent.

2.2.4 Hierarchies of Criteria

Both decision trees and influence diagrams focus on the events that can affect the outcome of a decision situation. An alternative approach is to focus on the *criteria* affecting the decision, and build *hierarchies* of these criteria. At the top level of the hierarchy is the *goal* to attain by making the decision. The second level of the hierarchy consists of criteria directly affecting the goal. Subsequent levels of the hierarchy consist of *subcriteria* contributing to the assessment of criteria (and subcriteria) in levels above. At the very bottom of the hierarchy are the alternatives available to the decision maker.

Once a hierarchy has been constructed, each *element* (criterion or alternative) in the hierarchy must be assigned weights. The weights of an element determines its level of influence on criteria above it compared to other elements at the same level in the hierarchy. Weights can be assigned by *pairwise comparison* between elements at the same level. What must be considered when making the comparison is *dominance* and *intensity*. Dominance determines which of two elements meets a criterion more. Intensity stands for how much one element dominates another.

In Figure 2.4, the goal is to score on a penalty kick in a soccer game. Two players are being considered for taking the penalty kick. In this very simple example there is only one level of criteria, which contains "self-control" and "skill". Weights, on a scale from 0 to 100, are given next to each link. For example, it can be read that self-control is an extremely important criterion for succeeding in scoring on a penalty kick, while skill is a criterion with slightly less influence on the goal node.

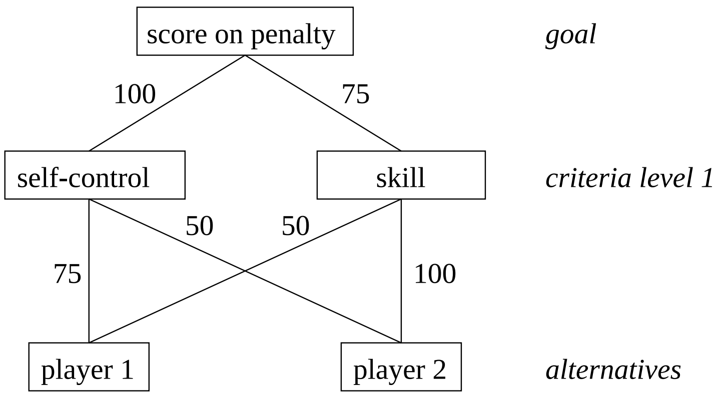


Figure 2.4: A sample decision problem modeled as a hierarchy of criteria.

2.3 Solving Decision Problems

There are many methods for solving decision problems. Which method to use is partly dependent on how the decision problem has been modeled. Finding the solution often involves calculating the expected utility of each alternative available in the problem. If the decision problem has been modeled as a decision tree this is a fairly straightforward process, and can be accomplished with a dynamic programming algorithm called *averaging out and folding back* [Raiffa, 1968].

For influence diagrams, there are several approaches to calculating expected utilities. Influence diagrams were first introduced in [Howard and Matheson, 1984], and there a method for deriving a decision tree from an influence diagram is presented. The decision tree can then be solved with conventional methods, but in general the tree is of exponential size. A direct method for solving influence diagrams by reversing links and removing nodes was presented in [Shachter, 1986]. Yet another approach is to transform the influence diagram into a belief network [Jensen, 1996].

Inference in belief networks is based on Bayes' theorem for conditional probability and is called *Bayesian updating* or *belief updating*. The method amounts to calculating the posterior probability distribution for some random variables given the exact value for other variables. In [Pearl, 1986], a message-passing scheme is given for updating probability distributions in *singly connected* networks[3] (also known as *polytrees*). An algorithm that works with general networks is given in [Lauritzen and Spiegelhalter, 1988]. It transforms the belief network into a junction tree of cliques before performing probabilistic inference.

Multi-criteria decision analysis constitutes a large area of research within the field of decision analysis. A method for solving decision problems modeled as described in Section 2.2.4 is *the Analytic Hierarchy Process* (AHP) [Saaty, 1990]. There is a potential drawback with AHP if alternatives are added to the initial model later. Due to the relative nature of judgments in AHP, adding a new alternative—even if it is very poor—can reverse the relative ranking of previously top-ranked alternatives. *The Simple Multi-Attribute Rating Technique* (SMART) is similar to AHP in that it works with hierarchies of criteria, but since it does not use relative ratings of alternatives it is less sensitive to addition of alternatives at later stages.

2.4 Supersoft Decision Theory

In classical decision theory, probabilities and utilities are assigned exact values. The probability of a given event occurring can for example be set to 0.17. This often gives a pretence of accuracy, as decision makers in most realistic situations lack foundation for preferring a certain value like 0.17 to other values close by. This problem is addressed in *supersoft decision theory*, which allows assessments of probabilities and utilities to be represented by vague and imprecise statements [Malmnäs, 1995]. This could be statements like "Event A is very likely to occur" and "Outcome o_1 is slightly more desirable than outcome o_2, but both are quite undesirable". The former is an example of a *qualitative* statement, while the first part of the latter is an example of a *comparative* statement.

In order to handle vague and imprecise statements computationally, they must be translated into a quantitative form. A number of translation rules from linguistic statements to intervals of probabilities and utilities are given in [Malmnäs, 1994]. In [Danielson, 1997], the DELTA method for solving decision trees containing so-called supersoft decision data is presented. Since probabilities and utilities are given as intervals, the expected utilities will not take on

[3] In a singly connected network, there is at most one undirected path between any two nodes.

single values. Usually the possible expected utilities of alternatives overlap, making it non-trivial to compare them. The DELTA method measures the relative strength between two alternatives in order to determine *dominance*. The dominance of one alternative over another can be either weak or strong, thus indicating how much better an alternative is compared to other alternatives.

2.5 Sensitivity Analysis

A *sensitivity analysis* can be performed in order to determine how variations in one or more of the quantities involved in a decision problem affect the result of a decision analysis. Since probabilities and utilities often are uncertain, a sensitivity analysis yields insight into where more effort should be put in order to make the decision more certain. Traditionally, a sensitivity analysis is carried out by studying the effect of varying one or a few input variables at a time. The result of a *one-way* sensitivity analysis (changes in the value of a single input variable) is thresholds for the involved variable for when the optimal strategy changes. When one strategy is better than other strategies over the entire interval included in the analysis, the decision problem is insensitive to changes in the involved variable. A sensitivity analysis can also be *multi-way* (simultaneous changes to multiple input variables), in which case the resulting thresholds are hyperplanes.

Even in cases where supersoft decision data is used in the representation of the decision problem, a sensitivity analysis can be useful (c.f. [Boman, in press]). Although intervals for probabilities and utilities are already used, and dominance gives a measure on the closeness between alternatives, varying the intervals slightly (e.g. by contraction or expansion) will reveal which factors the dominance depends on the most.

