ISAC:
a Case-Based Reasoning System for Aircraft Conflict Resolution

Andrea Bonzano

A thesis submitted to the University of Dublin,
Trinity College, for the degree of
Doctor in Philosophy

April 1998

Declaration

The work described in this thesis is, except where otherwise stated, entirely that of the author and has not been submitted as an exercise for a degree at this or any other university.

Signed:
Andrea Bonzano

_____

April 1998

Permission to Lend or Copy

I agree that Trinity College Library may lend or copy this thesis upon request.

Signed:

Andrea Bonzano

_____

April 1998

Acknowledgements

I would like to express my sincere thanks and gratitude to my supervisor Dr. Pádraig Cunningham for his involvement in this research, for the technical discussions and particularly for his support and friendship throughout the course of my Ph.D. studies.

This research could not have proceeded without the help and support of Dr. Colin Meckiff in Eurocontrol Experimental Centre, during the period I spent in Paris.

I am grateful to Kathleen Hanney and Barry Smyth for insightful discussions and advice in CBR and Mark Keane for discussions on how to evaluate the system.

For the technical support, thanks to all the wonderful people in the Department of Computer Science in Trinity College Dublin, Phil Gibbs, Steve Owen and Werner Goettlinger in Eurocontrol and Luca Di Taranto from far Italy.

I am also very grateful to everybody of the Artificial Intelligence Group for creating an environment where it is a delight to work, and to Michelle, Conor, John, Ronan, Shane and Shaw who were “obliged” to proof read this thesis.

I would like to express my sincere gratitude to the controllers of the Eurocontrol Experimental Centre: Andrew Barff, Peter Csarnoy, Ray Dowdall, Frank Dowling, Peter Eriksen, Robin Hill, Diarmuid Houlihan, Paul Humphreys, Roger Lane, Leif Lundquist, Rod McGregor, Hugh O’Connors, John O’Gorman (Dublin Airport), Guy Tod, Nigel S. Thorne, Michael Weldon (Irish Aviation Authority), Paul Zabka and especially to Nigel Makins.

Finally, for friendship and moral support I want to thank Jimmy, with whom I enjoyed all these years in Dublin, Luca, Gaio, Albillo, Chris, Louise and everyone else who made these three years really enjoyable.

This thesis is dedicated to my parents and my sister Camilla.

Summary

Case-Based Reasoning (CBR) has emerged from research in cognitive psychology as a model of human memory and remembering. It has been embraced by researchers of AI applications as a methodology that avoids some of the knowledge acquisition and reasoning problems that occur with other methods for developing knowledge-based systems.

Previous attempts to use Artificial Intelligence in Air Traffic Control (ATC) have never attained the level of confidence necessary for controllers to effectively use it in the real world. This lack of success is due in large measure to knowledge engineering difficulties in modelling ATC decision making. In this thesis we describe the successful application of case-based reasoning to this problem. We describe what was required to make CBR work and asses the knowledge engineering impact of CBR. The novelty of the approach presented in this thesis is in the manner that artificial intelligence is used as an intelligent assistant rather than an expert system, and in the technique used, which is CBR instead of the standard rule-based systems (RBS).

The acronym ISAC stands for Intelligent System for Aircraft Conflict Resolution. It is a CBR system that helps air traffic controllers to solve conflicts between sets of aircraft. The three stages of the decision making process for conflict resolution are: selection of the aircraft to manoeuvre, deciding on the type of manoeuvre and specifying of the details of the manoeuvre.

ISAC assists the controllers in the first two stages of this decision process. ISAC is interesting in itself because of the critical safety issues involved and because of the question of what constitutes a case in this problem domain.

Several issues were encountered during the development of ISAC. The most interesting ones, that constitute the main contribution of this thesis, are:

- the analysis of the knowledge engineering problem;
- the use of a hierarchical case-based reasoning structure;
- the issues of case reuse and case representation;
- the analysis of the discriminatory power of the case parameters.

Table of Contents

List of Figures

Glossary

AI	Artificial Intelligence
ATC	Air Traffic Control
BADA	Base of Aircraft DAta
BDM	Bad Down Matching
BUU	Bad Up Unmatching
CBR	Case-Based Reasoning
GDU	Good Down Unmatching
GHMI	Ground Human-Machine Interface
GUM	Good Up Matching
HIPS	Highly Interactive Problem Solver
KBS	Knowledge-Based System
KE	Knowledge Engineering
IL	Introspective Learning
ISAC	Intelligent System for Aircraft Conflict Resolution
MAC	Multiple Aircraft Conflict
NN	Nearest Neighbour
RBS	Rule-Based System
TAC	Two Aircraft Conflict
TMA	TerMinal control Area
TOD	Top Of Descent

Associated Publications

An Incremental Retrieval Mechanism for Case-Based Electronic Fault Diagnosis, Cunningham P., Smyth B., Bonzano, A., to be published in the Knowledge-Based Systems Journal, January 1998.

Learning feature weights for CBR: Global vs. Local, Bonzano A., Cunningham P., Smyth B., in Proceedings of the 1997 Conference of the Italian Association of Artificial Intelligence, Springer Verlag Lecture Notes in Artificial Intelligence, September 1997, pp.417-426.

Using introspective learning to improve retrieval in CBR: a case study in air traffic control, Bonzano A., Cunningham P., Smyth B., in Proceedings of the 1997 International Conference on Case-Based Reasoning, Springer Verlag Lecture Notes in Artificial Intelligence, July 1997, pp.291-302.

ISAC: a CBR system for decision support in air traffic control, Bonzano A., Cunningham P., Meckiff C., in Proceedings of the 1996 European Workshop on Case-Based Reasoning, Springer Verlag Lecture Notes in Artificial Intelligence, November 1996, pp.44-57.

A review of CBR for use in Air Traffic Control, Bonzano A., Cunningham P., EEC Internal Report, April 1997.

An Incremental Case Retrieval Mechanism for Diagnosis, Cunningham P., Bonzano A., Smyth B., Technical Report, TCD-CS-95-01, Dept. of Computer Science, Trinity College Dublin.

Chapter 1
Introduction[1]

Despite the fact that modern aircraft are packed with sophisticated electronic equipment, air traffic control (ATC) has always been more of an art than a science. Ground-based control essentially consists of people following the progress of aircraft represented by points derived from radar data and displayed on a flat display screen. The simple nature of the data available means that the controllers themselves are required to build and maintain a mental picture of extrapolated 4D traffic based on experience and other rather ill-defined heuristics. Having done this, the controller must mentally compare every pair of predicted trajectories to determine whether any pair of aircraft will pass within the minimum permitted separation - in which case he is required to intervene in some way to resolve the potential conflict.

Such an unscientific approach to ATC is, however, becoming less and less acceptable. Pressure for change is coming from two sources: firstly, the ATC world, as elsewhere, is undergoing an information explosion - controllers potentially have access to gigabytes of data of every sort, and have the possibility to communicate with aircraft and other ground systems in ways, and at speeds, which were unimaginable when their practices were conceived. Secondly, airlines are demanding greater efficiency and quality of service from the air traffic control providers: efficiency, because ATC currently accounts for about 15% of the price of a ticket, and quality of service to allow airlines to increasingly fly their preferred and presumably near-optimal flight paths.

The problem cannot be approached from a uniquely technical viewpoint. Removal of the “artisanal” aspects of ATC, particularly with regard to the task of preventing contact between aircraft, touches the very heart of the profession. This, therefore, means that any enhancement of the controller’s skills with automation must be done in a way which is sympathetic to current practices and therefore acceptable to controllers.

[1] This Ph.D. research has been funded by Eurocontrol Experimental Centre in Paris, the European Centre for Air Traffic Control.

Previous attempts to use Artificial Intelligence in ATC have never attained the level of confidence necessary for controllers to effectively use it in the real world. This lack of success is due in large measure to knowledge engineering difficulties in modelling ATC decision making. In this thesis we describe the successful application of case-based reasoning (CBR) to this problem. We describe what was required to make CBR work and asses the knowledge engineering impact of CBR. The novelty of the approach presented in this thesis is in the function of artificial intelligence used as an intelligent assistant more than an expert system, and in the technique used, which is CBR instead of the standard rule-based systems (RBS).

1.1 Case-Based Reasoning

"Case-based reasoning means reasoning based on previous cases or experiences. A case-based reasoner uses remembered cases to suggest a means of solving a new problem, to suggest how to adapt a solution that does not quite work, to warn of possible failures, to interpret a new situation, to critique a solution in progress, or to focus attention on some part of a situation or problem" (Leake, 1996).

The CBR cycle rarely occurs without human intervention. For example many CBR tools act primarily as case retrieval and reuse systems. Case revision, i.e. adaptation, is often undertaken by human managers of the case-base. This should not be viewed as a weakness of CBR but as an encouragement for human collaboration in decision support (Watson, 1994).

CBR is a step ahead of the traditional RBS. The early systems, like DENDRAL, MYCIN and PROSPECTOR, all operated in domains where there were good underlying models. Unfortunately, in a commercial environment and outside of the Universities, many people make decisions without reference to first principles and underlying causal or statistical models. These people solve problems by using their experience (Watson, 1996).

CBR makes it possible to give solutions even if the domain is open-ended or ill-defined (Leake, 1996). This seems to be one of the characteristic of ATC. Usually a controller solves a conflict by referring to situations that he has already seen. Moreover, training on a specific sector is essential to get used to the environment and more importantly to learn the patterns of traffic that should automatically trigger the solution. For these and other reasons, ATC seems to be a suitable domain for the application of CBR.

However the represent-retrieve-reuse model of CBR is often difficult to apply even in situations where human competence is obviously reuse-based. This difficulty is almost

always associated with the granularity of retrieval and the question of what constitutes a case leads to the knowledge engineering problem.

1.2 Expert Systems versus Intelligent Agents

An expert system is a computer program that has the same competence as a human expert. Moreover, it can increase its expertise on the domain and update its knowledge base while in use. Expert systems are often used for the resolution of problems, for planning and for design.

It should be pointed out that an expert system, like the majority of artificial intelligence systems, is competent only in the domain that it has been taught. An expert system competent in the ATC domain does not necessarily have to be competent in any other domain. This is the purpose of artificial intelligence: finding algorithms to build computer programs that can learn and apply the acquired knowledge, and not the commonly perceived notion of building generic thinking machines. Deep Blue, the program that beat Kasparov can be considered an artificial intelligence application specialised in the chess domain. Criticisms of the type: “it beat Kasparov but it cannot talk” show that people still have not understood the purpose of artificial intelligence. If people want to talk about thinking machines, it is to cognitive science and not artificial intelligence that they should refer. Artificial intelligence provides algorithms to cognitive scientists, but the domains are different. In AI the performance is essential whereas in cognitive science, the imitation of the brain is the main issue.

Lately, a new concept has appeared in the AI domain, the concept of Intelligent Agents. An expert system tends to act as a substitute for humans whereas an intelligent agent helps and co-operates with the human (Maes, 1994). In ATC it is *not* possible to substitute controllers first of all for safety reasons, but for legal reasons, too.

If an expert system which is in charge of a production line makes an error, the worst thing that can happen will result in a loss of time and money. Even if not desirable, this is acceptable and the occasional loss of money is compensated by the savings that the computerised system offers. Whenever an error from the expert system could cause either injuries or loss of lives, its use must be considered very carefully. ATC is one of those domains: there must always be a human to take the responsibility for the decisions taken. But this human can be helped in making decisions by an intelligent agent. The use of an intelligent agent will not only reduce the controller’s workload, but also reduce human errors and biases (Kitano, 1996). Typical of an intelligent agent is the possibility of

introducing thresholds that indicate how confident the system is about the solution that it is presenting. The two thresholds usually present in an intelligent agent are called: “do-it” and “tell-me” thresholds (Maes, 1994). Above the “do-it” threshold the agent automatically executes an action without asking the user. Between the “do-it” and the “tell-me” thresholds the agent gives a suggestion that is usually correct and below the “tell-me” threshold the agent does not know what to do. This oncept, even if slightly modified, has been used in the construction of our system[2].

1.3 ISAC

The acronym ISAC stands for Intelligent System for Aircraft Conflict Resolution. It is a CBR system that helps air traffic controllers to solve conflicts between sets of aircraft. The three stages of the decision making process for conflict resolution are:

- selection of the aircraft to manoeuvre,
- decision on the type of manoeuvre and
- specification of the details of the manoeuvre.

The choices made depend on several factors: the geometry of the conflict, the capabilities of the aircraft, their position relative to the destination, etc. ISAC is an intelligent agent that assists the controllers in the first two stages of this decision process.

The advantages of early conflict prediction and resolution are the reduction of the controller’s workload, relaxation of ATC restrictions and the possibility of having more aircraft flying with a direct route and at preferred altitude profiles. It means that, with the same constraints, both the controllers and the pilots will be more satisfied (Shively and Schwamb, 1994).

In ISAC we introduce a new threshold, called “don’t do” threshold, which is coded inside the system: if the similarity between the case and the target is below the “don’t do” threshold, the system does not suggest any solution. The “tell-me” threshold is not in ISAC’s code, but it is up to the controller to decide whether the solution is acceptable (above the “tell-me” threshold) or not acceptable (below the threshold). This means that the “tell-me” threshold, even if not explicitly stated, is implicitly used by the controller.

Finally, ISAC is interesting in itself because of the critical safety issues involved and because of the question of what constitutes a case in this problem domain.

---

[2] ISAC is considered an intelligent agent even if it does not operate autonomously, which is a characteristic common to several intelligent agents.

1.4 Contributions of this Thesis

Several issues were encountered during the development of ISAC. The most interesting ones, that constitute the main contribution of ISAC, are:

- the analysis of the knowledge engineering problem;
- the suggestion of a hierarchical case-based reasoning structure;
- the issues of case reuse and case representation;
- the analysis of the discriminatory power of the case parameters.

These points, explained below, will be treated in more detail in the next chapters.

1.4.1 The Knowledge Engineering Problem

The Knowledge Engineering (KE) problem is not always treated in the intelligent agents or expert systems literature because often the databases used for the evaluation of these systems are toy-databases, i.e. databases that have been created especially with the purpose of testing that particular system or databases that are already available. This was not the case for ISAC, a system that had to be built to solve a real world problem widely known for its complexity. This means that the power of ISAC is mainly in its database and in the parameters used to describe it.

All the steps of the KE process are described explicitly in Chapter 6 and implicitly in all the thesis: the understanding of the domain, the definition and acquisition of parameters, the different approaches to important CBR issues and successive changes of direction, the acquisition of the data with solutions for the construction of the case-base etc. are all different aspects of the knowledge engineering problem. The last issue, i.e. the construction of the case-base, has probably been one of the most problematic. The availability of the flight plans of all the aircraft flying above Europe means that it is possible to easily extract a lot of conflict descriptions. The problem is that these conflicts, to be stored in a case-base, need a solution that has to be given by a controller, an operation that requires a lot of time. This bottleneck shifted the focus from the effective acquisition of the case-base to the development of a hierarchical structure and a different case representation.

1.4.2 The Hierarchical Structure

The need of solving multiple aircraft conflicts inspired the hierarchical structure. A multiple aircraft conflict can involve 3 or more aircraft. It would be too difficult to build different case-bases for three aircraft conflicts, four aircraft conflicts etc. For this reason, the conflict has to be decomposed into two aircraft conflicts, but some high-level analysis has to be

applied because the solution to a multiple aircraft conflict is *not* necessarily one of the solutions of the component two aircraft conflicts. A hierarchical structure would allow ISAC to use the same case-base for both two aircraft conflicts and multiple aircraft conflicts with big savings in space and time.

1.4.3 Case Representation

The choice of the structure for a case is not obvious. A case could contain the description of all the aircraft involved in a conflict or, alternatively, for each aircraft involved in the conflict a new case could be created. While the first choice is more intuitive and closer to the way the controllers think, the second one is more extendible. Having one case for each aircraft facilitates the generalisation of the case-base to multiple aircraft conflicts because the same cases containing one aircraft conflicts could be used for solving both two aircraft or multiple aircraft conflicts. The problem with this case structure is that, by splitting the conflict into two separate cases, there is the risk of loss of information.

Please note that two types of case reuse have been mentioned: case reuse with a hierarchical structure and case reuse with the case representation. Those are two different approaches to the same problem. The hierarchical structure reuses two aircraft conflicts for solving multiple aircraft conflicts, independently of the structure used to represent a case. A different approach is to change the case structure with the purpose of reusing each single aircraft description in any type of conflict. Both the approaches have been developed.