Chapter 3

Current Tools

3.1 Introduction

Decision analysis is a computationally demanding task. Until the early 1980s, computers were only used to a limited extent by decision makers. Along with the rapid progress made by the computer hardware industry in the past few decades, this has changed. Today, practically every decision maker has access to a computer. Consequently, a great number of tools have been developed to meet the demands of human decision makers. This chapter starts with an overview of what these tools have to offer human decision makers. The overview is followed by a discussion on aspects where intelligent agents differ from human decision makers, and what impact this has on the demands for the tools. Attention is then given to tools that at a first glance appear to be useful for intelligent agents, and it is investigated how any of these tools could be used to assist intelligent agents in making decisions. The chapter concludes with some preliminary findings.

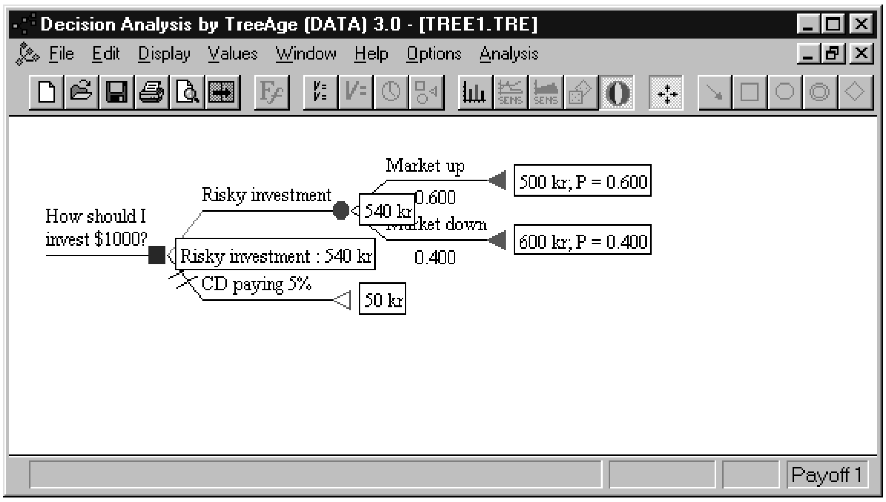


Figure 3.1: Screenshot from DATA 3.0 showing a simple decision tree.

3.2 Decision Analysis for Humans

Available tools for decision analysis have been designed with human users in mind. They are well integrated with other desktop applications for PCs, and are equipped with intuitive graphical user interfaces (Figure 3.1). These allow the decision maker to focus on the structure of a decision problem, and help visualize the effects of possible outcomes.

Several tools take the integration with other elements of the decision maker's desktop environment one step further. These are the so-called *add-ins* for existing spreadsheet applications, which do not work as stand-alone applications. Instead they make functionality for decision analysis directly available in the spreadsheet application (Figure 3.2). In many decision situations (e.g. in business activity), data used in the model of the situation come from spreadsheets. Thus, add-ins allow decision makers to transparently build models where data for the models is found.

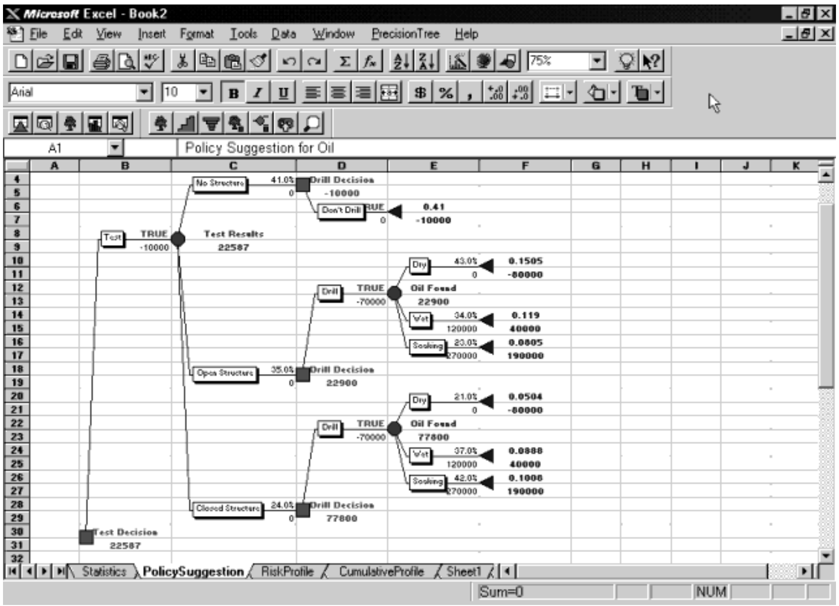


Figure 3.2: Screenshot from Microsoft Excel with PrecisionTree as add-in.

3.3 Decision Analysis for Intelligent Agents

To an intelligent agent, an extensive graphical user interface (GUI) is of little use. Other means to interact with the decision analysis tool are needed. While this does not per se disqualify tools that work as add-ins for spreadsheet applications, there are other reasons for not including such tools in any further discussion. As has been argued for, add-ins make perfect sense in a situation where data for the decision analysis is taken from a spreadsheet. In general, this will not be the case for the intelligent agents mainly concerned

	build model	*load model*	*set value*	*type of interface*
Ergo	x	x	x	C library
GeNIe/SMILE	x	x	x	C++ library
Hugin	x	x	x	C library
Netica	x	x	x	C library
Analytica	x	x	x	OLE
DATA		x	x	DDE, OLE
DPL	x	x	x	DDE
DecisionPro	x	x	x	DDE, TCP/IP
Delta	x		x	C library
Expert Choice				(GUI only)
Criterium DecisionPlus		x	x	OLE

Table 3.1: Functionality provided by interfaces suitable for intelligent agents.

in this thesis—agents in real-time applications. Furthermore, it seems safe to assume that stand-alone tools will perform at least equally well as any add-ins with equivalent functionality—this without adding the computational power of a spreadsheet application, which in most cases will prove superfluous and sometimes even a burden.

Most current tools for decision analysis do provide means of interaction other than through a GUI. The functionality provided through these alternative interfaces, compared to what is provided through the GUI, vary. Some tools (e.g. DATA) only provide an alternative interface whose main purpose is to allow sharing data with other applications (e.g. a spreadsheet application). These interfaces are often limited to providing functionality for reading and modifying values in the model of the decision problem. For other tools, alternative interfaces exist that provide total control over model building and inference in models (e.g. Netica).

In order to build a basic pronouncer based on a tool for decision analysis, the minimum requirement is that values (probabilities, utilities, etc.) in an already built model can be set by the agent using the pronouncer. A more advanced pronouncer could allow the agent to build the model from scratch, and modify the structure of the model at later stages (e.g. by removing or adding alternatives). A more thorough discussion on implementation issues for pronouncers is given in Chapter 4. Table 3.1 shows to what extent these functionalities are provided by alternative interfaces of current tools, and also what type of interfaces these are. Evidently, all tools except Expert Choice do provide functionality so that at least a basic pronouncer could be based on them.