Having all the aircraft described in the same case gives rise to the problem of deciding the order in which the aircraft are described. For this purpose either all the combinations of the aircraft could be stored as independent cases or a “canonical”, i.e. standard, form has to be found. Again, both these approaches have been developed.

1.4.4 Discriminatory Power of the Parameters

Not necessarily all the parameters that describe a case have the same importance. Using decision trees or calculating directly the discriminatory power of the parameters is a way of better understanding the case-base under construction. Those two methods simply indicate what are the most important parameters in the case description and could be useful to purge some useless parameters or to better specify very important parameters in the case description.

The task of effectively finding the weight of a parameter is quite difficult if it is up to a human expert. A lot can be learnt about what parameters are important for retrieval by

comparing similar cases in a case-base. It can be automatically determined which parameters are necessary in predicting outcomes and weights to parameters can be assigned accordingly. In the same manner it can be discovered which parameters are used in specific contexts and determine localised parameter weights that are specific to individual cases. The property of a parameter changing weight depending on the value of other parameters is called context sensitivity.

1.5 Summary and Structure of this Thesis

In brief: the next two chapters deal with ATC and CBR. After a chapter with the technical description of ISAC, the mainly theoretical chapter treating all the CBR issues is presented. Then the chapter on knowledge engineering shows the process of building the system, the chapter on introspective learning analyses this technique and finally the evaluation of the system and the conclusion chapters judge whether the system has been successful in marrying CBR and ATC.

In more detail: Chapter 2 introduces the reader to the basic concepts of air traffic control and to HIPS which is a computer aided tool that helps the controller in the visualisation and resolution of a conflict. The approach of other intelligent systems to the problem of ATC is analysed and the points that could be useful for ISAC are highlighted.

Chapter 3 gives the background to case-based reasoning. The steps that constitute the typical CBR system are explained, then a prototypical example is described. The related literature is analysed to raise the CBR issues that will be treated in Chapter 5.

In Chapter 4 some technical issues like the interface between ISAC and HIPS, the structure of the system and of the web of pointers used during the retrieval process are treated. The parameter acquisition and the changes adopted during the knowledge engineering process are justified. Of all the choices presented in Chapter 3, the most suitable for ISAC are explained in Chapter 5. Issues like the case representation, the case structure, the coverage of the case space with the alternative between a few gold standard cases and lots of noisy cases, the possible solutions and the “don’t care” values are treated. Talking about the importance of the parameters: the discriminatory power and the decision trees are treated in Chapter 5 whereas Introspective Learning techniques will be analysed in Chapter 7. Finally, the hierarchical structure for the resolution of multiple aircraft conflicts is analysed here, applied in Chapter 6 and evaluated in Chapter 8.

The knowledge engineering problem is discussed in Chapter 6 with all the steps done to build a realistic and robust system that could satisfy the controller’s needs.

In Chapter 7 a comprehensive set of techniques for learning local and global parameter weights are described. These techniques are evaluated on the ATC case-base and with other case-bases. It is shown how introspective learning of parameter weights improves retrieval and how it can be used to determine context sensitive local weights. Introspective learning does not work well in case-bases containing only pivotal cases because there is no redundancy to be exploited. It is shown that local weights are better than global weights in the ATC domain and which update policies are most effective. Finally it is discussed how the overfitting problem, common in introspective learning, can be avoided.

The evaluation of the different versions of the systems is in Chapter 8. The possible ways of evaluating are analysed: "LeaveOneIN", "LeaveOneOUT"[3] and real test with traffic samples from real time simulations. Some general conclusions on the performance of the system and on the applicability of CBR to ATC are here and in Chapter 9. Figure 1.1 gives the plan of the thesis.

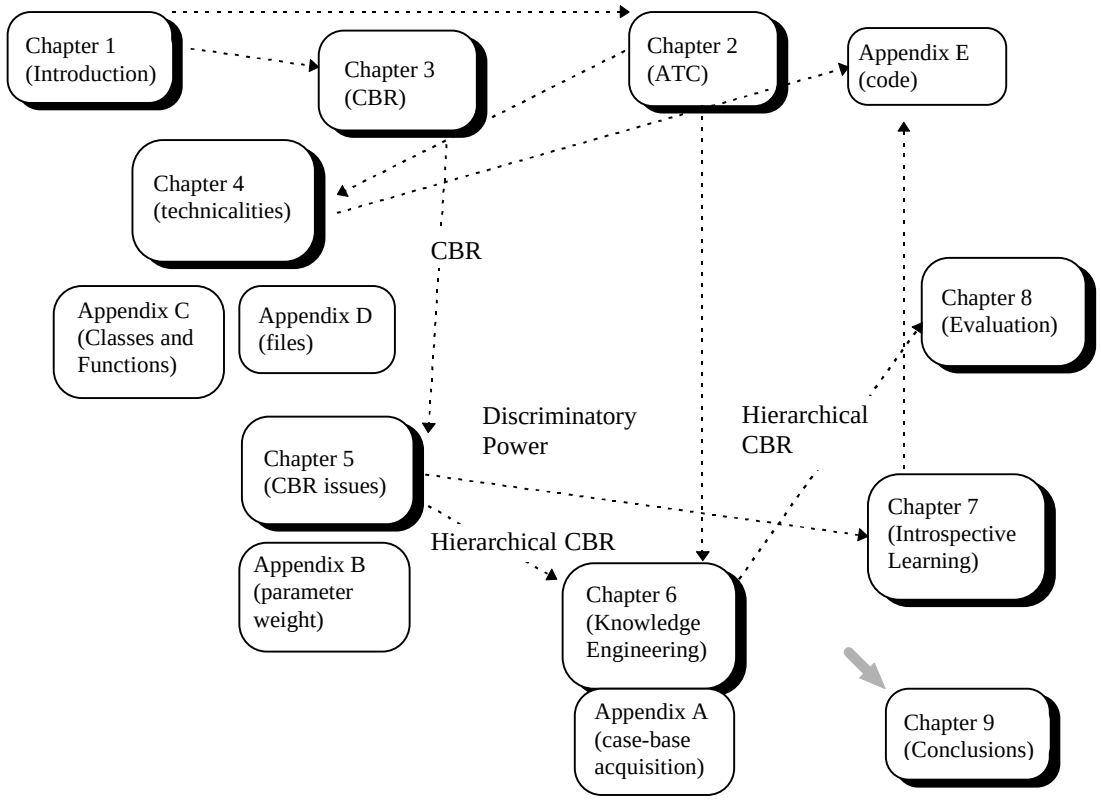

Figure 1.1: *Dependencies of the chapters.*

[3] In a "LeaveOneIN" simulation all the cases in the case-base are used as a target, but the target is left in the case-base. When the case used as a target is taken out of the case-base the simulation is called "LeaveOneOUT".

Chapter 2

Air Traffic Control

2.1 The Problem of Air Traffic Control

Traditionally, the stated objective of air traffic control is the safe, orderly and expeditious flow of air traffic. Nowadays, it is necessary to add that air traffic control should be impartial, cost effective, noise abating and fuel conserving. Current and future air traffic control systems must meet these additional requirements without any sacrifice of the vital essential safety, orderliness and expedition (Wiener and Nagel, 1988). This point of view is the same as Shively and Schwamb in AIRPAC (1984): the solution to a conflict must provide the minimum separation, achieve fuel efficiency, minimise the number of commands and minimise the delay.

These conditions have to be respected by the controllers and obviously by any system that tries to help them. Systems thinking is very similar to Human Factors thinking: it often requires a statement of the obvious and to look outside the lines. The problem is that if the obvious is stated people consider this simplistic and "obvious", if the obvious is not stated it is often missed and some expensive design errors can be made that only become apparent when an accident occurs. When designing any ATC component, apart from the technical aspects and feasibility, there is the need to examine the wider systems perspective to fully understand its value and impact. An example can make this need more clear.

The following report of a flight on an Airbus 340 confirms what is being reported elsewhere, i.e., A340s have a problem fitting into a congested organised track system, like NOPAC (North Pacific Track structure), dominated by faster aircraft such as Boeing 747s, because of their cruise speed.

> The trouble began on the ground. The scheduled departure time was 5:15 p.m., but we had to wait for a clearance because the ATC wanted to get faster traffic out ahead of us. At 5:35, ATC told us they could give us a clearance up to flight level (FL) 280 then or we could get the FL340 we wanted in about half an hour, so we took FL280, and took off 20 min behind schedule.
>
> After about an hour, we were cleared to go up to FL300, but still not FL340 we wanted. The reason is two B747s were trailing us, and they had been given higher altitudes. The flight

> crew kept requesting higher altitudes, which they did not get because the ATC would not place a slower airplane (A340 at Mach 0.82) ahead of faster ones (B747s at 0.85). The crew spent the first half of the flight trying to figure out a way to get up to FL340. Finally they gave up and called their dispatch office and got a new flight plan.
>
> When we landed, the actual fuel on board was 9 tons versus 10.4 according to the original flight plan. Since the winds were on the mark, the cost of flying at the lower altitude was about 1.4 tons of extra fuel burn.

Thus we have an aircraft that may well be technically excellent and very economical to operate, but in densely utilised airspace it will not be easy to achieve those economies because of the wider system requirements. This is an example of how not fully applying a systems view can lead to inefficiencies that probably appeared outside the scope of the designers brief.

2.2 Principles of ATC

Commercial aircraft are controlled by ground-based air traffic controllers from the moment the engines are started at the origin of the flight to the moment the engines are stopped at the destination (Field, 1985). To facilitate the control task once the aircraft is en-route, the airspace is divided into horizontally and vertically bounded sectors, each sector normally being the responsibility of two controllers. The size of a sector depends on the amount of traffic to be processed, the number of aircraft per hour normally being limited to around 30. This means that in areas of high traffic density the sectors will tend to be smaller giving an average transit time of around 6 or 7 minutes, whereas in low density areas with larger sectors, transit time can be around 20 minutes.

Sector capability is an indicator of how busy a sector is, but it is difficult to calculate it correctly. The capability of a sector used to indicate the maximum number of aircraft that can enter a sector in one hour. This measure was too vague because the aircraft could arrive at regular intervals or could enter the sector at the same time. More sophisticated measures consider not only the number of aircraft that entered a sector but also the number of aircraft who exited it. In this case the capability of a sector is the maximum number of aircraft that can be in the sector *at the same time*.

Apart from national boundaries, the shape of the sector is normally a function of route structure, a sort of road system in the sky normally followed by commercial aircraft. The route structure has been designed so that major route crossing points do not occur near the edges of the sectors to avoid co-ordination problems.

In Europe, aircraft going Northbound, i.e. with heading from 0° to 180°, fly at odd levels, i.e. 25, 27, 29 thousand ft, whereas aircraft going Southbound fly at even levels. In France the separation is between Eastbound and Westbound aircraft because the majority of the traffic through France is Northbound and Southbound.

Unidirectional airways are an exception to this rule. An example of unidirectional airway is between London and Paris where 3 levels one above the other are used for aircraft going in the same direction, making it easier for the controllers to change level to an aircraft without crossing trajectories of aircraft going the other direction.

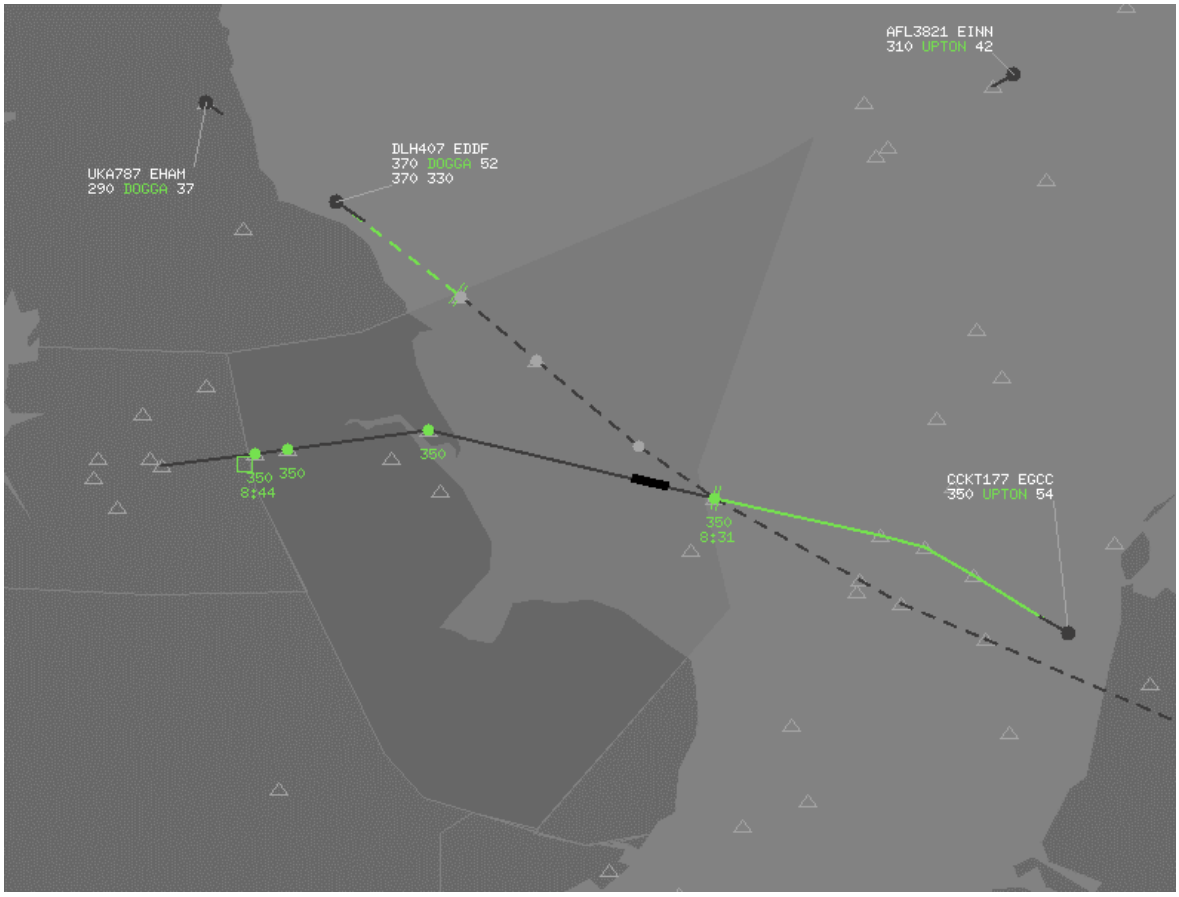

Figure 2.1: *The radar screen.*

Figure 2.1 shows a typical radar screen: the sector under examination (above the North Sea) is in a darker grey. Two aircraft with their trajectories are shown. The darker segment indicates a loss of separation between the two aircraft DLH407 and CCK177.

When an aircraft is about to enter a sector, the controller responsible for that sector is notified of its arrival, and this should correspond more or less with its appearance on the radar display. A short time later the controller assumes responsibility for the aircraft, a complementary release of responsibility having taken place in the upstream sector. The

bilateral agreement of the conditions for transfer from one sector to another is known as co-ordination, and actually represents a substantial part of the controller's workload.

It is then up to the controllers to see the flight through the sector and clearly the main concern is that the aircraft transits the sector conflict-free. There is however a secondary requirement which is to provide the aircraft with a cost and time-efficient passage.

A controller needs a licence specific to the sector to work on it. This, with the fact that the licence is not valid if not used for six months, shows how important the training on a particular sector is.

2.2.1 Types of Airspace

When a commercial aircraft takes off, the planning of the trajectory has already been done. People tend to think of the control tower as the normal air traffic control workspace. In fact, only those controllers handling air traffic in the immediate vicinity of the airport have a direct view of the air traffic; most have no outside view at all. There are three types of commercial airspace: en-route sectors, TerMinal control Area (TMA) and tower airspace. There are 65 ATC centres in Europe and 400 in the USA that control the non-military airspace. Military airspace is not considered in this work because of the different procedures and priorities in use.

Traffic usually gets into an en-route sector already cruising and usually exits still cruising. There are few level changes because sector exit levels have to be achieved and there are few aircraft in evolution. When there is a conflict, the best manoeuvre would be a slight horizontal turn, usually no more than 10-15°. When the controller is not too busy, an aircraft can be put on purpose on a wrong level which is called Opposite Direction Level (ODL). The workload cannot be too high because an aircraft in an ODL has to be monitored by the controller. For example, above Ireland each morning there is a flow of aircraft going from the USA to London, but not the other way around: the unused westbound levels are reserved for eastbound traffic and the following sectors are ready to accept aircraft at ODL.