3.4 A Closer Look at the Tools

The various stand-alone tools that exist for decision analysis are mainly based on the different techniques for modeling and solving decision problems briefly introduced in Chapter 2. Table 3.2 shows which of these techniques are supported by which tools. In the closer presentation of the tools below, they have been divided into groups roughly based on which techniques they support. For each tool listed, the name of the company or academic institution responsible

	BN	*ID*	*DT*	*AHP*	*SMART*
Ergo	x				
GeNIe/SMILE	x	x			
Hugin	x	x			
Netica	x	x			
Analytica		x			
DATA		x	x		
DPL		x	x		
DecisionPro			x		
Delta			x		
Expert Choice				x	
Criterium DecisionPlus				x	x

Table 3.2: Support for belief networks (BN), influence diagrams (ID), decision trees (DT), AHP, and SMART in stand-alone tools.

for the tool is given in parenthesis. A list of Internet URLs for the companies and academic institutions is given in Appendix A.

3.4.1 Tools Based on Probabilistic Inference

These are tools that have belief networks as basic structure, and use probabilistic inference for solving decision problems.

Ergo (Noetic Systems Inc.)

In Ergo, decision problems can only be modeled as belief networks. As mentioned in Section 2.2.2, belief networks do not provide any explicit means for modeling alternatives and utilities. Consequently, Ergo is a less interesting alternative for basing a pronouncer on. Nevertheless, it would be possible. A C function library supplies an API with functionality virtually identical to that of Ergo.

GeNIe/SMILE (Decision Systems Laboratory, Univ. of Pittsburgh)

Graphical Network Interface—GeNIe for short—uses probabilistic inference to solve decision problems. Belief networks and influence diagrams can be used to model the problems. In reality, GeNIe is just a graphical interface to SMILE (Structural Modeling, Inference, and Learning Engine). SMILE is a platform independent library of C++ classes which contains functionality for building and modifying decision models, and for making inferences in such models. SMILE implements several algorithms for probabilistic inference. Both exact algorithms mentioned in Section 2.3 are included, as well as a number of approximate algorithms based on stochastic sampling.

Hugin (Hugin Expert A/S)

Hugin is another tool that works with both belief networks and influence diagrams. Models are compiled into junction trees for fast probabilistic reasoning.

The inference engine is available as a platform independent C function library, which also provides complete control over model building.

Netica (Norsys Software Corp.)

Netica is very similar to Hugin. Belief networks and influence diagrams are supported and—as with Hugin—these are compiled into junction trees for increased performance. Support for entering positive, negative, and likelihood findings is also provided. This can be used for probabilistic learning. Netica API—a platform independent C function library—provides total access to the functionality of Netica.

3.4.2 Tools for Classical Decision Analysis

This group basically contains all those tools that do not fit into any of the other groups. They all require probabilities and utilities to be given as exact values. All but one of these tools uses decision trees as basis for models of decision problems.

Analytica (Lumina Decision Systems)

Analytica only allows decision problems to be modeled using influence diagrams. Analytica Decision Engine (ADE) gives access to the functionality of Analytica through OLE Automation.

DATA (TreeAge Software, Inc.)

DATA, available for Windows and MacOS, works with decision trees and influence diagrams. The Windows version includes a scripting language called DATAScript, which enables external access to the program through DDE connections. DATAScript allows for probabilities and utilities to be modified, and analysis results to be retrieved. However, models can only be created through the GUI.

An ActiveX control, called ActiveDATA, which contains the inference engine of DATA is currently under development. Through OLE Automation, models can be loaded (only decision trees), probabilities and utilities modified, and analysis results retrieved. ActiveDATA is better suited for basing a pronouncer on than DATA.

DPL (Applied Decision Analysis, Inc.)

DPL stands for Decision Programming Language. It is a tool based on a programming language for decision analysis and runs on Windows platforms. Both decision trees and influence diagrams can be created and controlled through the language. A special developer version of the tool is available, which enables external applications to utilize the decision engine in DPL through DDE connections.

DecisionPro (Vanguard Software Corp.)

In a sense opposite to Analytica, DecisionPro works only with decision trees. DecisionPro allows for external access to the functionality of the tool through DDE or TCP/IP network connections.

3.4.3 Tools for Supersoft Decision Analysis

This group consists of tools that support supersoft decision analysis (see Section 2.4). Only one of the surveyed tools falls into this category.

Delta (DECIDE, Stockholm University/KTH)

In Delta, decision problems are modeled as decision trees, with probabilities and utilities stated on interval form. Comparative statements are represented by links between consequences. Delta is the only tool which also allows partial information, i.e. not all probabilities and utilities have to be specified in the model prior to carrying out a decision analysis. The functionality of Delta is available through a platform independent C function library called DeltaLib.

3.4.4 Tools for Multi-criteria Decision Analysis

Tools in this group are based on multi-criteria decision analysis.

Expert Choice (Expert Choice, Inc.)

Expert Choice could be considered as the standard tool for decision analysis using AHP. Yet, since it does not provide any other means of interaction other than a GUI, it is of little interest for this work.

Criterium DecisionPlus (InfoHarvest, Inc.)

Criterium DecisionPlus supports both AHP and SMART. Model building can only be performed through the graphical editor. However, an ActiveX control called WDObj (Weighted Decision Object) allows models to be loaded, weights to be altered, and alternatives to be ranked through OLE Automation.

3.5 Preliminary Findings

As has been noted in this chapter, most of the surveyed tools are likely to be useful to base a pronouncer on. The most common ways making the decision analysis mechanisms available to other software entities are through a C/C++ function library, or a dynamic link library accessible through OLE Automation. Both of these types of "interfaces" to the inference engines are fairly easy to use, and can either be utilized directly in the code of the agent or in a separate pronouncer called upon by the agents.

Ergo lacks explicit means for modeling decision problems and is thus left out of further consideration. The other three probabilistic tools GeNIe/SMILE, Hugin, and Netica are all very similar in the functionality that they provide, and could all provide an excellent basis for a pronouncer. Empirical tests have to be performed in order to determine if there are any significant differences in performance between them, and if any of them are fast enough for real-time decision making. Hugin has been used in a control system for unmanned underwater vehicles [Lockheed Martin, 1996], but there the real-time reasoning was handled by a rule-based system while Hugin was used for overall mission planning. It is not evident whether Hugin was ever considered for use in the real-time component, so the example does not necessarily rule out Hugin—or any of the other probabilistic tools—as useful for real-time decision making.

Delta is interesting because it is the only tool allowing supersoft decision data. Unfortunately the documentation of the API is almost non-existent—something which the commercial tools (and even GeNIe/SMILE) do not suffer from. This makes it harder to utilize the API, in particular since the decision models are somewhat more complex than in most other tools. In [Boman and Verhagen, 1998], it is also indicated that Delta might be too slow for use in real-time situations. A new version of the tool based on a refined version of the original theory—more appropriate for real-time use—is planned, but not yet completed. Until then, it would still be interesting to obtain measures on the performance of Delta.

Of the other tools, primarily DATA (in particular ActiveDATA) and Criterium DecisionPlus (WDObj) could be worth taking a closer look at. This would give at least one candidate for a real-time pronouncer from each of the groups as given in Section 3.4. The remaining tools—Analytica, DPL, and DecisionPro—are all good candidates as well. At this stage they are left out mainly because they are not as readily available as the other tools, but they should be taken into consideration again if all the other tools prove to have insufficient performance.