TMA sectors are around airports and they are limited by a maximum flight level. Approach controllers decide the arrival sequence and it is up to them to decide to have parallel landings. In a TMA sector there are a lot of aircraft in evolution. A radar, i.e. horizontal, solution is often used and an horizontal turn can be up to 60°. Because there are a lot of aircraft in a restricted space, there are a lot of opposite direction conflicts and the turbulence effect is even more important because the aircraft are close to each other. Speed

differential could be useful for sequencing but not for conflict resolution. In general, en-route sectors are simpler to control than TMA sectors.

Finally, there are the tower sectors that surround aerodromes. Separation is often kept with direct sight, without the need of the radar. The separation between aircraft depends on their type because each type generates a different turbulence. Usually there must be three minutes between two aircraft landing or taking off. In Heathrow, where the traffic is more intense, the separation has been reduced to two minutes.

2.3 Conflicts and Conflict Resolution

Internationally agreed rules exist defining separation standards below which aircraft are said to be "in conflict". The values of these separations vary according to a number of factors such as the type of controlled airspace. Minimum horizontal separations are typically 5 nautical miles (1nm = 1852m) in radar controlled regions and either 1000 ft or 2000 ft vertically, depending on altitude. In areas not covered by the radar the horizontal separation is bigger, reaching even 40 nautical miles like in Turkey or 120 nautical miles like in Iran, i.e. 15 minutes of horizontal separation. In (ICAO,1994 and ICAO,1996) the rules of the air and air traffic services are explained and standardised.

Note that "conflict" is not synonymous with "collision" but is rather the infringement of the applicable separation minima as can be read from the Daily Telegraph, August 1997:

> Two British Airways jets, carrying more than 300 passengers, came within seconds of a mid-air collision because of an error by an air traffic controller. The near-miss happened as the two Boeing 757s were in a holding pattern awaiting permission to begin their approaches to Heathrow.
>
> One, flying from Paris with 165 passengers, was told to descend from 11,000 ft. It reached 10,400 ft before the crew realised that the second aircraft, at 10,000 ft, was maintaining its altitude. The pilot of the higher plane quickly levelled out and turned sharply away from the other, which was carrying 150 passengers from Geneva, before returning to his previous altitude. The Civil Aviation Authority inquiry established that the emergency had occurred because the controller handling the two flights had "inexplicably" issued the descent instruction to the wrong aircraft.
>
> At the time, the flights were so close that their altitude and flight number data on the controller's radar screen were overlapping and "virtually indecipherable". By the time the control centre's "conflict alert" sounded, the descending pilot had already taken evasive action. The captain of the Paris flight told investigators that if his plane had been fitted with automatic collision avoidance systems, he would never have begun the descent. Such equipment, called TCAS, will not be compulsory in Europe until the year 2000.

The incident took place in November 1996, a week after the world's worst mid-air crash when a Saudi Arabian jumbo collided with a Kazakh cargo plane near Delhi. Almost 350 died.

In practice controllers will often apply separations significantly larger than 5 nautical miles or 1000 ft, mainly due to the difficulties they have in accurately visualising future trajectories and conflict situations. This has a number of implications: for example a manoeuvre applied to resolve a conflict may end up significantly larger than is actually necessary (i.e. non-optimal) and indeed, there will often be unnecessary intervention where, had the aircraft continued on their existing trajectories, there would not actually have been a loss of separation.

One of the most important advances in computer support for air traffic controllers in the next few years will be the provision of relatively accurate predictions of future aircraft trajectories. Such a development should in principle allow clearer visualisation of where aircraft will go, and in particular whether they will be in conflict. Even with such information, however, it is not immediately obvious how controllers could use it.

2.4 The HIPS System

One system which presents all this information in a usable way is HIPS (Highly Interactive Problem Solver) (Meckiff and Gibbs, 1994), a system developed at the Eurocontrol Experimental Centre in Paris. HIPS is a novel support tool which comprises two main parts: firstly, it displays conflict situations relative to one selected aircraft in a time-independent way, and secondly, it provides a means for the controller to modify trajectories and to find solutions to these conflicts. A simple example follows which will help illustrate HIPS techniques.

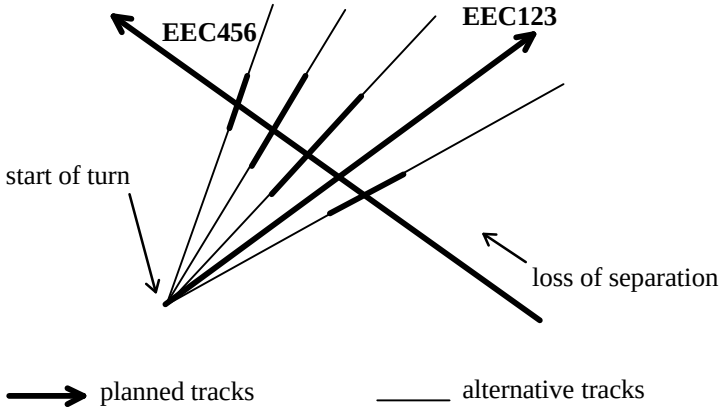

Figure 2.2: *A possible conflict representation.*

In Figure 2.2, the aircraft which interests us, EEC123, is traversing the airspace from left to right. Its trajectory is in conflict with that of another aircraft, EEC456, which is travelling in a northerly direction. The part of the trajectory for which there is a loss of separation between EEC123 and EEC456 is marked with a thicker line. If we imagine that we wish to solve this conflict by changing EEC123's heading, we could attempt various new headings assuming a certain point as our start of turn and for each one we could check for conflicts and again mark any loss of separation in bold.

Having tried a number of possibilities the next step is to group together all the bold lines to produce a single "no-go" zone as shown in Figure 2.3. This provides an immediate and powerful visual device by which the controller can rapidly see that in this case the conflict can be solved by a relatively small southward or a larger northward deviation to EEC123.

The example assumes linear constant-speed trajectories with the start-point of the manoeuvre already known. Unfortunately these assumptions are unrealistic in real life which means that the techniques used for generating the diagrams are quite complex. As well as generating a horizontal view, a similar approach can be used to produce diagrams for vertical and speed dimensions, giving a total of three pictures.

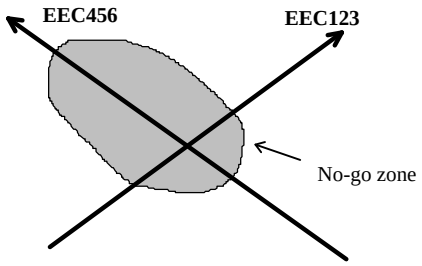

Figure 2.3: *How HIPS represents the conflict.*

Figure 2.4 (*a*) and (*b*) show two screen shots with the three HIPS windows and the no-go zones. The display of Figure 2.4 (*a*) is similar to the radar screen shown in Figure 2.1. The difference is in the red no-go zone that indicates the loss of separation between the two aircraft. The green and black trajectory belongs to the aircraft DLH407. Between the waypoints 3 and 4 this aircraft is in conflict with the aircraft CCK177. In Figure 2.4 (*b*), the speed (above) and vertical (below) views are displayed. It is shown that the conflict happens when the aircraft DLH407 descends from flight level 370 (37,000 ft) to 330.

The controller can try to solve the conflict by pulling the trajectory out of the red no-go zone. In this particular situation, the best manoeuvre is to keep the aircraft DLH407 at level 370 because the controller knows that the aircraft is too far from destination, so it is too early for a descent. The applied solution is shown in Figure 2.4 (*c*). The no-go zone is now

yellow because it is not a conflict anymore, but is a potential conflict. The modified trajectory is in white.

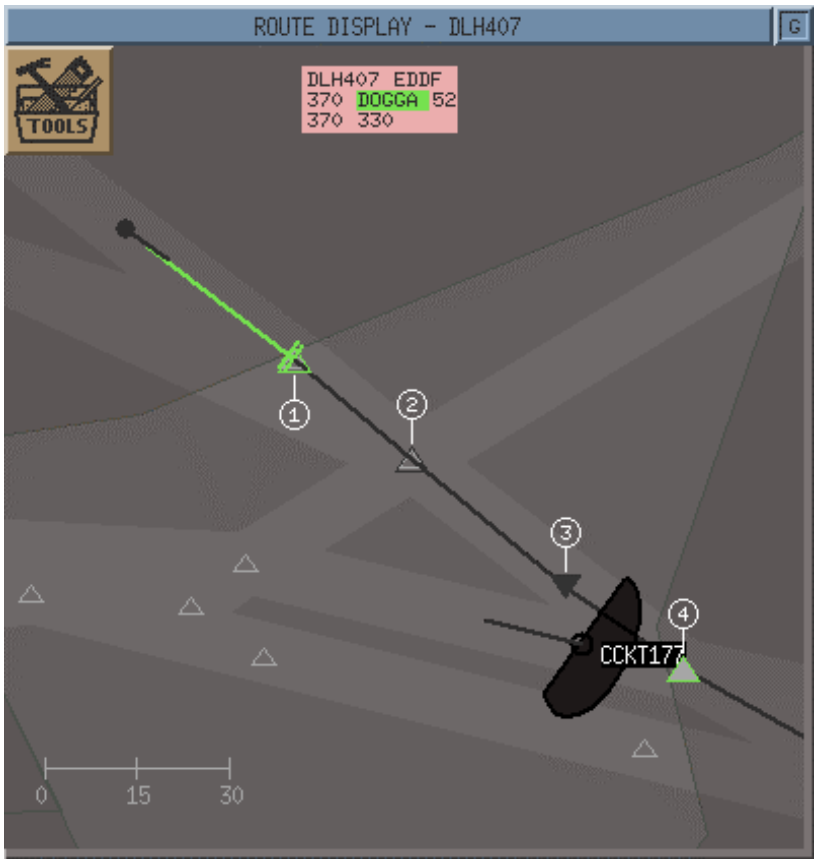

Figure 2.4 (a): *The horizontal display in HIPS.*

2.5 Conflict Resolution Support for HIPS

HIPS does not, of itself, attempt to present complete solutions. It presents information to the controller in a way that he can understand, and it is then up to him to find solutions. This approach has been important in gaining a degree of acceptance. However, there are still a number of steps to be taken between the time when a potential conflict is recognised, and the implementation of the solution. In particular, the controller must:

- evaluate the conflict situation and decide which aircraft he is going to manoeuvre,
- decide which type of manoeuvre is appropriate and
- determine the details of the manoeuvre (e.g. turn right 10°, go 0.1 Mach faster etc.).

These decisions imply the examination of the horizontal, altitude and speed display for each aircraft involved in the conflict. The aim of ISAC is to automatically highlight the display corresponding to the best manoeuvre of the best aircraft. This means that ISAC has to decide which aircraft has to be manoeuvred and the type of manoeuvre to avoid the conflict. The given solution can be accepted by the controller who will complete it with a deeper

specification of the manoeuvre, alternatively, it may be discarded because it is considered not adequate. If this happens, the controller will choose another display of the six available. The main purpose in having an intelligent system behind HIPS is to reduce the controller's workload. Moreover, the system could suggest a manoeuvre that did not come to the controller's mind, but is more efficient. Finally, HIPS and ISAC could be used as a training tool for non-expert controllers. The technical description of ISAC is in Chapter 4.

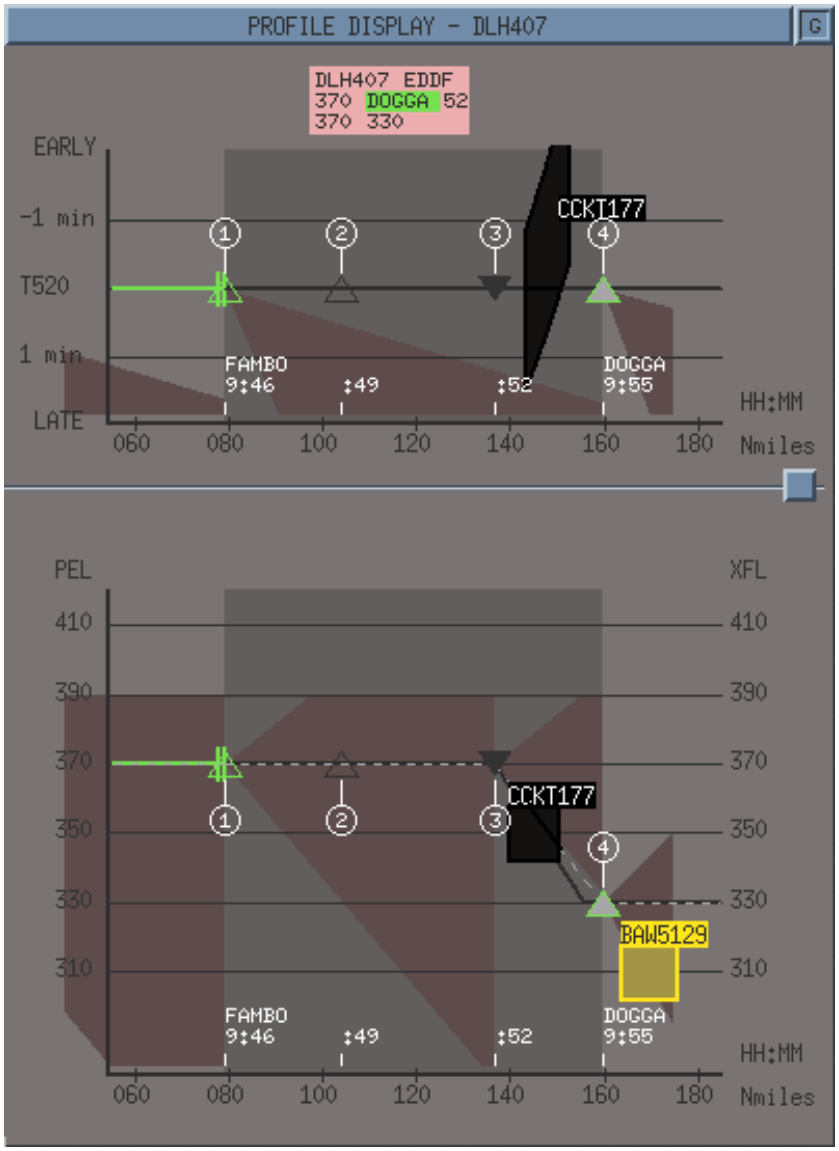

Figure 2.4 (b): *The speed and vertical displays in HIPS.*

2.6 Experts Systems for Conflict Resolution in ATC

Air Traffic Control is one of the domains where AI is applied with some reticence because a wrong decision could imply a loss of lives. That is why a characteristic common to all the AI systems applied to ATC is that they help and support the controller but they never try to substitute him. Some expert systems used in air traffic control are analysed below.

AIRPAC

AIRPAC (Shively and Schwamb, 1984) is another rule-based system for aircraft conflict resolution written in LISP. In contrast with ASTA, AIRPAC gives an explanation on how the solution is reached. After the suggested manoeuvre has been applied, the conflict is checked again to ensure that the solution does not generate new conflicts.