Chapter 4

Basing Pronouncers on Current Tools

4.1 Introduction

In [Boman and Verhagen, 1998], a *pronouncer* is defined to be an entity giving advice to intelligent agents. It is distinguished from a *decision module* in that it suggests an extrinsic entity, while a decision module suggests an entity intrinsic to an intelligent agent. Furthermore it suggests "that the advice given is formal and authoritative, giving the entity a normative status" (p. 3, ibid.).

In this chapter, general implementation issues for pronouncers based on current tools for decision analysis are discussed. In particular, it is discussed what functionality a pronouncer needs to provide, and the term *basic pronouncer* is established. The question of which problem representation to use for intelligent agents using a pronouncer is also treated, as well as the question of whether the decision analysis functionality should be made internal or external to agents.

4.2 Functionality of Pronouncers

What functionality should a pronouncer provide? An answer to that question was hinted in Section 3.3. Here, a more thorough treatment of the question is given. What is referred to as *basic functionality* below could be seen as a minimum requirement for a pronouncer. A pronouncer implementing only basic functionality is called a *basic pronouncer.* When considering if a certain functionality should be categorized as basic, special attention is given to its potential use for intelligent agents requiring real-time decision analysis, but also what can actually be done with most current tools for decision analysis is included as a factor.

4.2.1 Basic Functionality

The purpose of a pronouncer is to give advice to intelligent agents in decision situations. The agents are likely to be situated in dynamic environments, so the functionality of the pronouncer should reflect this in order for the pronouncer to be of any essential interest. This can be provided for in different ways, with varying expressibility for the agents as result. On one extreme, the agents could be allowed to formulate decision problems on the fly. This requires that the pronouncer is based on a tool that supports model building. In [Boman, in press], it is noted that formulating a decision problem from scratch is extremely difficult for an agent. Instead it is suggested that agents can use template models developed in advance for decision situations that can be presumed to occur. In real-time decision situations, using templates can be further motivated by the gain in speed by having an already constructed model compared to having to build the model when faced with the decision problem. Given a template, an agent can then—to varying extent—be allowed to modify the model before an advice is queried for. At the very least, agents must be allowed to set and modify values (e.g. probabilities and utilities) in the model. A more advanced pronouncer could allow for structural modifications of a model, but since this rules out a number of otherwise interesting tools (see Table 3.1) it is not considered as basic functionality.

To conclude this discussion, basic functionality is then restricted to comprise the following:

- load template models
- set/modify values
- evaluate model, and answer with a single recommended alternative

As has been argued for, this functionality is an absolute minimum for any pronouncer. In many real-time decision situations it should also be sufficient. In addition, the restrictions made allows for most of the tools presented in Chapter 3 to constitute the basis for a basic pronouncer.

4.2.2 Extended Functionality

There is hardly a limit to what could be included in the concept of extended functionality. Here, only functionality that is supported by some of the tools for decision analysis is concerned.

One obvious candidate is structural modification of models, which was dismissed as basic functionality in the previous section. Its main use would perhaps be to allow for alternatives to be added to or removed from a template model. The example in [Boman, in press], with robots collecting radioactive material in a room, illustrates well how it could be useful. In the example, a robot experiences a malfunction in one of its legs. It then has to decide which robots in the room that it should engage in repairing the leg. The template model for the given decision situation yields two alternatives: for the robot to repair the leg itself, or for the robot to seek assistance from another robot in the room. In reality, the second alternative may comprise more than one alternative—viz. one for each robot that the damaged robot believes can perform the repair. At one point in time it might believe that two other robots are able to perform the repair, while later it can have been brought to its attention that yet another robot also is capable of doing the work. This kind of change would best be addressed by a change in the structure of the decision model.

A basic pronouncer only returns a recommended alternative when giving advice, without further motivating the choice or indicating how much better it is compared to other alternatives. A more advanced pronouncer could make use of sensitivity analysis (see Section 2.5), enabling it to give an indication of which factors the advice rely on the most. In some cases this indication could give the agent receiving the advice a hint on what kind of information it should put more effort into acquiring.

An interesting feature of tools based on probabilistic inference is that of *findings* (also called *evidence*). A pronouncer making use of this feature would allow agents to enter findings as they acquire knowledge about the world. The agents could in this way gradually build up a more accurate model of a decision situation.

4.3 Problem Representation

One important issue when designing a pronouncer is how intelligent agents should represent their decision problems when communicating with the pronouncer. In Chapter 3 it was shown that there is no single representation of decision problems that all current tools for decision analysis can handle. In [Bovin and Hammarberg, 1998] it is suggested that the agents could use any representation they like, and that this representation could be translated by a wrapper into a representation that the pronouncer understands. The authors recognize that the translation can be time-consuming, making this approach unattractive in domains where time is of essence. This is of course a most relevant observation to this work, since the focus is on real-time decision making. However, there is a more crucial reason why using a translation wrapper is not always a feasible approach, viz. the fact that some representations of decision problems are not translatable into some of the other representations. Influence diagrams can be transformed into decision trees and vice versa, but neither of these can be transformed into the hierarchies of criteria used by tools for multi-criteria decision analysis. Furthermore, if the agent uses supersoft decision data, there is no evident method for translating these into data that tools based on classical decision analysis can utilize without loosing fundamental information.

Consequently, when designing an agent that makes use of a pronouncer it is important to know what kind of models the pronouncer supports. In some cases it will be possible to switch pronouncer without having to modify the agent. In particular, this may be the case when switching between pronouncers using the same model representation. Although the pronouncers might use different protocols for communication, a wrapper could in such cases be used to translate between the protocols. Time will still be an issue, but this kind of translation is likely to be less time-consuming than translations between models, making the use of a wrapper more feasible.

4.4 Situating the Pronouncer

A pronouncer can either be used as a decision module, local to each agent, or be an external entity that querying agents call upon. In [Boman and Verhagen, 1998], the latter alternative is chosen. The reason is that the pronouncer then can account for social factors when evaluating a decision problem, thus helping to achieve social intelligence. When using a pronouncer for real-time decision making, the extra time it inevitably will take for an agent to communicate with an external entity must be taken under consideration. In cases where this is a significant factor, an internal entity might be the only feasible choice.

Chapter 5

Implementation of Pronouncers

5.1 Introduction

Chapter 4 discussed the implementation of pronouncers based on tools for decision analysis in general terms. The plausibility of the design proposals for basic pronouncers given there are investigated in this chapter. Basic pronouncers based on SMILE, Netica, and ActiveDATA have been implemented, and are presented here together with a description of a general architecture for such pronouncers. The chapter also includes a brief discussion on more advanced implementation issues.

A brief discussion on why SMILE, Netica, and ActiveDATA should be selected for further testing was given in Section 3.5. There, Hugin, DELTA and Criterium DecisionPlus (WDObj) were also suggested, but no pronouncers have been implemented based on these tools. Hugin and Netica are very similar products, and testing one of them should be sufficient in an initial stage. Netica was selected because it was easier to get hold of. DELTA has already been judged as too slow for real-time use, although there are no actual figures supporting this judgment. Getting test results for DELTA would be interesting, but due to problems with the undocumented API and the lack of a function for loading models, no pronouncer based on DELTA was implemented. Finally, Criterium DecisionPlus is fundamentally different from the other tools in that it requires decision problems to be modeled as hierarchies of criteria. It is not obvious how this type of models could be used in the domain primarily aimed at with these tests—the RoboCup domain. For that reason, Criterium DecisionPlus has been left out.

5.2 Elements of Implementation

When implementing the three pronouncers, as much as possible of the code was shared. The main elements of the implementations are the following:

- communication with agents
- parsing and dispatching of incoming messages

- decision analysis functionality

The list is ordered from the most general element of the implementations to the most specific. Each element can be viewed as a layer (Figure 5.1), which only communicates with adjacent layers.

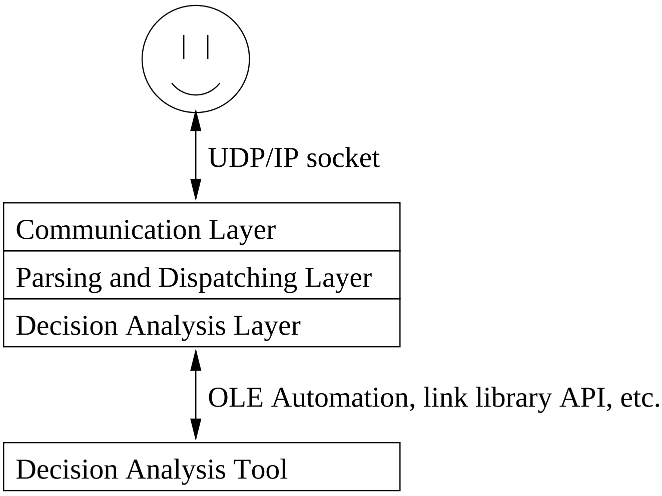


Figure 5.1: Architecture of implemented pronouncers.

5.2.1 Communication Layer

The pronouncers receive messages from agents through a UDP/IP socket (this is the type of communication used in the RoboCup simulation league). There are three types of commands that these messages can contain.

- load model
- set value
- get advice

These correspond to the three basic functionalities given in Section 4.2.1. When a message is received, it is immediately passed down to the parsing and dispatching layer for further processing. Once the message has been processed, a result is returned to the communication layer. Either this can be an indication of an error having occurred while parsing the message, or it can be a result that should be forwarded to the agent that sent the message. The latter occurs when a "get advice" message has been processed, in which case a recommended strategy is returned through the UDP/IP socket to the calling agent.

This layer only provides general functionality for communication. For that reason the implementation of the layer could be shared between all pronouncers.

5.2.2 Parsing and Dispatching Layer

In this layer, messages are parsed and then dispatched to the decision analysis layer. It works as a mediator between the decision analysis layer and the communication layer. Although the structure of some messages (e.g. the "load model" message) can be kept constant for all pronouncers, other messages (in particular the "set value" message) must reflect the underlying representation of decision problems used by a specific pronouncer. This implies that there must be at least one implementation of this layer for each representation of decision problems. If the decision analysis layer acts as a wrapper in the sense suggested in Section 4.3, one implementation of this layer for each type of problem representation is also sufficient.

5.2.3 Decision Analysis Layer

As stated above, the interface to this layer can be shared between pronouncers using the same underlying representation of decision problems. The implementation, however, must be specific to each pronouncer since each tool defines its own interface to the decision analysis functionality. What follows is a brief statement about the implementations of this layer in each of the three pronouncers.

SMILE

All the necessary functionality for a basic pronouncer is available in SMILE. The API is well documented, although some parts of the documentation feels a bit unorganized. The vast number of classes included in the API allows for many alternative ways of performing a certain task (e.g. setting probabilities in a chance node). This could be seen as a strength, but it also makes it harder to determine which way is the fastest—something that is essential when aiming for real-time domains. Moreover, the API is incongruent in the sense that utilities are set and retrieved in a slightly different way than probabilities.

The pronouncer based on SMILE was implemented for Solaris, and its size was about 678 kB. For Windows, SMILE is available as a dynamic link. This allows for substantial cuts in pronouncer size, since the largest part—the inference engine—is not statically linked with the pronouncer.

Netica

Netica API provides the three basic functionalities in a very straightforward way. The functions of the API are very easy to use, and exemplary documented. A free demo version of Netica was used when implementing the pronouncer. This version put a limit to the size of models, but was in other respects identical to the full version.

Two pronouncers based on Netica was implemented—one for Solaris, and one for Windows. The size of the Solaris version was 477 kB, which is about 70 percent the size of the SMILE-pronouncer. The Windows version was slightly smaller—363 kB.

ActiveDATA

ActiveDATA is still in beta-state, and some of the functions do not work properly. In the version used for this work, probabilities could for example not be set immediately. There were, however, work-arounds for this and a basic pronouncer could be implemented. Instead of setting probabilities and utilities immediately in a model, they are set through variables. This has its advantages, and is not only a work-around for bugs. By putting all variable definitions in the root node, probabilities and utilities can be set without traversing the tree. While this put constraints on how decision trees should be modeled, it can be justified by potential gains in performance.

Since ActiveDATA is an ActiveX control, and thus external to the program using it, the size of the pronouncer based on ActiveDATA was only 58 kB.

A drawback with ActiveDATA is that models from DATA cannot immediately be used. First a package has to be produced from the model. This is done with a special utility provided by TreeAge. When a package is produced, a package number is generated by the utility. With this number, a license for the package can be purchased from TreeAge with an accompanying ticket string. The ticket string is needed when opening the package with ActiveDATA. This system may be reasonable when a few large models are used, which probably is how DATA is traditionally being used. If ActiveDATA is to be used for real-time decision making, however, there will be many small models instead. Having to purchase tickets for all these models does not seem to be an attractive option.

5.2.4 Decision Modules

Figure 5.1 defines an architecture for pronouncers. In Section 4.4 it was stated that making the decision analysis functionality internal to the agent can sometimes be desirable. In order to achieve this, the agent can bypass the two upper layers in the original pronouncer architecture and instead communicate immediately with the Decision Analysis Layer. This alternative approach is depicted in Figure 5.2. The Decision Analysis Layer for all three implemented pronouncers provides an API consisting of C functions, and can be be statically linked with the agent as a decision module.

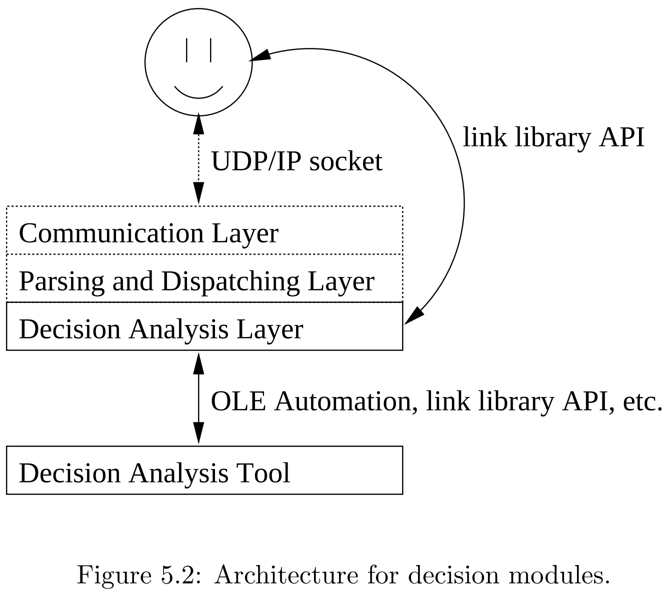


Figure 5.2: Architecture for decision modules.