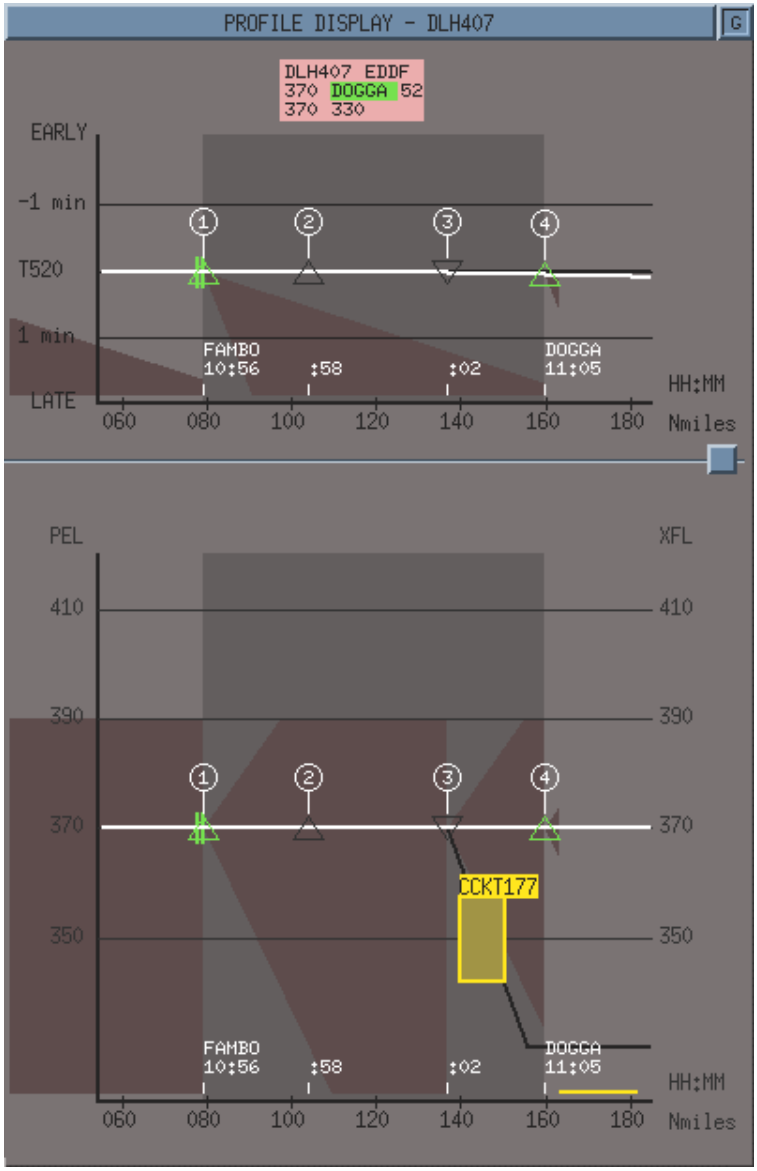

Figure 2.4 (c): *The speed and vertical displays in HIPS with the solved conflict.*

Three groups of parameters have been identified to describe a conflict: conflict description, constraints on resolution and goals of resolutions. The conflict description contains two subclasses:

- conflict situation with parameters like geometry, distance from unsafe separation, aircraft relative position and speed,

- aircraft situation with parameters like speed, manoeuvre status at conflict and type of aircraft.

A constraint can be generated by either the aircraft, e.g., maximum altitude, climb rate, speed, pilot ability to comply, aircraft not subject to manoeuvre, or by the environment, e.g., neighbouring aircraft, special-use airspace, severe weather, boundary considerations etc.

Different goals of resolution have been identified. In absence of special aspects of the conflict situation, AIRPAC reverts to a conflict resolution policy good for any type of situation. As soon as a good solution is found the search is stopped.

Two sources of uncertainty are examined: the uncertainty due to the input data and the uncertainty due to the heuristic knowledge. Input data could be incorrect because of the estimation of aircraft flight paths based on flight plan data. Heuristic knowledge is not always complete and consistent because of both general and specific problem solving methods. Some parameters used in ISAC come from the list of rules used in AIRPAC.

ASTA

Another rule-based system that was intended to be part of ARC2000 is ASTA (Tumelin, 1990). It is written in PROLOG and its aim is to help the controller by giving him in advance all the conflict-free trajectories and a proposed exiting altitude for all the aircraft entering the sector.

In ASTA only two aircraft conflicts are considered and are classified in three classes that depend on the horizontal geometry of the conflict: converging, catching-up and facing. The "status" of an aircraft depends on its altitude profile and can be: cruising, climbing or descending. Fourteen different conflict configurations are obtained from the combination of the horizontal geometry and the altitude status. This categorisation was adopted with some changes in the first version of ISAC's case-base.

Three types of manoeuvres are considered in ASTA:

- change of the horizontal position (6 different manoeuvres like heading change, maintaining heading for a longer period, direct route etc.)
- change of the vertical position (9 manoeuvres: level change, anticipate-delay descent-climb etc.)
- change of performance (7 manoeuvres: speed change, increase-decrease climb-descent rate).

In ISAC this approach is simplified. There are only 3 possible manoeuvres: horizontal, vertical and speed manoeuvre. When similar solutions are found in ASTA, an algorithm chooses the solution that reduces most the length of the trajectory. This choice is not optimal and a cost function would work better.

ASTA does not always find a solution. From ISAC's point of view, this is acceptable, because it operates as a support tool always under the supervision of the controller. A problem with ASTA is that the solution given cannot be immediately understood by the controller because it is often difficult to find the rules that brought to the solution. This is typical of all the rule-based systems.

In ASTA two different cost functions are examined: the controller's cost function and the aircraft's cost function. The parameters considered for the aircraft function are the safety, the flight time increase, the fuel consumption and the respect of the scheduled arrival time. For the controller's cost function, the number of manoeuvres to avoid the conflict and the environmental conditions are considered. These parameters should be kept in mind in case the construction of a cost function in ISAC will be necessary.

ARC2000

In ARC2000 (Nicolaon and Tumelin, 1992), a system developed in Eurocontrol Experimental Centre, Paris, the shortest path around the no-go zones is found algorithmically. The system tries to move the selected aircraft from its trajectory to a new one with a change in altitude, speed or horizontal position. ARC2000 defines the priorities between flight phases and between aircraft and the manoeuvres to apply to the selected aircraft. All the possible manoeuvres are successively tried and a cost is associated to each solution. Then, the least expensive solution is chosen. The algorithms for the search of the manoeuvre and the rules for its evaluation are implicit in the ARC2000 program code, making it difficult to test, maintain or adapt to new problems. The actual weather is given with wind speed and wind direction for 8 arbitrary altitudes and with temperature and pressure at sea-level. All given values are constant over the whole simulation area and over time. There are four vertical flight phases: *climb, descent, pre-descent, cruise*. In ISAC, no difference is made between the *pre-descent* and the *descent* phases.

The rule-based system in ARC2000 gives the basic structure for the rule-based system in RAMS, a Reorganised ATC Mathematical Simulator, (Model Development Group, 1995), an analysis tool to increase the simulation capabilities.

GEARS Conflict Resolution Algorithm

Also GEARS, Generic En-route Algorithmic Resolution Service, has as background the ARC2000 research. The algorithm combines the two steps of finding the right manoeuvres and putting them into the trajectory with the idea of the no-go zones (Irvine, 1997). Two similarities with ISAC are that the algorithm needs a conflict detector and that the trajectory predictor must provide reliable data.

The algorithm, that has applications in free-flight simulations, makes use of the concept of *preferred* manoeuvre, *candidate* manoeuvres and *avoiding* manoeuvres (Irvine, 1997). The right manoeuvres are recursively searched and the good ones are used to construct a set of conflict-free trajectories. The Rubber-Banding heuristic proves to be particularly powerful in avoiding the construction of sub-optimal trajectories. This heuristic comes from the idea of threading a rubber band between fixed obstacles and then stretching it around the no-go zones.

OASIS

The OASIS air traffic management system (Ljungberg and Lucas, 1992) performs tactical air traffic management. In order to alleviate air traffic congestion the system maximise runway utilisation. OASIS is agent-oriented: its major components are independent agents, each solving a part of the overall problem. The system's flexible behaviour results in part from this co-operative problem solving approach, and in part from the multiple levels of feedback employed between agents in the system and between the system and its environment. OASIS computes the landing sequence using an any-time algorithm and is implemented using the Procedural Reasoning System (PRS), a real time reasoning system capable of reasoning about and performing complex tasks in a robust and flexible manner.

Other Systems

The most exotic AI techniques have been applied to ATC, from genetic algorithms (Gotteland, 1995) to the use of the potential field method (Zeghal, 1994). In both these approaches the conflict is simplified by considering only horizontal manoeuvres and aircraft flying at the same level. Moreover, Gotteland assumes that the aircraft are cruising at the same speed. When using the potential field method, the goal, which is the destination in the ATC domain, produces an attractive potential which pulls the aircraft towards it, while the obstacles, i.e., the other aircraft involved in the conflict and the environmental aircraft, produce repulsive potentials which push the aircraft away from them.

Another rule-based system that gives a solution in the form of a conflict free trajectory is Aera (Hamrick, 1991). The possible manoeuvres given by the system are: vertical, horizontal or speed change, a combination of the two, or a solution that involves two aircraft. Approximately 100 rules are used to search for all the possible manoeuvres and to rank them in a best-worst list. The system, written in LISP, is able to generate alternative resolutions in case a pilot cannot accept the initial resolution. Aera's algorithm takes into account statistical uncertainty in the prediction of the future aircraft positions.

PLATONS (Ly, 1987) is a rule-based system written in PROLOG for altitude level allocations planning. This is usually the job of one of the two controllers that monitor a sector while the other tries to re-route aircraft to improve efficiency. In PLATONS, the negotiation with the pilot is very important and the final decision depends on this.

In (Bayles et al., 1993), CBR is used to capture and analyse experiences of Traffic Flow Management (TFM). The goal of TFM is to organise complex air traffic flows through busy areas like airport sectors. ATC becomes relevant when TFM fails and there is a potential conflict. Because ATC is different from TFM, the indices that describe a case are different: in the system for TFM, more stress has been put on the weather conditions, on the day of the week and on the period of the day. Moreover, the scope is not limited to only two or three aircraft but to an entire group.

The typical CBR issues are treated. In particular, the domain of applicability which has been limited to a specific situation. This happened because in a more general situation too many parameters would have been necessary. The authors agree that CBR has some advantages over RBS, but admit that "CBR must be complemented with other systems such as RBS to build successful application, including our application" (Bayles et al., 1993).

Suggestions on the use of bayesian networks and fuzzy logic for conflict resolution are in (Meckiff, 1994). The steps that compose the model are: definition of the inputs, definition of fuzzification functions for the inputs, definition of the relationships and development of a graphical model and assignment of conditional probability values to the relationship. This model has not been implemented yet.

From the overview of all the systems it can be seen that the majority of them use either and algorithmic approach or a rule-based approach. In both the situations the authors reported problems, for example during the extraction of the knowledge from the knowledge base and for the maintenance of the system. Case-Based Reasoning can help in these bottlenecks. Even if it does not solve all the knowledge engineering issues, as reported above by Bayles, it helps in reducing them. In the next chapters it will be shown how CBR reduces

considerably the steps that come after the understanding of the domain, i.e. the need to identify causal models in the problem domain is reduced.

2.7 The Future

Some of the biggest changes in the future of ATC will deal with the Human Machine Interface (HMI) field like frequency congestion that indicates the difficulties in voice communication between pilots and controllers. Some english controllers admitted that when faced with a conflict involving, for example, an English and a Chinese aircraft they tend to manoeuvre the English aircraft because they are sure that they will be better understood. The apparent solution to frequency congestion will be digital data link (Perry, 1997). Some of the areas not related to the HMI domain where the most effective changes will take place are the introduction of the Reduced Vertical Separation Mode (RVSM), the introduction of free flight and having controllers that will control some aircraft for all their journey and not anymore only when the aircraft passes above a particular sector.

Nowadays, aircraft have a vertical separation of 1000 ft when they are below 29,000 ft and 2000 ft above this level because the higher an aircraft goes, the less precise the altimeter is. RVSM has become possible now that aircraft have more precise instruments on board and will imply a separation of 1000 ft even above 29,000 ft.

With the introduction of on-board tools like TCAS (Traffic alert and Collision Avoidance System) in the USA since 1993 and in Europe from year 2000 the possibility of having aircraft going on a straight line from departure to destination seems more feasible. In the ATC communities there is a big debate on whether introducing free flight as it has already been done in the USA above a certain flight level (40,000 ft). The problem is that controllers do not feel at ease in a scenario where aircraft can arrive from anywhere and go wherever they want because the controller cannot any more easily recognise conflicts. On the other hand, the advantage of free flight would be in time and fuel saving because of the reduction of the trajectories. Finally, with the new technology improvements, the new radar have a much wider range and nowadays they can easily follow an aircraft along all its flight path. Moreover the idea of having sectors that usually have different standards above each country starts being considered obsolete. Mainly for these two reasons the controller might change his function. He will not be anymore bounded by the sector's borders having to control only the aircraft the overfly it, but he will take care of the same aircraft for all the duration of the flight, from departure to destination.

Chapter 3

Case-Based Reasoning

Case-based reasoning has emerged from research in cognitive psychology as a model of human memory and remembering. It has been embraced by researchers of AI applications as a methodology that avoids some of the knowledge acquisition and reasoning problems that occur with other methods for developing knowledge-based systems.

3.1 CBR Principles

The basic assumption of CBR is that, rather than solve a problem from first principles, it may be easier to retrieve a similar problem and transform the solution to that problem. The main issues to be considered in developing a CBR system are:

- representation and indexing,
- retrieval,
- adaptation,
- learning.

One of the central advantages in using a case-based approach to developing knowledge-based systems (KBS) is that CBR systems can be developed without encoding a strong domain theory for the problem domain. This means that CBR should avoid much of the knowledge engineering bottleneck that is such a problem in KBS development. In the next chapters it will be shown that although this might be true for toy systems, it is not completely true for a real world application like ISAC.

3.1.1 Representation and Indexing

Problem solving episodes are represented as cases, the key part of the case being the set of parameters that characterise it and the possible values that each parameter can assume. The case description is then completed with its solution that can be either atomic or compound. This is an important issue, since the performance will depend on the representation adopted and a lot of knowledge engineering is needed. The parameters must represent all the

knowledge necessary for the distinction of a case from the closest ones with a different solution. Cases may be indexed on key parameters in order to facilitate retrieval.

Solutions to the case can be atomic, compound or compound-manipulable. An atomic solution cannot be decomposed whereas a compound solution can be decomposed into one or more components by some problem decomposition process (other than adaptation). A compound-manipulable solution has components that can be manipulated during adaptation.

Incremental-CBR

An analysis of the use of CBR in different domains illustrates that the structure of conventional CBR is very rigid when compared with the flexibility of reuse that humans exhibit in problem solving. For some CBR tasks, like diagnosis, a full case description may not be available in advance of case retrieval. The standard CBR methodology requires a detailed case description in order to perform case retrieval and this is often not practical as the case can be characterised by a large set of parameters, not all of which are required in order to make a diagnosis. Moreover, many of these parameters will be expensive to determine so it is desirable that the number required to deliver a good solution should be minimised.

In this situation a technique called Incremental-CBR (Cunningham, Smyth and Bonzano, 1998) can be used. The incremental CBR mechanism can initiate case retrieval with a skeletal case description which is used to retrieve a matching subset of the case-base. This retrieved set is analysed to determine discriminating tests that the operator is asked to perform. The Incremental-CBR technique proved capable of retrieving good matches while requiring a minimal case description (Cunningham, Smyth and Bonzano, 1998).

3.1.2 Retrieval

The choice of the retrieval algorithm can increase or decrease the retrieval time but more importantly, can influence the selected cases that lead to the final solution. The simplest and most common retrieval algorithm is the Nearest Neighbour algorithm which is a *lazy* learning flat search algorithm. A learning algorithm is *lazy* when the processing is deferred to run-time. A consequence to this is that all the knowledge base has to be completely searched every time the system is asked for a solution. *Lazy* learning algorithms do not require any training period but are slow because the knowledge base has to be re-examined at run-time.

As opposed to *lazy* learning algorithms we have *eager* ones. *Eager* learning algorithms build a structure that represents the knowledge base before run-time. Approaches like

Neural Networks (Naughton, 1995), (Micarelli and Sciarrone, 1996) or Decision Trees (Quinlan, 1986) are of this type. They are very fast because the knowledge base is accessed only during the training but not anymore at run-time. One disadvantage is that the training period could be long. The two main alternatives for retrieval are *k-NN* retrieval and D-Trees.

3.1.3 Adaptation

When the retrieved case is not a perfect match for the problem in question, it must be adapted to fit the new situation. A lot of research has been done on adaptation even if in (Barletta, 1994) it is argued that adaptation should be kept as simple as possible and should not be essential for the success of a CBR system.

A preliminary analysis of the CBR literature suggests that CBR adaptation might be divided into three categories arranged in order of increasing complexity as follows (Smyth and Cunningham, 1993).

- Substitution Adaptation: this is the simplest type of adaptation and merely involves adjusting or substituting some of the parameters in the solution.
- Transformational Adaptation: this adaptation is more complex and involves structural changes to the solution.
- Generative Adaptation: this is the most complex adaptation and involves a reworking of the reasoning process in the context of the new problem situation. Generative Adaptation is also known as Derivational Analogy.