A drawback with statically linking a decision module with the agent is that the agent then becomes considerably larger. The size of the implemented pronouncers range from 58 kB to 678 kB. If each agent in a multi-agent system is linked with an entity of that size, it can have a negative impact on the overall performance of the system.

5.3 Advanced Implementation Issues

A pronouncer should be able to handle calls from several agents concurrently. If models are shared among agents, then an agent must be able to send composite commands in one message. Otherwise their commands may come in conflict with one another. For example, agent A sets a probability in model M by sending a "set value" command, and then sends a "get advice" command to evaluate the model. If these two commands are sent separately, agent B might set another probability in model M—conflicting with A's beliefs—by sending a "set value" command between the two commands sent by A. In order to assert that no inappropriate changes are made by others, A must send the two commands as one atomic message.

A further consequence of using shared models is that agents must set all values in a model before evaluating it in order to assert that the model corresponds to the agent's beliefs. Unless the agent's beliefs change dramatically between each evaluation of the model, this will lead to loss in speed. As in Section 5.2.4, one must weigh the loss in speed against the savings in space. The pronouncers that have been implemented as part of this thesis work support composite commands. Currently, however, they only have one model loaded into memory at a time. Loading a model is a relatively time-consuming activity (due to disk access, memory allocation, etc.), so there is no good support for multiple models. It could easily be added by having some sort of cache for models that have been loaded, and this will probably be necessary if the pronouncers are to be used in any real applications.

Chapter 6

Performance of Pronouncers

6.1 Introduction

This chapter presents quantitative performance measures for the pronouncers that have been implemented. Although qualitative tests also need to be run before the usefulness of the pronounces can be determined, these results give a first indication on whether any of the pronouncers are fast enough for real-time decision making, and also if there are any notable differences between the different tools. First four test cases, used in the tests, are presented. This is followed by a description of the specific test procedure used, and the results from the tests.

6.2 Test Cases

For all the test cases used, the decision maker is an imagined soccer playing intelligent agent. When designing the decision problems, the RoboCup domain was kept in mind. There are two scenarios, each with one simple and one extended model of the decision problem. The models range from very simple (4 probabilities and 4 utilities) to more complex (52 probabilities and 12 utilities). It is hard to tell how realistic the models are, since they have not yet been utilized by any soccer playing agent. The models should, nevertheless, be sufficient to get a first indication on the performance (both relative and absolute) of the tools when used with models of varying complexity.

6.2.1 Scenario 1: Agent Cannot See Ball

In the first scenario, the agent must decide whether to move or to wait. The precondition is that the agent does not currently see the ball. The possible outcomes of the decision situation is that the ball either becomes visible to the agent, or remains out of sight.

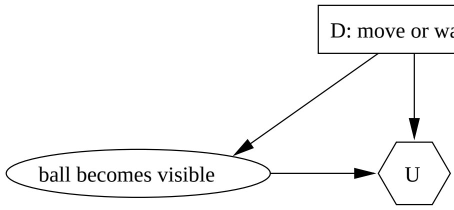


Figure 6.1: Influence diagram for decision problem "cantsee1".

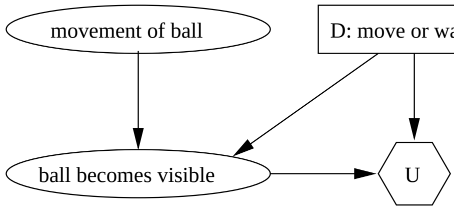


Figure 6.2: Influence diagram for decision problem "cantsee2".

Simple Model

In the simple model (Figure 6.1), there is only one chance node. This is the node representing the two possible outcomes, which contains a 2 by 2 conditional probability table. The utility node contains 4 values. No other information is explicitly modeled.

Extended Model

The extended model contains more explicit information about the decision situation. A node representing the agent's beliefs about the current movement of the ball relative to the agent has been added (Figure 6.2). The new chance node has two possible states—towards agent, and away from agent—and has a link to the node representing the outcomes of the decision problem. Consequently, this latter node now contains a 2 by 4 conditional probability table (compared to 2 by 2 in the simple model).

6.2.2 Scenario 2: Agent Has Ball

In the second scenario, the agent possesses the ball and needs to decide what to do with it. It can move with the ball, pass the ball, or just wait. For each action, there are four possible outcomes: the agent remains in possession of the ball, a fellow-player gets the ball, an opponent captures the ball, or no one gets the ball.

Simple Model

As in the first scenario, the simple model contains no explicit information about the decision situation other than the possible outcomes of the entire decision problem, the alternatives, and the utility function (Figure 6.3). The chance node

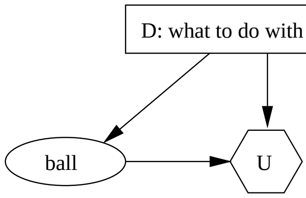


Figure 6.3: Influence diagram for decision problem “hasball1”.

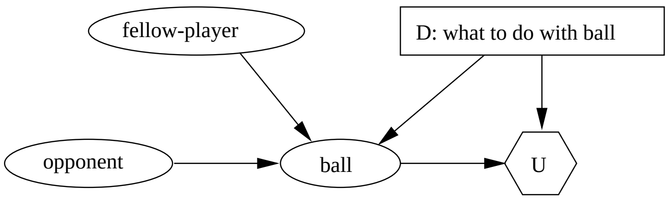


Figure 6.4: Influence diagram for decision problem “hasball2”.

representing the possible outcomes contains a 4 by 3 conditional probability table, and the utility node has 12 entries.

Extended Model

The extended model adds two new chance nodes—both influencing the outcome node. The new nodes represent the agent's beliefs about the relative position of a fellow-player and an opponent respectively, and have two possible states each: near agent, and far from agent. The additions has as effect that the conditional probability table in the outcome node grows to 4 by 12, i.e. it now contains 48 entries.

6.3 Test Procedure

The performance measure used in all tests was the time it takes to first set all values in a model, and then get an advice from the pronouncer. For each of the four test cases given above, 1000 set/evaluate-runs were performed and timed. All the probabilities and utilities were generated by a pseudo random number generator. The same seed was used for all test series so that all tools got to evaluate the same 1000 models.

6.4 Results

ActiveDATA only works on Windows platforms, and due to some problems with the Windows version of SMILE, the SMILE pronouncer was only implemented for Solaris. Consequently, the tests had to be run on two different platforms. Although this precludes direct comparison between ActiveDATA and SMILE, they can both be compared with Netica and thus indirectly with each other.