These different adaptation categories are appropriate for problems of different complexity. Substitution Adaptation will only work for comparatively simple problems where the solution statement is simple or atomic, e.g. it is expressible as a single price or a fault category. Transformational Adaptation can work where the solution has a more complex structure like in a plan but the components of the solution are not very interdependent. Transformational Adaptation offers more coverage than Substitution Adaptation because cases can be transformed into a wider variety of solutions but a more complete domain model is required to do so (see Figure 3.1). This implies a deeper knowledge model. For problems where the solutions are made up of interdependent components, as occurs in design for instance, solutions are too brittle to be transformed in this manner. Instead, it is necessary to re-generate solutions as is done in Derivational Analogy.

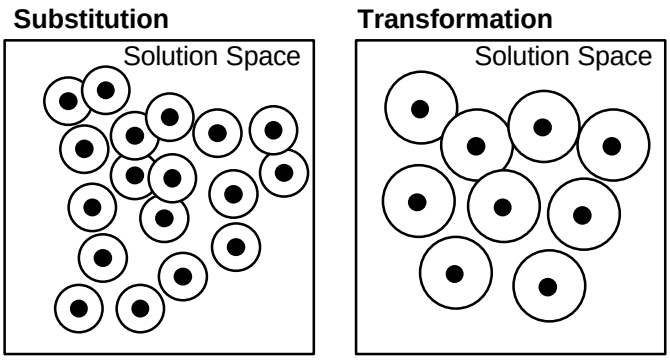

Figure 3.1: *Transformation adaptation has more coverage than substitution.*

3.1.4 Learning

Once new problems are solved with the aid of the CBR system, it may be useful to add them to the case-base. This mechanism, which is called the update mechanism, reflects quite closely human learning behaviour. From this point of view, the advantages of CBR over Rule-based Systems (RBS) are noticeable. A new case can be added to the case-base with no particular precaution, whereas a consistency check has to be done before adding a rule to the knowledge representation of a RBS.

The extreme situation when two cases with identical descriptions but with different solutions are introduced in the case-base should cause the system to give a double possible solution without generating any inconsistency. This is one of the situations where the human intervention is essential in the decision process as already mentioned in Chapter 1.

A policy is needed to decide whether it is worthwhile or not to update the case-base. If all the new solved problems are added as cases, the case-base could become too big and the retrieval process too slow. Moreover the solutions given by the system might not necessarily change for the better if the updating is not supervised.

3.1.5 An Example

The Breathalyser (Doyle, 1997), a Web-based CBR application that predicts the blood alcohol content, is an example of the CBR cycle presented above. A case is stored as a flat parameter record and five parameters are used to characterise it: the gender and the weight of the person, the units of alcohol consumed, whether the person consumed some food and the duration of the drinking session. These five parameters come from the medical literature on the subject. Each parameter has an importance weight which is fixed *a priori* by the expert and remains the same for all the cases. In Chapter 7 we show how it is possible to

automatically extract weights from a given case-base and to determine the context sensitivity of the parameters.

In the Breathalyser process, the two main steps are retrieval and adaptation. The retrieval engine retrieves the closest case to the input case, then the solution to this case is adapted using adaptation rules automatically learned from the case-base (Hanney and Keane, 1996). When applicable adaptation rules are found, the system is fairly accurate but when no applicable rules are found, it is not as accurate, the accuracy depending on how close a match for the case input is found in the case-base. As the case-base used with this project is quite small, there are usually no very close matches although adaptation makes up for this in a lot of situations. The information given when a solution is returned by the system gives some indication of the accuracy of the answer.

Some sample cases and associated rules are shown in Figure 3.2.

CaseName	n1	CaseName	n55	CaseName	n3	Casename	n33
Gender	male	Gender	male	Gender	female	Gender	female
FrameSize	1	FrameSize	1	FrameSize	4	FrameSize	6
AmountConsumed	1	AmountConsumed	3	AmountConsumed	4	AmountConsumed	3
Meal	snack	Meal	snack	Meal	full	Meal	full
Duration	60	Duration	120	Duration	90	Duration	90
Solution	0.2	Solution	0.7	Solution	0.8	Solution	0.5

The rule generated by comparing cases `n1` and `n55` above is:

r0: *if the units of alcohol consumed changes from 1 to 3 and the duration of the session changes from 60 to 120, then increase blood alcohol content by 0.5.*

The rule generated by comparing cases `n3` and `n33` above is:

r25: *if the frame-size changes from 4 to 6 and units of alcohol consumed changes from 4 to 3, then decrease blood alcohol content by 0.3.*

Figure 3.2: *Some sample cases and associated rules.*

The domain in which the Breathalyser works is a "weak theory" domain, i.e., there are no applicable algorithms or formulæ to compute blood alcohol content from the five parameters used to describe the cases. Therefore reasoning from cases is the only option. The performance of the system support the assumption that CBR works well in "weak theory" domains.

3.2 Overview of Relevant CBR Systems

In (Hanney et al., 1995) 53 case-based reasoners have been examined to build a taxonomy of systems and tasks useful in the initial stages of the design of a CBR system. Four dimensions for the classification of the CBR systems are identified.

- Whether adaptation is present or absent.

- The solution is extrapolated from either a single or multiple cases.
- The solution is either atomic, compound or compound-manipulable.
- There may be considerable interaction between solution components constraining the effectiveness of naïve manipulation during adaptation.

This classification and the initial understanding of the ATC domain from Chapter 2 give us some directions on how to apply CBR to conflict resolution in ATC. Some of the hints from the literature are useful, some others seemed to be useful at the beginning, but as our understanding of the problem improved during the making of the system, they could not be applied. In the next sections the initial approach to the system is presented, whereas in Chapter 6 all the final choices are presented and justified. The technical aspects of the system are discussed in Chapter 4.

3.2.1 The Case-Base

In a real world application such as this, there is a strong argument for populating the case-base with hand-crafted high quality cases (gold standard cases). By doing this the system should be able to fulfil its double function of helping the controllers in taking solutions and teaching non-experts the steps to take the right solution. It seemed that a small set of cases, 30 to 50, would have been adequate, but when the real complexity of the system was discovered the dimension of the case-base had to be increased of at least one order of magnitude.

The alternative to the gold standard cases option is the use of learned cases. This option is valid when certain situations will recur regularly and it is desirable that the system should be able to learn good solutions as they are developed. For example, if two flights systematically conflict in a particular configuration, it is desirable that the system is able to learn a good solution to this conflict.

Learning from failures like in PROTOS (Bareiss, Porter and Murray, 1989) or storing unsuccessful cases as done in CADET (Sycara and Navichandra, 1989) could be useful when the system will focus on the solution of conflicts on a particular sector. If a particular conflict configuration happens often and the most obvious solution is known not to be the correct one, it could be useful to have a message that says: “do not choose this manoeuvre”. This approach does not work in a very general situation because there would be too many exceptions.

Hierarchical Structure

As it will be explained in the next chapter, ISAC's solution has two components: which of the aircraft involved in the conflict has to be manoeuvred and the type of manoeuvre that has to be applied. It seemed that because of this double solution each case could be broken into two sub-structures, each one dealing with one part of the solution as done in APU (Bhansali and Harandi, 1993) and ARCHIE (Domeshek and Kolodner, 1992). The alternative of having one solution that includes both the components at the same time is simpler and proved to work as well.

Very often in the ATC domain a conflict involves more than two aircraft. If this happens we have a multiple aircraft conflict that can be decomposed into two aircraft conflicts. The problem is that the resulting conflicts do not necessarily have solutions independent to each other. Maybe a common solution could solve the multiple aircraft conflict more efficiently, e.g., by manoeuvring the aircraft which is in conflict with all the other aircraft in the conflict. With a multiple aircraft conflict a hierarchical structure of the same type as the system Déjà-Vu (Smyth and Cunningham, 93) can be used and in Chapter 5 we show how. We reuse the case-base of the two aircraft conflict by building some abstraction hierarchy as done in CADET (Sycara and Navichandra, 1989).

3.2.2 The Case Representation

This proved to be the key issue in ISAC. A concrete case representation is available from the host system that is the basis of the actual case representation, so the initial set of parameters will be acquired from the host system as is done in Archie (Domeshek and Kolodner, 1992). Many of the parameters are represented numerically but sometimes the representation is expanded to produce some more abstract symbolic parameters that support useful reminding in the case retrieval process. A similar approach for the conversion of numeric parameters into symbolic ones with the use of ranges is used in CLAVIER (Hinkle and Toomey, 1994). The parameters for the case representation come from the controller's habits in solving a conflict. An accurate description of some of these typical habits, called "preferences", is in (Meckiff, 1994).

In the original case representation we had two kinds of parameters: some used for the retrieval of the case, others for the case adaptation and for building the solution. This approach, inspired by the system Déjà-Vu (Smyth and Cunningham, 1993), has been simplified after the first discussions with controllers because it became clear that all the

parameters had to be taken into account for both the case retrieval and the solution. Moreover our adaptation mechanism is almost non-existent (see next section).

In the system JULIA (Hinrichs, 1988), the unsolvable parameters are either weakened or not considered. This is a quite common situation in the air traffic control domain, where, for example, data and performance about an aircraft might not be available. If this happens the controller uses his background knowledge. Our system can either retrieve the missing information from a common database or simply assign a "don't care" to the missing value. If the database is well structured, the retrieval of the missing data should not take too long.

The case solution has, in Hanney's terms, a compound manipulable structure (Hanney et al., 1995) because it contains the name of the aircraft to manoeuvre and the kind of manoeuvre and it can be extracted from more than one case. The option of storing the sequence of manoeuvres necessary to solve the conflict as done in PRIAR (Kambhampati and Hendler, 1992), will be considered if the system will be asked to give more specific solutions. If more than one case is retrieved, some control rules as used in PRODIGY (Carbonell and Veloso, 1988) could be useful.

Granularity of the Case Representation

In situations of increased traffic the future ATC scenario implies more complex conflicts involving more than two aircraft. A key design criterion has been to develop a case representation that will be extendible from two aircraft conflicts to conflicts involving three or more aircraft. This militates against having a single conflict as the basic unit of retrieval, i.e. the case (Bonzano, Cunningham and Meckiff, 1996). For reasons of economy in case coverage, we want solutions in two-aircraft conflicts to be reusable in three-aircraft conflicts, and so on. This means that conflicts should be decomposable so that the basic unit of retrieval is an individual aircraft in a conflict. This problem of representing cases describing two conflicting entities has already been faced in the CBR literature, for example in two classical systems, Mediator (Simpson, 1985) and Persuader (Sycara, 1987), and more recently in Truth-Teller (Ashley, 1995). In all these systems, perhaps because they describe interaction between humans, there is a vocabulary to characterise the "type" of conflict and this is critical in determining the solution. This is less true in ATC where the solutions depend on the arrangement of the aircraft and the context of the individual aircraft as described by their flight plans. The conflict between two aircraft can be described roughly with one or two global parameters but the final solution depends on a lot of dependent variables related to a single aircraft. For this reason the approach adopted in ISAC is

somewhat different to the above systems, with an emphasis placed on some parameters that describe an aircraft on its own. While our ultimate objective in developing ISAC is to have a single aircraft as the unit of case retrieval, we have considered three case organisations in detail. We have evaluated two alternatives with two aircraft per case and one alternative with one aircraft per case as shown in Chapter 5.

3.2.3 The Retrieval Mechanism

The two serious alternatives for case retrieval have been presented in Section 3.1.2. Retrieval may be based on a sequential search of the case-base using a tailored similarity metric as a basis of comparison. Alternatively, the cases can be stored in a decision tree of depth *k*, where *k* is the number of parameters considered in assessing similarity. Flat search has the advantage that sophisticated similarity measures can be used like the Foot-Print metric (Veloso and Carbonell, 1991) but it has the disadvantage that retrieval time increases linearly with case-base size. This is particularly a problem if the case-base is to be allowed to grow as may be the case in ISAC. Decision trees have the advantage that retrieval time is practically constant as the case-base grows. However the search may prove to be *myopic* with cases excluded from consideration because they do not match on a particular parameter. The spreading activation mechanism used in ISAC is an hybrid approach between the lazy learning mechanism and the eager one and is explained in detail in Chapter 4.

The way of calculating a similarity metric changed during the development of the system, e.g., the way of considering a “don’t care” value and the numeric parameters similarity policy, changed several times. There are different ways of calculating the similarity metric depending on whether the parameters is symbolic or numeric. We use a more elaborate version of the direct matching metric described in PRODIGY (Veloso and Carbonell, 1991): two parameters match either if they are equal or at least in the same range of values, or if each argument of parameter A is of the same type of the corresponding argument of parameter B. The Foot-Print metric is not used. This method identifies the set of weakest preconditions necessary to achieve the goal. Then it recursively creates the Foot-Print of the problem that has to be solved by projecting back its weakest preconditions into the initial state.

3.2.4 The Adaptation Mechanism and Update Mechanism

In our work, the adaptation is not very important because we assume that our case-base is dense enough to always provide a case close enough to the problem that has to be solved. In the conclusions we show that our assumption is wrong, the case-base is too complex and can only be partially covered for one sector. Nevertheless adaptation is not used because if there is adaptation there is a rule-based system behind it. If adaptation is too strong, the role of CBR is reduced as seen in (Hanney and Keane, 1996) and (Doyle, 1997). The aim of our research was to see how suitable CBR was for the ATC domain and for this reason we wanted to keep the influence of any RBS at the minimum. This view is supported by (Barletta, 1994) and from the development steps of the system CLAVIER (Hinkle and Toomey, 1994). In CLAVIER case adaptation was performed only in its first version, but the process was too error prone and in the final version it was up to the user to manually adapt the case.

Adaptation requirements could be met using a small set of heuristic rules that adjust the solution parameters. We would not aim to support any significant solution transformation in the adaptation process. It appears that the basic substitutional adaptation will be adequate in this situation. Case-base coverage should be sufficiently extensive that any structural transformations will not be required if not at the beginning when the conflict, e.g., a multiple aircraft conflict, has to be loaded for the retrieval as done in KRITIK (Goel and Chandrasekaran, 1989).

An updating mechanism as used in PROTOS (Bareiss, Porter and Murray, 1989) would be useful but in certain circumstances. The need to provide a learning facility in the system introduces a problem of consistency. Different air traffic controllers may provide different solutions to similar situations. Each controller has his own point of view depending on his habits in solving conflicts.

If the system is to incorporate such a learning facility it will also introduce problems of controlling case-base size. Prodigy is the only system where the time problem is treated analytically, with a distinction between the retrieval time and the adaptation time (Veloso and Carbonell, 1991) and advices for increasing the system performances, for example, by changing the retrieval mechanism. An updating function will try to reduce the sum of these two periods, by not keeping in the case-base solutions to problems that are easily and correctly adapted (i.e. with a short adaptation time). By doing this, the retrieval time is not increased because the case-base is not changed. Moreover, there will be a need to estimate the coverage of individual cases in order to control redundancy in the case-base.

It would be useful to be able to measure solution quality in order to rank different solutions. This might be achieved by estimating the cost of different manoeuvres by using simple estimates of fuel use and time use. In CASEY (Koton, 1988) such evaluation function consists of a rule-based system which is, again, a problem due to the complexity of the rules necessary to determine how good a solution is.

3.2.5 Time Constraints

The general architecture of a CBR system is discussed in (Hinrichs and Kolodner, 1991): all the functions that constitute the system should be integrated to minimise redundancy and to maximise efficiency. Information hiding and modularity should be achieved with a layered architecture. Inheritance should be used to propagate some values to different cases belonging to the same group. These guidelines have been useful for the definition of ISAC's structure. ISAC has to give the conflict solution as soon as the conflict is seen on the radar screen. Potential conflicts are automatically recognised 20 minutes in advance, but this does not mean that the system has 20 minutes to solve them because afterwards the controller has to complete the solution with more details and this will need some more time. Moreover, it is likely that other conflicts will appear and they may interfere with each other. So the time for the retrieval of the case and for its adaptation is very short as in the real time system ACBARR (Ram et al., 1992), where a robot under control cannot stop waiting for the system to take the correct decision. In ACBARR the system cannot stop to update the case-base because of the time constraints.

3.2.6 Introspective Learning and Discriminatory Power

REBECAS (Rougegrez-Loriette, 1994) predicts the fire behaviour in a wood. In this system it is necessary to choose what are the most important parameters because there are so many that it is impossible to check all of them. This implies the need of an expert to decide priorities in the list of parameters. This is not a user friendly approach and the automatic ways of learning the importance of the parameters given a case-base are more effective. Two similar methods are shown in Chapters 5 where the discriminatory power of the parameters is calculated and in Chapter 7 where an Introspective Learning mechanism is presented.

3.3 Conclusions

In this chapter we highlighted the theoretical basis of the work that we will describe in the next chapters. As pointed out in Ram et al. (1992), the five points that have to be pursued for the success of a CBR system are:

- the case-base must be complete,
- the case representation must contain all the relevant parameters,
- an efficient retrieving mechanism is needed,
- an efficient adaptation mechanism is needed,
- the solution must be evaluated in order to update the case-base or not.

The first three points listed above will be our list of priorities for the future work. The last two points, adaptation and update could be either treated or not, depending on how the other points are successful. We will see that the most difficult issue will be to have a well covered case-base. Different approaches will be tried but no one will prove to be better than the effective coverage with cases coming from the real world.

Chapter 4

Structure of the System and Acquisition of the Parameters

In this mainly technical chapter the architecture of ISAC is presented and the choices made are justified. The spreading activation mechanism is compared with the standard flat search mechanism and the advantages of the first are proved with some experiments. It is explained how ISAC has been interfaced with the system that provides the radar screen and the detection of the conflicts.

As it will be said in the next chapters, the process of the decision and acquisition of the parameters involved several steps. Some parameters introduced at the beginning of the knowledge engineering process have been discarded and other more descriptive parameters have been introduced. The final part of this chapter is dedicated to the analysis of these changes and the way these parameters are extracted from the data available. The reasons why the language used to write ISAC is C++ are explained in the last section together with some simplifications and assumptions.

4.1 The Environment and Technical Information

ISAC is a module of HIPS. HIPS, presented in Chapter 2, is embedded in a system called GHMI[4] that gives the controller a realistic environment to work. This system GHMI is shown in Figure 2.1. When HIPS is called from GHMI the three HIPS windows appear with all their usual functions, as seen in Figures 2.4 *a*, *b* and *c*.

[4] The Programme for Harmonised Air Traffic Management Research in Eurocontrol (PHARE) is a multi-year work programme, the objective of which is "to organise, co-ordinate and conduct - on a collaborative basis - studies, experiments and trials aiming at proving and demonstrating the feasibility and merits of a future air-ground integrated ATM system in all phases of flight". Ground Human Machine Interface (GHMI) is part of PHARE Demonstration 3 and consists of the development of guidelines for, and prototyping of, a common man-machine interface to improve efficiency in the combined use of ground functions.

When a conflict is detected in HIPS, its description is sent to ISAC: i.e. the flight plan and performance of the aircraft involved, the shapes of the no-go zones etc. Using this data, ISAC selects the aircraft to manoeuvre and the type of manoeuvre which it sends back to HIPS. Then HIPS can either highlight the display to be used by the controller in determining the final details of the manoeuvre, or can simply open a window with a message for the controller. Throughout this process, the controller has full visibility of all the data and has full responsibility for the manoeuvre that will be communicated to the pilot. ISAC merely *suggests* the "best" manoeuvre, based on the conflict solutions stored in its knowledge base. The conflict resolution process with the interaction between HIPS and ISAC is shown in Figure 4.1.

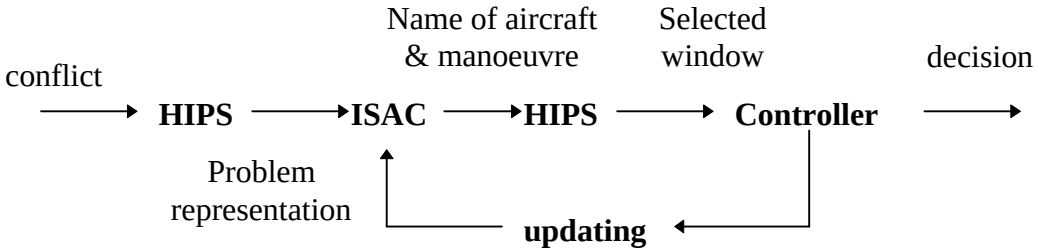

Figure 4.1: *How ISAC is embedded in HIPS.*

ISAC needs a supporting system with the ability to detect and describe the conflict. HIPS is this system in the prototype presented here, but another similar system could be used. The interface between ISAC and the supporting system varies depending on the data that the system can provide. This means that the case description could change if the supporting system is changed.

The current version of ISAC operates as a decision support system. It is certainly important for its acceptance in the ATC culture that it should be a support system rather than an expert system. The retrieval process is shown in Figure 4.2. A key criterion in the design of the retrieval mechanism in ISAC is that it should be fast because it will be required to operate in a real time environment. When a controller selects a conflict in HIPS for resolution, ISAC must immediately suggest a solution. The retrieval mechanism that has been settled upon is a two stage process. These two stages reflect the fact that the case parameters are divided into constraints and ordinary parameters. The characteristics of the domain dictate that there are some parameters that *must* be matched if cases are to be considered similar. These parameters are considered constraints and the base filtering stage selects cases that match on these constraints.

During the GHMI and HIPS start-up, ISAC loads the case-base into memory and builds a network of pointers among the cases that will speed up the retrieval process. The Base

Filtering mechanism discards from the case-base all those cases whose constraints do not match those of the target exactly. This step is necessary because of the characteristics of the domain but it also has the advantage that it reduces the size of the case-base before the comparatively expensive spreading activation stage. The choice of constraints could influence the competence of the system significantly because, as will be explained in Chapter 5, constraints cause cases to be eliminated from consideration.

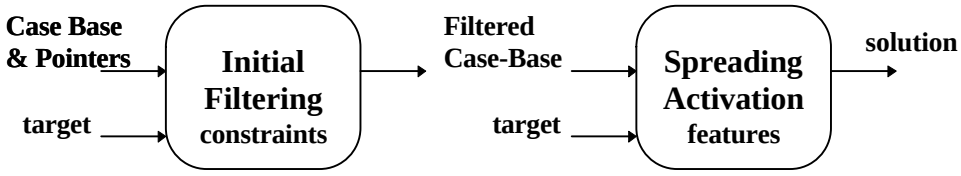

Figure 4.2: *The case retrieval architecture in ISAC.*

The objective of the next stage is to select cases that match the target best on the remaining parameters. The outcome is equivalent to *k*-Nearest Neighbour (*k*-NN) retrieval but is implemented as a spreading activation process for reasons of speed. The pointers link all the cases that have the same value for a given parameter. During retrieval, activation is calculated through these links. The importance of the different parameters is weighted and activation is proportional to this importance. A more detailed description of the functions executed by ISAC and the corresponding classes can be found in the next section.

In ISAC the solution can come from one or more cases and is compound manipulable (following the convention introduced in Hanney et al., 1995). At the moment there is no adaptation because the solution required does not specify the details of the manoeuvre and the case coverage should be sufficiently extensive that any structural transformation is not required. There is still some complexity in the reuse process in the aggregation of solutions when different cases with different solutions are retrieved. The policy adopted is explained in the section devoted to Solutions in Chapter 5.

4.2 Structures and Functions Used in ISAC

ISAC reads the case structure with the function `ReadCaseStruct`, then the case-base with the function `ReadCaseBase` and finally the targets with the function `ReadAllTargets`, where "target" is the conflict that has to be solved. The function `BuildWebOfPointers` builds a web of pointers from the data read, then the retrieval is started by the function `FindCases`. All of these functions are now examined in more detail. The header files that contain all the classes and functions used in ISAC are in Appendix C.

The Case-Base and the Target(s)

In the case-base file, the symbol "`@n`" marks the beginning of a case description and its name. The symbol "`@s`" marks the end of the case and its solution. All of the parameters are identified by a couple:

`(Parameter Name - Parameter Value)`.

The case-base and the target are stored in memory using the class `OneCase`, which contains a list of parameters, each one stored in the class `OneFeat`. The structure of these classes is shown in Figure 4.3. All the classes are defined in the file `header1.h`.

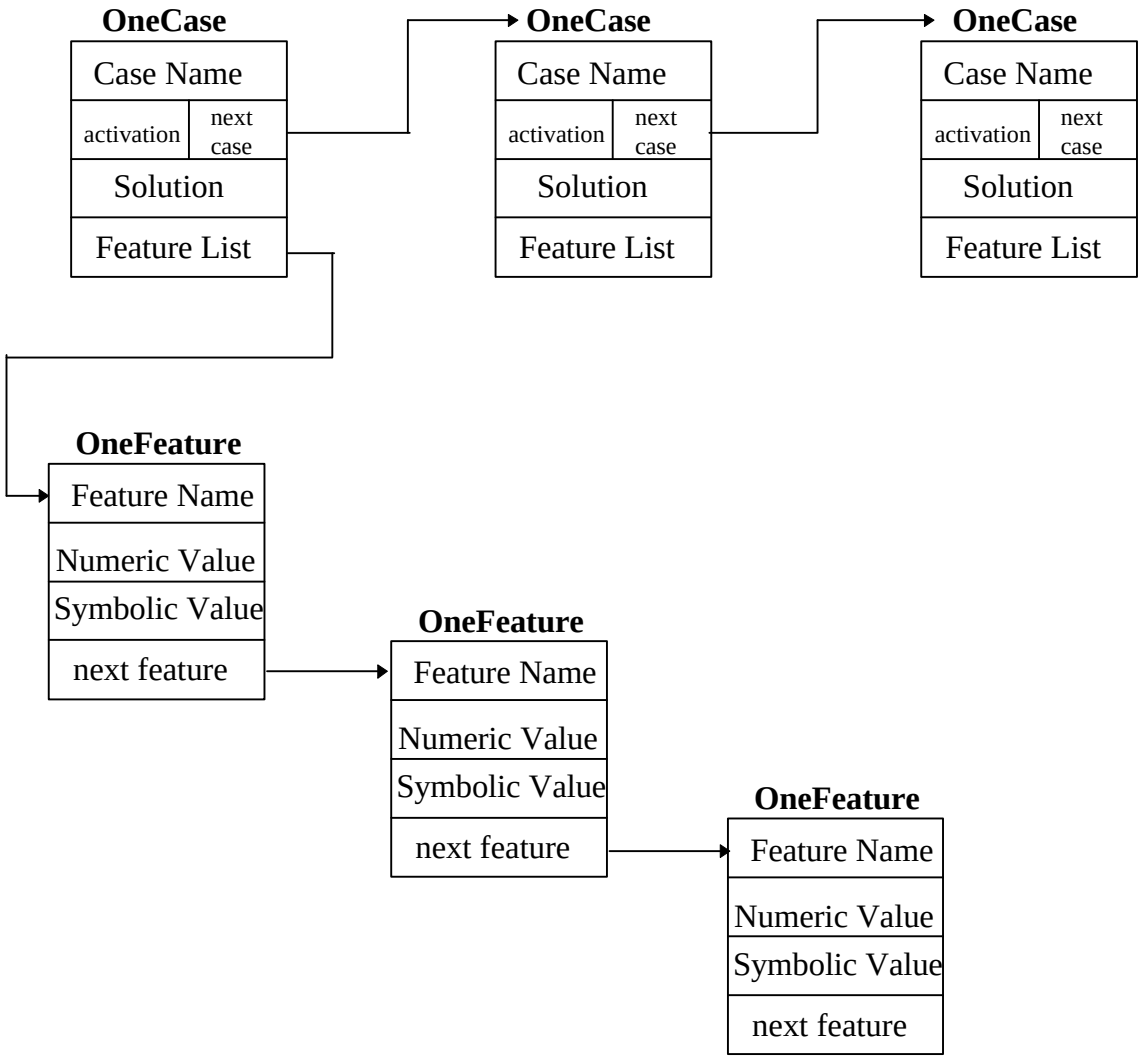

Figure 4.3: *The structure of the case-base in ISAC.*

The functions defined in `OneCase` and `OneFeat` are used to get or store data and to automatically scan the case-base to retrieve the desired information. In the class `OneCase`, the field `Activation`, not depicted in the figure because it is used internally, is used

during the retrieval. In the class `OneFeat`, only one of the fields `NumericValue` and `SymbolicValue` is used depending on the type of the parameter. This is memory consuming, but it is acceptable like other non-standard choices because the system is still a prototype.

All the lists can be of any length, and the field `next` of the last element of each list points to NULL. The target uses the same class used for storing the cases (`OneCase`), the only difference being that the field `Solution` is left empty or, for evaluation purposes, stores a solution suggested by the controller that is compared to the one found by ISAC. The two functions that read the data from a file and create this structure are `ReadCaseBase` and `ReadAllTargets`:

```
OneCase *CaseList=ReadCaseBase(FileWithCaseBase,StructList);
OneCase *TargetList=ReadAllTargets(FileWithTargets,StructList).
```

The function `ReadCaseBase` reads the file `FileWithCaseBase`, where all the cases are stored. The structure `StructList` is used to check that the names of the parameters and their values are acceptable. The function returns a pointer, `CaseList`, to the structure shown in Figure 4.3.

The function `ReadAllTargets` reads the file `FileWithTargets`, where all the targets are stored. The structure `StructList` is used again to check that the names of the parameters and their values are acceptable. The function returns a structure similar to the one returned for the `CaseBase`. The file `CaseStructure` is used to store all the information concerning the parameters: the name of the parameter, whether it is a numeric or symbolic value, whether it is a constraint or a normal parameter, its weight and, if it is a symbolic parameter, the possible values. The weight field implies that a weight is assigned to each parameter by an expert. In Chapter 7 a technique that automatically assigns the weights to the parameters is described. When in evaluation mode, the file `FileWithTargets` is artificially generated by an evaluation program and contains the description of a conflict used to test the system. When in operation mode, the file `FileWithTargets` is directly generated by HIPS and contains the description of the conflict which is visualised on the radar screen. The files `CaseStructure`, `CaseBase` and `Solutions` for the final version of the case-base used by ISAC are in Appendix D.

The Web of Pointers

The web of pointers is built during start up to speed up the retrieval process. For each possible value of each symbolic parameter, a list that contains pointers to all the cases that

have that value for that parameter is created. It would be inefficient to build the same kind of web for numeric parameters by dividing the numeric values into ranges.

The web is built using the function `BuildWebOfPointers`:

```
branch *Branches=BuildWebOfPointers(StructList,CaseList).
```

An empty branch is built for each symbolic parameter's value as read from `StructList`. The case-base is then searched to find all the cases that have that particular value and a pointer to that case is stored in the branch. The function returns a pointer, `Branches`, to the structure shown in Figure 4.4.

ISAC automatically eliminates any possible ambiguity between identically named values of different parameters by prepending on each value the name of the corresponding parameter. For example, if both the parameters "faster" and "slower" have the same possible value "easy", these two values are represented as "faster-easy" and "slower-easy" in the web of pointers.

The web speeds up the retrieval process because it takes less time to find all the cases that have the same value for a certain parameter by starting from `Branches`, rather than having to scan the entire case-base.

The Retrieval Mechanism

The retrieval of the best matching cases is executed by the function `FindCases`:

```
void FindCases(CaseList,TargetList,Branches,StructList).
```

In `TargetList` there could be either one or two targets depending on the case representation. This function consists of a set of instructions that are executed for each target present in `TargetList`.

For each target, the case-base is filtered according to the constraints, using the function `BaseFiltering`. The pruned case-base is returned with the pointer `SubList`. If there are no constraints, all the cases in the case-base are kept. The function `SpreadingActivation` calculates how similar each case is to the target. This returns a pointer, `FinalList`, to the list of all the cases that are equally most similar to the target under examination. This list is passed to the function `Analyse` that extracts only one solution for the target.

When these steps have been executed for all the targets, `ChooseFinal` finds the best solution for the conflict by examining the solutions for each target. This solution is either displayed on a window or it is sent back to HIPS, which highlights the window corresponding to the best manoeuvre.