	Netica		SMILE	
	mean	*std.dev.*	*mean*	*std.dev.*
cantsee1	1.89	8.67	13.32	24.79
cantsee2	2.75	10.27	16.71	27.82
hasball1	2.64	11.29	18.63	29.07
hasball2	4.34	13.20	28.23	35.33

Table 6.1: Means and standard deviations (in milliseconds) for 1000 runs of each problem on Solaris.

	Netica		ActiveDATA	
	mean	*std.dev.*	*mean*	*std.dev.*
cantsee1	5.38	14.84	9.18	13.99
cantsee2	8.09	11.24	19.99	15.07
hasball1	8.05	12.13	28.07	15.40
hasball2	16.51	9.99	97.15	23.27

Table 6.2: Means and standard deviations (in milliseconds) for 1000 runs of each problem on Windows.

Results from the tests on the Solaris platform (a 167 MHz Sparc Ultra Creator) are given in Table 6.1. The Netica pronouncer outperforms the SMILE pronouncer by a wide margin, which can be seen more clearly in Figure 6.5. The ratio between the average running time for the Netica pronouncer and the average running time for the SMILE pronouncer stays almost constant for the four test cases. The Netica pronouncer outperforms the SMILE pronouncer by a factor 6 to 7 throughout the tests, which strongly suggests that the Netica pronouncer is the better of the two.

Also in absolute measures does the Netica pronouncer perform well. In the RoboCup domain, for example, an agent has about 30 milliseconds at its disposal for making a decision. The Netica pronouncer keeps below 5 milliseconds for all four test cases, and seems to adhere to the requirements of this particular real-time domain. Although the SMILE pronouncer in average keeps below 30 milliseconds for all test cases, it comes very close with the most complex model. Furthermore, the standard deviations suggest that the requirement is likely not to be met in a fair amount of the runs.

On the Windows platform (a Pentium 200 MHz), the Netica pronouncer keeps the same high performance. The ActiveDATA pronouncer is consistently slower than the Netica pronouncer. Results from the test runs are given in Table 6.2. It is noteworthy that while the ratio between Netica and SMILE was practically constant, ActiveDATA seems to have a greater performance degradation with a growth in the decision model to evaluate (Figure 6.6). If the same ratio between Netica and SMILE is assumed on the Windows platform as on the Solaris platform, ActiveDATA should outperform SMILE at least for the first three test cases. For the fourth ActiveDATA and SMILE should perform about equally well, and for more complex models, SMILE is likely to outperform ActiveDATA.

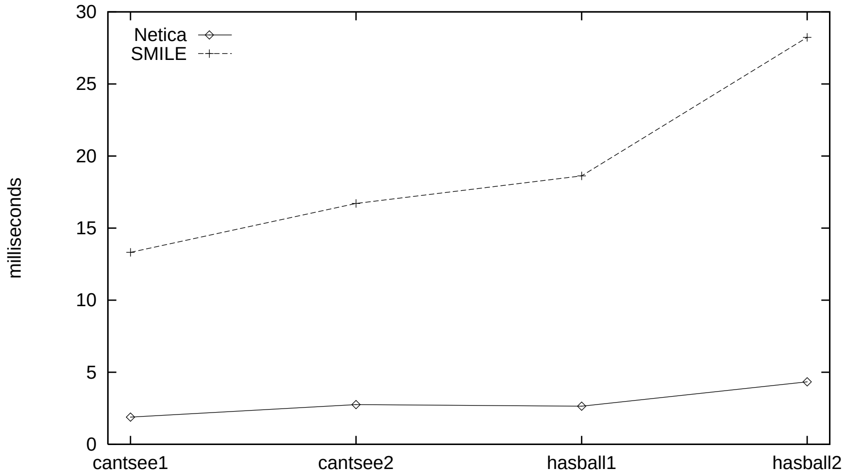


Figure 6.5: Performance on Solaris.

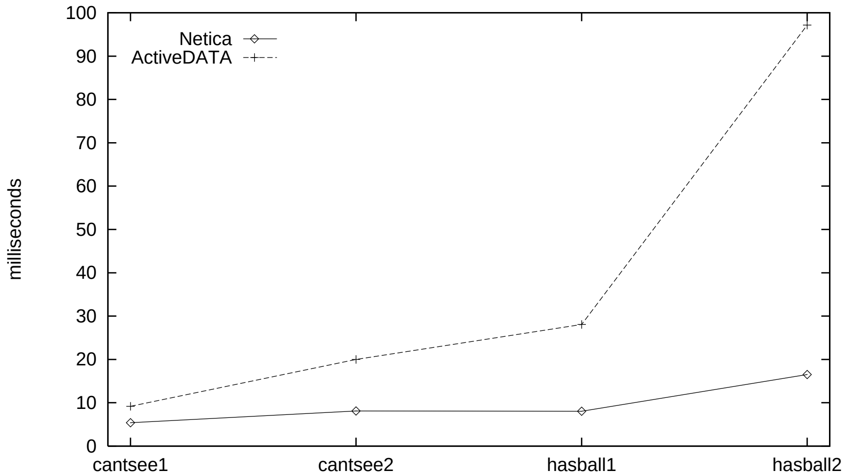


Figure 6.6: Performance on Windows.

Chapter 7

Discussion

7.1 Conclusions

There is no doubt that many of the currently available tools for decision analysis can be used for assisting intelligent agents in making decisions. The three so-called pronouncers, based on some of the tools, that have been implemented as part of this work show this emphatically. A framework for implementation of pronouncers based on an existing tool for decision analysis has also been presented. All this constitutes by itself a significant result, but the question then remains whether any of the tools are useful for decision analysis in real-time. The answer depends on what is understood by the term "real-time". In the RoboCup simulation league, each player has about 30 milliseconds at its disposal for making a decision. Other domains can inflict weaker or stronger time-constraints. The results that have been presented in this work give a first indication on how fast current tools are. Whether this is fast enough or not must be determined separately for each domain.

One of the tested tools—Netica—performs notably better than the other two. Not all the tools mentioned in Chapter 3 have been tested, and there are reasons to believe that at least Hugin's performance would be similar to that of Netica—perhaps even better. Still, in absolute measures, Netica seems sufficiently fast for use in the RoboCup domain. This alone is an important step towards applying formal decision theory to real-time decision problems.

7.2 Future Research

In [Russell and Norvig, 1995] (p. 843) it is stated that "as AI systems move into more complex domains, all problems will become real-time because the agent will never have long enough to solve the decision problem exactly". If Netica (or any of the other tools for that matter) at least to some extent can be used to perform exact decision analyses in real-time, it will be a result of great interest to the AI community. Although the results given by this work are promising, much remains before it can be accomplished. Even if a pronouncer is found to be fast enough in evaluating a model, a good utilization of the pronouncer is necessary. The correct models must be created, and it must be decided when to make calls to the pronouncer. These are by no means trivial tasks. This is a definite area for future research. Furthermore, the pronouncers implemented so far have been restricted to very basic functionality, like setting values and evaluating models. Many of the tools support additional functionality, and a future research project could be to evaluate the usefulness of some of the extended functionality suggested in Section 4.2.2.