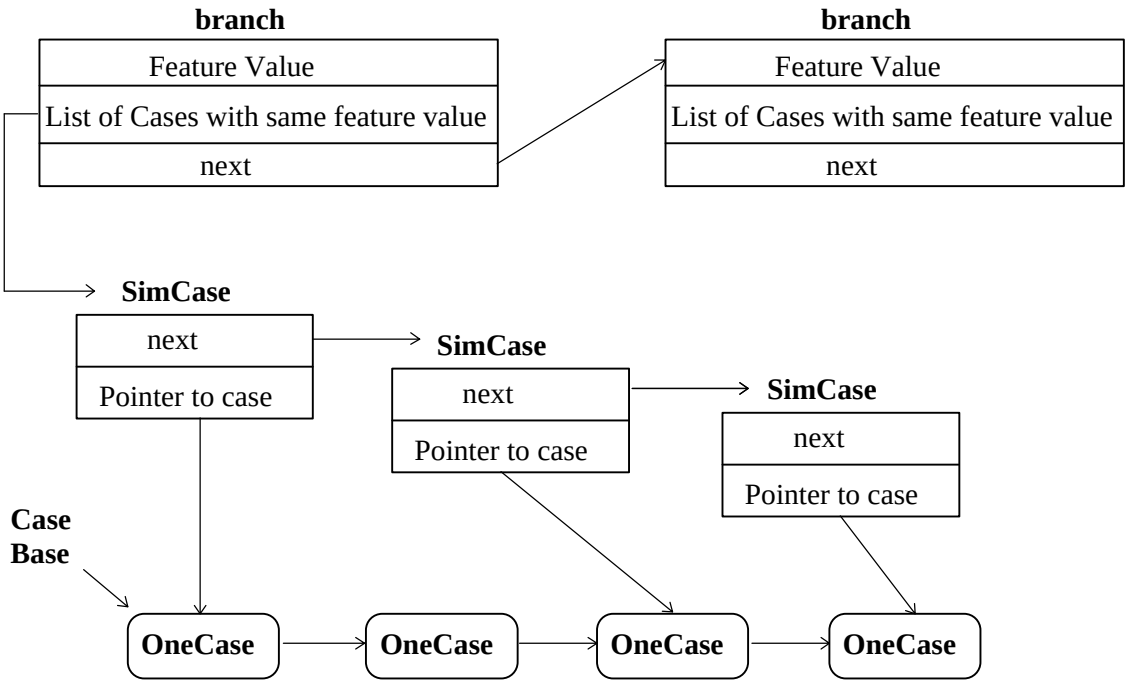

Figure 4.4: *The* `Branches` *structure.*

Retrieval Time Reduction with Constraints and with Spreading Activation

As it can be seen from Figure 4.5, the retrieval time when there are two constraints instead of one is smaller because less cases are passed to the function `SpreadingActivation`. Different tests with case-bases of different dimensions have been performed and the corresponding retrieval time is shown in the figure. CPU time rather than the clock time has been used in both time simulations because it is more reliable. The problem of losing some useful cases with the introduction of the constraints will be treated in Chapter 5.

In Figure 4.6, the retrieval time reduction using spreading activation is compared to that using flat search. Four different situations have been tested: flat search with symbolic and numeric parameters (F.S. N+S), flat search with only symbolic parameters (F.S. S), spreading activation with symbolic and numeric parameters (S.A. N+S), spreading activation with only symbolic parameters (S.A. S). The figure shows that the spreading activation mechanism is faster than the flat search mechanism. Spreading activation only works for symbolic parameters and does not work for numeric parameters. This explains why the retrieval time with numeric and symbolic values is greater than that with only symbolic values, (see figure).

The curves are not linear because in ISAC there are some functions that can only use flat search, e.g., the function that resets all the activation values before a new simulation. These functions will not be used in the real time system but are used here for evaluation purposes.

A list of “activated cases” is not built because it would take too much time to check if an activated case is already in the list.

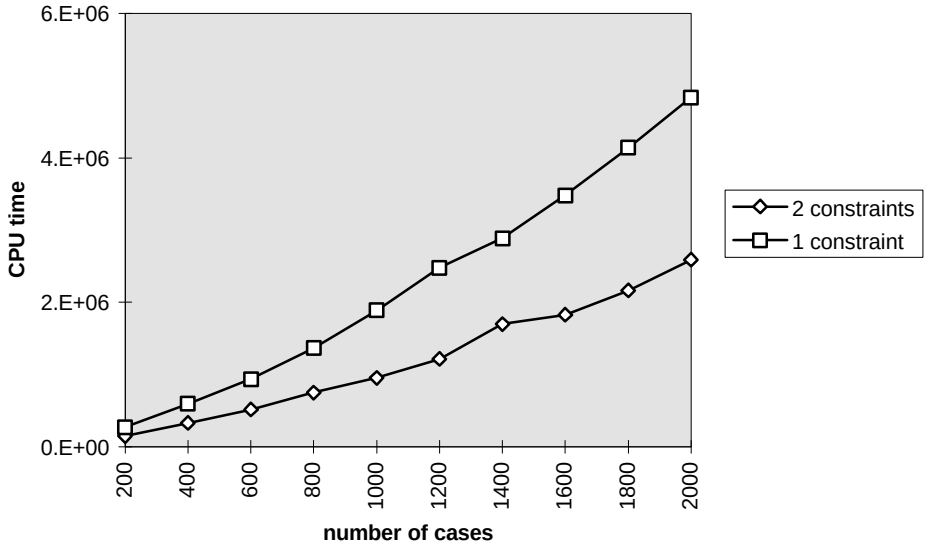

Figure 4.5: *Retrieval time reduction when constraints are used.*

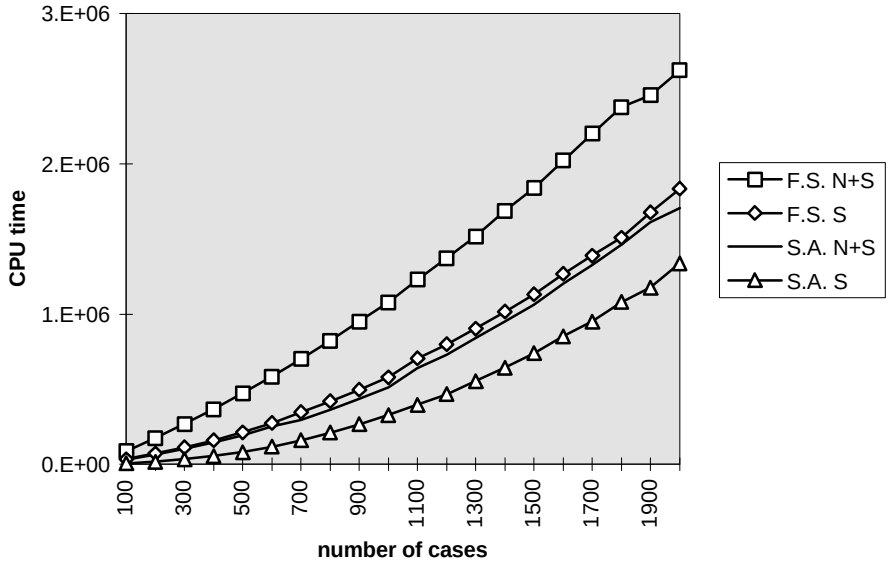

Figure 4.6: *Retrieval time with spreading activation and with flat search.*

Each simulation has been repeated several times and the average of the CPU time has been calculated. The “TwoInOne” case representation was used (see Chapter 5), but it is assumed that the results are extendible to any case representation.

4.3 The Acquisition of the Parameters in ISAC

In this section we describe the algorithms that are used for the extraction of parameters that describe a conflict from the data structure used in HIPS. We report the final version of the

algorithms and the differences from the original versions. The process of refining the acquisition of the values from the data provided by HIPS has been run in parallel to all the development steps. It was independent from the construction of the case-base, but, obviously, essential for the performance of the system.

CaseName

The name of the case is usually the callsign of the aircraft if the representation is "OneInOne", (see Chapter 5). If the representation is "TwoInOne" the name is made up of the two callsigns linked by an underscore. The time of acquisition is added to the end of the case name to eliminate the possibility of duplicate case names. Otherwise, an aircraft being involved in two different conflicts stored in the case-base would result in the same callsign becoming the name of two different cases in the "OneInOne" case representation.

HorConflConf

This parameter indicates the Horizontal Conflict Configuration and can have four different values: *head-on*, *converging*, *diverging* and *crossing*. The angle between the two vectors that represent the trajectory of the aircraft before entering the no-go zone is calculated. The angle is between the last waypoints before the no-go zone of the two trajectories and has as vertex the centre of the no-go zone.

If the angle between the two aircraft is bigger than `BiggestAngle` (defined in the header file to be equal to 155°), the value of HorConflConf is *head-on*. If the angle between these two vectors is smaller than `BiggestAngle`, ISAC checks if there is more than one point in common between the two trajectories. If there is only the conflict point in common between the two trajectories, the value for HorConflConf is *crossing* because the angle is already less than `BiggestAngle`. If there are two or more points in common, and if the common points are the last points of the flight plan, then the aircraft are *converging*, otherwise they are *diverging*.

The way the angle is acquired could change the final value. Earlier versions considered the angle between the two vectors whose extremes are the last waypoint on the flight plan before entering the no-go zone and the point where the trajectory crosses the border of the no-go zone.

AltitudeNow

This parameter indicates the relative altitude of the two aircraft. Its value can be *same* if between the two aircraft there is a difference in altitude smaller than 100 ft; it is *different*

otherwise. In an earlier version of the system, this parameter had the two values *higher* and *lower*, instead of the single value *different*, depending on which aircraft was at least 100 ft higher or lower than the other. Discussions with controllers showed that this distinction was not necessary.

AltConfiguration

AltConfiguration indicates the altitude profile of an aircraft. The three possible values are: *stable, climbing* and *descending*. The altitude of the aircraft is checked before entering the no-go zone and after exiting it. If there is a change in altitude bigger than 50 ft then the aircraft is either *climbing* or *descending*.

In earlier versions of the system, the parameter "SomebodyClimbing", extracted from "AltConfiguration", was used. It is not used any more because its information is redundant and implicit in "AltConfiguration".

Speed

This parameter depends on the relative speed between the two aircraft. If the first aircraft is faster that the second one by more than `SpeedDiff`, the "Speed" is *faster*. Vice versa for *slower*. If the two speeds do not differ by more than `SpeedDiff`, the value is *same*. All the speeds are converted into Mach. The value of `SpeedDiff` is 0.1 Mach.

CloseToTOD

This is a number expressing the distance of the aircraft from the destination airport in nautical miles. In earlier development steps, "CloseToTOD" was a symbolic parameter, with values *yes* and *no*, depending on whether the aircraft was closer than 100 nautical miles to the destination airport or not. The Top Of Descent (TOD) is usually 90-100 nautical miles from the destination and indicates the start of the descent to the airport.

CloseToBoundaries

This is a number that indicates the distance in minutes between the first point of the trajectory which is in the no-go zone and the entry or exit point in the sector, i.e., the points of the trajectory which are on the sector boundaries. The exit point has to be considered because a controller cannot manoeuvre an aircraft too close to the sector boundaries because he might need to co-ordinate with another sector, which would increase his workload. The entry point has to be considered also for the same reason.

The distance from the entry and exit sector boundaries respectively are calculated and the smaller time is kept. In earlier development steps, "CloseToBoundaries" was a symbolic parameter with values *yes* and *no*, depending on whether the smallest of the two calculated times was less than 4 minutes.

Manoeuvrability

This parameter used to depend on the percentage of accomplished trajectory and on the performance of the aircraft. The combination of the two gave the manoeuvrability of an aircraft. For example, an aircraft with good performance with a lot of fuel is not very manoeuvrable.

The *percentage* of accomplished trajectory was calculated when the co-ordinates of the actual position of the aircraft, the departure airport and the destination airport were known.

The *performance* was relative to the other aircraft involved in the conflict. An aircraft belonged to one of the following four empirical classes of aircraft: fighter, high performance, medium performance and low performance. An aircraft could have had *better*, *same* or *worse* performance than the other.

The manoeuvrability was *high* if the percentage of accomplished trajectory was bigger than 75% and the performance of that aircraft was *better* than the other. If either the performance was smaller than 75% or the performance was *worse*, the manoeuvrability was *low*, otherwise it was *medium*. If the percentage of accomplished trajectory could not be calculated, there was a direct correspondence between the performance and the manoeuvrability: *better* performance → *high* manoeuvrability, *same* performance → *medium* manoeuvrability and *worse* performance → *low* manoeuvrability.

The file with the look-up table for the type of aircraft and the correspondent performance was empirically built by a controller and reflected his preference. An extract of the hard-coded look-up table is shown below:

```
if((strcmp(type,"D328")==0)||    //if the type of the aircraft is either "D328"
   (strcmp(type,"AT42")==0)||    // or "AT42" or "FK27", then the
manoeuvrability
   (strcmp(type,"FK27")==0)||    // returned is "1", i.e. "low"
  return 1; // low
```

To solve this ad hoc and temporary situation, the BADA database (Bos, 1997) was used. In the final version of ISAC, the manoeuvrability is a numeric value, average of the maximum climb, cruise and descent Mach speeds of the aircraft. These are extracted from the BADA performance file, available for each type of aircraft. The percentage of accomplished trajectory has not been included yet in the final computation of this parameter.

Priority

A flight can be of different types: commercial, business, military, transfer or training. A commercial flight has the highest priority, a transfer and a military aircraft have the same lowest priority. The priority is *higher*, *lower* or *same* depending on the type of flight of both the aircraft in the conflict.

EasyToExitRight and EasyToExitLeft

These two parameters express how easy it is to exit the no-go zone by turning left or right. An angle, with vertex in the trajectory point immediately before the no-go zone, is calculated. This angle is the maximum of all the angles between the point on the trajectory in the centre of the no-go zone and all the points on the border of the no-go zone. This angle is called $\alpha_{conflict}$. At the same time, the angles generated by the no-go zones of the other aircraft in the environment are calculated. The minimum of all these angles is called $\alpha_{environment}$. If $\alpha_{conflict}$ is smaller than $\alpha_{environment}$, the value of the parameter is *difficult*. It is *veryEasy* if either the aircraft is already turning that direction and the angle is less than 10° or if the angle is less than 5°. It is *easy* if the angle is less than 10°, *possible* if the angle is between 10° and 15° and *difficult* if the angle is bigger than 15°.

LevelsAvailable

This parameter indicates which levels are available for the aircraft. If the aircraft is stable, the possible values are:

- *none*, if in each of the two levels above and below there is at least one no-go zone generated by another aircraft,
- *above*, if one of the two levels above is completely free,
- *below*, if one of the two levels below is completely free,
- *yes*, if there are any free levels above and below.

The “two levels above” refer to the level immediately above, even if it is reserved for the other direction, and the level above this.

If the aircraft is climbing or descending the possible values are:

- *none*, if none of the intermediate levels, the starting level and the final level are free,
- *yes*, if there is at least one level which is completely free,
- *spaces*, if there are no levels completely free, but there are some spaces between the no-go zones at some levels.

Faster and Slower

These two parameters indicate how easy it is to exit the no-go zone by increasing or decreasing the speed. All the speeds are converted into Mach and the altitude of the aircraft is supposed to be constant. All the border points of the no-go zone are taken into account and the maximum difference between the actual speed and the speed that correspond to the border points of the no-go zone in the speed display is calculated. If this difference is smaller than 0.1 Mach then the value is *easy*, if it is less than 0.2 Mach, the value is *possible*, otherwise it is *difficult*.

Agreements

This parameter indicates the agreements between the working sector and the next one. If the aircraft has a short window in time for the border crossing and a fixed exit level, the value is *sequencing*, otherwise it is *notSequencing*. This parameter is not yet used in ISAC because no data from the flight plan supplies this information.

Rules for Determining the First Aircraft

Whenever a conflict between two aircraft appears, a set of four rules decides which aircraft comes “first” and which “second” in the conflict description. This set is the result of an empirical process and depends on some of the parameters that describe the conflict. In the final version of ISAC the first aircraft is the one with the highest priority, i.e., the least likely to be manoeuvred. The four rules used are:

- if an aircraft is flying at a cruise level, it should not be moved from that level;
- if an aircraft is far from its destination it is heavy because of its fuel load, so it is less manoeuvrable;
- the aircraft with the worst performance is also the least manoeuvrable;
- a commercial aircraft should always have the fastest and least expensive route, if in conflict with a military, business, training or transfer aircraft.

These four rules are all considered at the same time and contribute with the same weight to the final decision.

4.4 Implementation Language

Because of the complexity of the system, the steps typical of the CBR process are executed by different functions that are integrated to minimise redundancy (i.e., loss of time and money) and to maximise efficiency. This has been achieved with information hiding and

modularity. In Julia (Hinrichs, 1988), a similar structure is implemented with a layered architecture. Inheritance is used to propagate some values to different objects of the same group.

Previous expert systems, like AIRPAC, were written in LISP, but because this language is too slow two solutions have been suggested:

- optimising the LISP code for speed or
- implementing the algorithms in a language faster then LISP (Shively and Schwamb, 1994).