Despite the promising results, it is too early to say if any of the tools surveyed in this thesis will prove to be useful in real-time domains. More tests with real decision situations have to be performed. What if all the tools turn out to be insufficient for real-time use? Then we need to ask what the reasons for the failures are. It might be that the techniques used are inherently slow, or it might only be that the particular implementations of the techniques have not been optimized for real-time applications. In the former case we must look for alternatives. One alternative is to use *anytime algorithms*. In [Horsch and Poole, 1998] an anytime algorithm for decision making under uncertainty is given which shows promising results for complex decision networks. Whether it is useful for smaller decision problems is not evident. Another anytime algorithm is mentioned in [Boman, 1997], but no results are given. In the latter case we can try to build tools for decision analysis more appropriate for real-time use. Work in this direction is currently being done with the DELTA tool [Boman and Verhagen, 1998].

Acknowledgements

I would first of all like to thank my advisor Magnus Boman for indispensable guidance and support. He has been available for discussions throughout the course of my work, and has continuously read and given insightful comments on drafts of my report. Another person I feel urged to acknowledge is Jeffrey Bellsey at TreeAge Software Inc. Without his devotion in helping me with all of the weird problems I ran into when trying to implement a pronouncer based ActiveDATA, I would have given up on ever getting it done. Many other people have also played a part in helping me accomplish my objectives, which only proves that alone we are mortal but together we become eternal.

Bibliography

[Boman and Verhagen, 1998] Magnus Boman and Harko Verhagen. Social intelligence as norm adaptation. In K. Dautenhahn and B. Edmonds, editors, *SAB'98 Workshop on Socially Situated Intelligence*, pages 17–25, Zürich, 1998. Technical Report, Centre for Policy Modelling.

[Boman *et al.*, 1998] Magnus Boman, Jens Andreasen, Mats Danielson, Carl-Gustaf Jansson, Johan Kummeneje, Johan Sikström, Harko Verhagen, and Håkan Younes. Ubu: Pronouncers in robocup teams. In Hiroaki Kitano, Gerald Seet, and K. Jagannathan, editors, *RoboCup Workshop, PRICAI'98*, pages 117–122, National University of Singapore, November 1998.

[Boman, 1997] Magnus Boman. Norms as constraints on real-time autonomous agent action. In Magnus Boman and Walter Van de Velde, editors, *Multi-Agent Rationality (Proc MAAMAW'97)*, pages 36–44, LNAI 1237, 1997. Springer-Verlag.

[Boman, in press] Magnus Boman. Norms in artificial decision making. AI and Law, in press.

[Bovin and Hammarberg, 1998] Mats Bovin and Marcus Hammarberg. Decision support for software agents. Master's thesis, Stockholm University, 1998.

[Chavez and Maes, 1996] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *PAAM 96 Conference*, pages 75–90, London, April 1996. The Practical Application Company.

[Danielson, 1997] Mats Danielson. *Computational Decision Analysis.* PhD thesis, Royal Institute of Technology and Stockholm University, May 1997.

[Dorais *et al.*, 1998] Gregory A. Dorais, R. Peter Bonasso, David Kortenkamp, Barney Pell, and Debra Schreckenghost. Adjustable autonomy for human-centered autonomous systems on mars. In *Mars Society Conference*, August 1998.

[Elofson and Robinson, 1998] Greg Elofson and William N. Robinson. Creating a custom mass-production channel on the internet. *Communications of the ACM*, 41(3):56–62, March 1998.

[Fishburn, 1982] Peter C. Fishburn. *The Foundations of Expected Utility.* D. Reidel Publishing Company, Dordrecht, Holland, 1982.

[Fou, 1997] Foundation for Intelligent Physical Agents, Geneva, Switzerland. *FIPA 97 Specification Part 3: Agent Software Integration*, October 1997.

[Genesereth and Ketchpel, 1994] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53,147, July 1994.

[Horsch and Poole, 1998] Michael C. Horsch and David Poole. An anytime algorithm for decision making under uncertainty. In *14th Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, July 1998.

[Howard and Matheson, 1984] Ronald A. Howard and James E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, California, 1984.

[Jensen, 1996] Finn V. Jensen. *An Introduction to Bayesian Networks.* UCL Press, London, 1996.

[Kitano, 1998] Hiroaki Kitano. *RoboCup-97: Robot Soccer World Cup I.* Springer-Verlag, Berlin, 1998.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2):157–224, 1988.

[Lockheed Martin, 1996] Lockheed martin autonomous control logic to guide unmanned underwater vehicle. `http://lmms.external.lmco.com/newsbureau/pressreleases/1996/9604.html`, April 1996.

[Malmnäs, 1994] Per-Erik Malmnäs. Towards a mechanization of real life decisions. In Dag Prawitz and Dag Westerståhl, editors, *Logic and Philosophy of Science in Uppsala.* Kluwer Academic Publishers, 1994.

[Malmnäs, 1995] Per-Erik Malmnäs. Methods of evaluation in supersoft decision theory. Unpublished manuscript, available on the WWW using URL: `http://www.dsv.su.se/DECIDE`, 1995.

[Pearl, 1986] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[Raiffa, 1968] Howard Raiffa. *Decision Analysis.* Addison-Wesley, Reading, Mass., 1968.

[Russell and Norvig, 1995] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, New Jersey, 1995.

[Saaty, 1990] Thomas L. Saaty. *Multicriteria Decision Making - The Analytic Hierarchy Process*, volume I of *the Analytic Hierarchy Process Series.* RWS Publications, Pittsburgh, PA, second edition, 1990.

[Shachter, 1986] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, 1986.

Appendix A

Internet Resources

Noetic Systems Inc.
`http://www.noeticsystems.com/`
Decision Systems Laboratory, University of Pittsburgh
`http://www.sis.pitt.edu/~dsl/`
Hugin Expert A/S
`http://www.hugin.dk/`
Norsys Software Corp.
`http://www.norsys.com/`
TreeAge Software, Inc.
`http://www.treeage.com/`
Applied Decision Analysis, Inc.
`http://www.adainc.com/`
Lumina Decision Systems
`http://www.lumina.com/`
Vanguard Software Corp.
`http://www.vanguardsw.com/`
DECIDE, Stockholm University/KTH
`http://www.dsv.su.se/DECIDE`
Expert Choice, Inc.
`http://www.expertchoice.com/`
InfoHarvest, Inc.
`http://www.infoharvest.com/`

Appendix B

Glossary

ActiveX control A Dynamic Link Library (DLL) providing functionality to other programs.

Anytime Algorithm An algorithm that can be halted at any time, and will then return a suboptimal solution to a problem. The longer it is allowed to run, the closer it gets to the optimal solution.

DDE Dynamic Data Exchange allows information to be shared or communicated between programs.

OLE Provides a standard conceptual framework for creating, managing, and accessing object-based components that provide services to other objects and applications.

OLE Automation The Microsoft standard for communication between Windows applications. OLE Automation allows one Windows application to control the objects exposed by another application.