The second option was taken when implementing ISAC. The most suitable language is C++. Firstly because C++, with its low level structure close to the hardware architecture, is the versatile and efficient. Furthermore, it automatically supports information hiding and inheritance. Finally, because C++ is an easily portable language and the same program can be run on different platforms without any changes (it will be shown later that this was not always true in our situation).

Using a portable language is important because the problem solver module is independent from HIPS and should be executed by "any available machine" (Meckiff and Gibbs, 1994), communicating with its host with standard protocols.

Simplifications

Because of the complexity of the domain and because ISAC is still a prototype, a lot of simplifications have been made. They will be highlighted in the relevant sections, mainly in Chapter 5 where the CBR issues are treated.

Even if all the data for the conflict description is available, procedures for the treatment of "don't care" and "don't know" values have been developed.

4.5 Summary

The technical description of ISAC and of the system in which it is embedded is given in this chapter. It is explained how it works, how it is interfaced and the main structures and functions used. Its internal architecture, the functions and the classes that constitute the core of ISAC are described in more detail. Some results are shown to prove that the spreading activation retrieval mechanism gives the same results but is faster than the flat search retrieval mechanism. Finally, the acquisition of the parameters has been discussed and the different possibilities of acquisition are analysed. All the parameters used in the final version of the case-base have been listed. We have explained why some old parameters are not used anymore and the knowledge engineering problem of changing the parameter that describe a case is highlighted.

Chapter 5

CBR Issues

In the previous chapter the technical characteristics of the system have been examined, whereas in this chapter some theoretical issues inherent to the CBR domain will be treated in more detail. It is explained how the case representation with the possible solutions and first of all the case structure have been influenced by the nature of the task. The problem of reducing the size of a potentially huge case-base and the need of reusing cases justify the introduction of three different case representations whose advantages and disadvantages are explained. The issue of deciding whether to use gold standard cases or specific cases is presented.

The possibility of deciding which are the most important parameters using either decision trees or the information content of each parameter is analysed. A hierarchical CBR structure is suggested for the solution of more complex air conflicts. Three possible architectures are analysed and one of these will be actually implemented for the resolution of multiple aircraft conflicts and evaluated. Finally, case adaptation is treated and we explain why the simplest type of case adaptation is effective enough for ISAC.

5.1 Case Representation

Most of the initial development effort in ISAC was focused on case representation which is typically the first step and main issue in the construction of an intelligent assistant. The evaluation of the domain presented two problems: the macro problem of what should constitute a case and the micro problem of how to characterise a case.

The construction of the system was characterised by the difficulty in determining the correct parameters, where “correct” means capable of describing exactly what the controller perceives on the radar screen. Moreover, the correct parameters having been identified, they have to be correctly acquired from the data available in the environment. Determining the correct conflict representation has involved extensive dialogues with ATC controllers and then the manipulation of the data available from HIPS.

As shown in Figure 4.2, when HIPS detects a conflict, it passes its representation to ISAC. All the data concerning the conflict that is available in HIPS is converted into parameters useful for the case representation. The conversion process eliminates useless data and transforms other data into more abstract and complex parameters. For example, the number of passengers on an aircraft is discarded whereas data that is otherwise meaningless, such as the co-ordinates of the no-go zones, becomes useful if related to the aircraft trajectory.

In a future scenario, more information will be made available provided by increasingly precise and intelligent instruments. Moreover, datalink will improve the accuracy of the manoeuvres available. Nowadays, the controller cannot ask to the pilot to accomplish a very accurate manoeuvre. For example, if it is extrapolated from HIPS that the aircraft must turn 17° to the right to exit the horizontal no-go zone, the aircraft will have to turn at least 25° to safely avoid any uncertainty. When a datalink connection between the control tower and the aircraft becomes available, a "17°" manoeuvre will be possible and methodologies of solving conflicts will change radically.

The process of determining the parameters was iterative and the selection of new parameters was driven by the analysis of errors at each iteration. The different versions of the case representation are shown in Chapter 6. The difficulty of determining a comprehensive set of important parameters from dialogues with the controllers is exacerbated by considerable differences in how individual controllers view and solve conflicts. An example of these differences is demonstrated in the use of speed change as a solution to a conflict. As it is known, HIPS provides a display showing how easy it is to avoid the conflict by changing speed. Some controllers would never change the speed of an aircraft which is climbing or descending even if HIPS indicates that it would be a very good solution for both the controller (easy to implement) and the aircraft (time and fuel gain). On the other hand, other controllers are not put off by the fact that an aircraft is climbing or descending and they trust HIPS by giving a speed solution even if it did not occur to them at first.

Another example that shows how differently a conflict can be represented in the controller's head is the concept of one aircraft "passing in front" of the other when there is a "crossing" conflict. For some controllers it is an important issue, whereas for others it has no importance at all.

A lot of effort has gone into trying to show to all the controllers the same environment tools and the same set of conflicts in order to reduce any discrepancy in the resolution. In the end there is a compromise between what is considered an important criterion and what can be

extracted from the geometric information from HIPS. For the different ways of acquiring the case-base, see Chapter 6 and for the description of how the case-base has been acquired in practice, see Appendix A.

5.1.1 Case Space Coverage

The case space is the set of all the possible cases that could constitute the case-base and its dimension depends on the parameters used to describe a case and their possible values. To have a rough idea of how many unique cases there are in the case space it would be enough to multiply together the number of possible values of each parameter. This is only possible if all the parameters are symbolic. Further, some cases produced in this way may not occur in practice.

To study case space coverage means to understand whether a case-base has enough cases and whether they are representative enough to obtain an accurate solution. To have an effective system, the case-base should be well covered, which does not mean that the case space should include all possible cases, but at least those cases which are "pivotals" where "a case is pivotal if its deletion from the case-base directly reduces the competence of a system" (Smyth and Keane, 1995).

Two parameters that could help in the visualisation of the case space in order to indicate whether it is well covered or not are AVE and SMA. AVE indicates the AVErage distance in term of similarity of a case from all the other cases. SMA is the average of the SMAllest distance of a case from all the others.

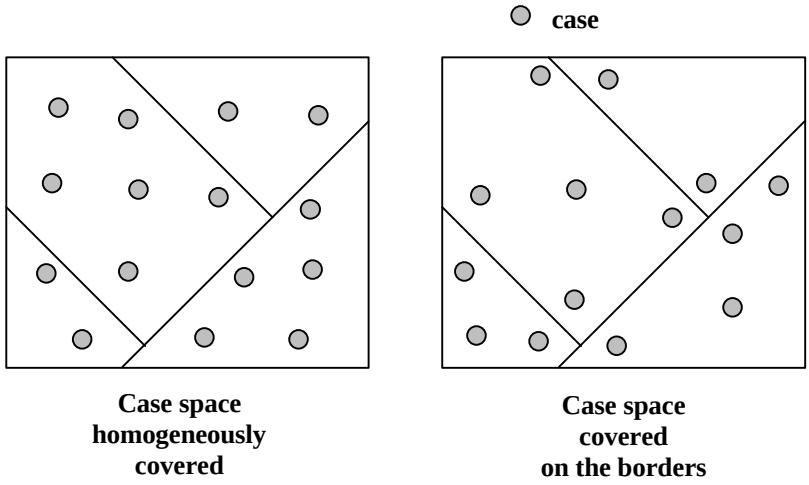

Figure 5.1: *Different types of case space coverage.*

With these two parameters it is possible to calculate which zones of the case space are not well covered by finding the cases that are furthest in terms of similarity from any other and to add these cases with the right solution to the case-base. The problem that arise, while

using the two parameters AVE and SMA, is that they only indicate whether a case space is *homogeneously* covered and this does not necessarily indicate that the case-base contains all the pivotal cases. Usually the case space must be well covered first of all on the border of the zones where the cases change solutions, as shown in Figure 5.1.

5.1.2 Gold Standard Cases versus Specific Cases

In the 1996 European Workshop on Case-Based Reasoning, two different points of view on how a case-base should be covered were suggested: Michael Manago suggested that a case-base should contain few clean cases; on the other hand, David Waltz suggested that in a case-base there should be a lot of noisy cases. During the development of ISAC both the alternatives have been tried.

The first approach to the construction of the case-base implied the use of prototypical cases, i.e. very general cases, with their ideal solutions decided by a team of controllers. This case-base should have been able to give solution to conflicts appearing in any sector and these cases were called gold standard cases.

Further steps in the knowledge engineering process showed that this hypothesis was too optimistic and that a lot of conflicts with the same description had differing solutions due to their location in different sectors. This is because there are some parameters that are sector dependent and hence cannot be stored in the case-base.

For this reason the choice of gold standard cases valid for any sector was abandoned in favour of a more realistic case-base which focused on a particular sector. This required that cases be recorded from a sector and solutions be generated by controllers that usually work on that sector. By doing this, the effects of the “forgotten” parameters that depend on the sector are minimised.

The concept of the gold standard cases can be reintroduced if the system is used for training or teaching purposes. In this situation the case-base can consist of gold standard cases whose ideal solutions are those taught to controllers.

The problem of different controllers having different solutions to exactly the same conflict in the sector is still relevant as it can be seen in Figure 8.3(a), where the solutions given by different controllers to the same conflicts are confronted. Assuming that all the controllers that have been trained in the same sector will give coherent solutions is a big issue and will be treated in more detail in the next chapters.

5.1.3 Solution Representation

The solution granularity required of the system is the choice of the aircraft and the type of manoeuvre. In a two-aircraft conflict either the first or second aircraft or both of them can be manoeuvred. The aircraft can be manoeuvred in altitude, in speed or horizontally. In the first steps of the knowledge engineering process, nine possible solutions have been identified. These have been labelled “alt1”, “alt2”, “alt3”, “spe1”, “spe2”, “spe3”, “hor1”, “hor2” and “hor3”. Where *alt*, *spe* and *hor* stand for altitude, speed and horizontal manoeuvre respectively which can be applied to either the first (1), second (2) or both aircraft (3).

In the last step of the knowledge engineering process, the altitude manoeuvre *alt* was substituted by the more specific climb solution, *upp*, and descent solution, *dow*. With this introduction the possible twelve solutions are: “upp1”, “dow1”, “upp2”, “dow2”, “upp3”, “dow3”, “spe1”, “spe2”, “spe3”, “hor1”, “hor2” and “hor3”. These solutions are used in all the case representations and they can be combined together when the solution to a conflict is complex. For example, a speed manoeuvre combined with a gentle horizontal manoeuvre might solve the conflict better than a sharp horizontal manoeuvre alone.

A horizontal manoeuvre implies turning right/left, a direct route to destination or a parallel heading with the other aircraft. The horizontal manoeuvre does not specify whether the aircraft has to turn right or left or the number of degrees. A manoeuvre with the “3” suffix means that the manoeuvre can be applied either to both aircraft at the same time or to each individually because the aircraft have exactly the same priority.

The manoeuvre suggested by ISAC should be the “best” manoeuvre for both controllers and pilots, but because the case-base contains solutions given by controllers, it is more likely that the controllers will be more satisfied than the pilots.

Usually when the controller’s workload is too high, the solution tends not to be very convenient for the pilot because the controller has no time to decide on the most economical solution for the aircraft. On the other hand, when the workload is low the controller has time to come up with a better solution that may need more monitoring but is less time and fuel expensive for the aircraft. The safest manoeuvre is an altitude manoeuvre and that is why ISAC specifies more precisely the altitude manoeuvre. Figure 8.3(a) shows that this is the manoeuvre most used by the controllers.

The policy for deciding the final solution when the retrieval process gives several cases with different solutions is still not completely defined. With the “TwoInOne” case representation there could be a number of cases which are similar to the target. In this situation the most

commonly occurring among the retrieved solutions becomes the final solution. With the "OneInOne" case representation there is one target for each aircraft and for each target there is a list of the most similar cases. For both targets the most common solution is extracted, then the two solutions are examined and a single coherent solution is extracted.
The solution for a multiple aircraft conflict is not the same as for a two aircraft conflict. The format is: manoeuvre + name of the aircraft. The four possible manoeuvres are the same as for a two aircraft conflict and a solution can be composed of more than one manoeuvre applied to different aircraft.

5.1.4 Meaning of NIL Values

The NIL value of a parameter has two different meanings depending on the environment. If a NIL value appears in the case-base it means that the value of the parameter is "don't care". On the other hand, if a NIL value appears in a target it means that the parameter is "not known". In the particular situation of ISAC, the case-base should not contain any unknown values because all the necessary parameters are available from the simulation instruments.

How NIL Values are Treated During Retrieval

Quinlan (1993) suggests some possibilities for the treatment of unknown values depending on the context: the use of the most probable value; the extrapolation of the value depending on the context or the use of probabilities.
Originally, when either a numeric or symbolic parameter with NIL value was encountered during the Spreading Activation process, its activation was incremented by 1, as if the conflict parameter's value was the same as the target's. This is because the case could possibly be a good solution for the target depending on the other parameters. On the other hand, if the NIL value was in the target its activation would have not been increased to avoid the risk of having too many retrieved cases at the end of the retrieval process.
When a different method of case acquisition was used, the policy for dealing with the NIL values had to be changed. The new case acquisition consisted in building by hand a set of representative cases instead of acquiring the cases directly from the traffic samples, this operation being too time consuming. In this new case-base a lot of parameters had NIL values and the above policy was not sufficiently discriminating. In the new policy the activation of a NIL parameter is kept at zero and the final activation of each case is weighted with the number of non NIL parameters in the case. With this policy the maximum final activation of a case will be 1 when all the non NIL values of the case are the same as

the target. Again, this policy is valid for both symbolic and numeric values. A simplified version of this policy is to simply ignore the NIL value without counting the number of parameters that have a non-NIL value. This is the policy adopted in the final version of ISAC.

5.2 CBR versus Decision Trees

In a decision tree the parameters are ordered from the root of the tree, the most discriminatory level, to the leaves, the least discriminatory level. The tree is built from a set of cases whose solution is known. This is called supervised learning because the solutions are given beforehand. Naturally, cases with the same parameter values that have different solutions cause a problem of incoherence. The four steps to building a decision tree for a given case-base are (Quinlan, 1986):

- extraction of a subset of cases;
- construction of the decision tree for the extracted subset;
- classification of the cases that were left out of the subset with the decision tree;
- addition of the cases that were not classified correctly to the subset and reconstruction of the decision tree.

These steps have been implemented in C4.5 (Quinlan, 1993) and produce one of the possible decision trees with the certainty that it works and is the simplest. ISAC has been tested in comparison with C4.5 because decision trees could be useful in deciding which parameters are non-redundant. The test, described in Appendix B, has been carried out with one of the first versions of ISAC but the results can be generalised for all the versions of ISAC because no big structural changes have been introduced afterwards.

Table 5.1 shows the results of the experiments done with the “LeaveOneIN” (the target is left in the case-base) and “LeaveOneOUT” (the target is taken out of the case-base) with C4.5 and ISAC. It can be seen that ISAC performs slightly better than C4.5. No tests have been made with the new version of Quinlan’s program C5.0 (Quinlan, 1997) even if its performance might have been better than C4.5 because this new version includes support for boosting.

Table 5.1: *Decision trees versus case-based reasoning.*

	LeaveOneIN	LeaveOneOUT
C4.5	82%	71%
ISAC	97%	73%

5.2.1 P-tasks and S-tasks

Comparing the performance of different learning algorithms is quite a common exercise. On the other hand, an uncommon approach is to explain the performance of a system not with the type of algorithm used but with the type of the task and the knowledge base used.

In (Quinlan, 1994), two types of tasks are identified: parallel and sequential tasks (P-tasks and S-tasks). In a P-task, the output depends on the value of all the input parameters and these values are examined simultaneously. In an S-task, the parameters are examined sequentially and not necessarily all the values have to be given to reach the solutions.

Some learning algorithms have a strictly parallel approach to the task, while some others have a typically sequential approach. For example, a P-task will be solved easily by a neural network because all the inputs are processed at the same time whereas an S-task will be more easily solved by a decision tree. CBR can easily solve both S-tasks and P-tasks, even if it is closer to a parallel algorithm.

From the fact that ISAC performs better than C4.5, it can be argued that the task of conflict resolution is essentially a P-task. This view is supported by conversations with air traffic controllers in which the “global” view of the conflict is considered essential for its good solution.

5.2.2 Discriminatory Power

ISAC gives the possibility of calculating the discriminatory power of the parameters that are used in the case description. This function, independent from the *k*-NN retrieval mechanism, can be used off-line to enhance the knowledge engineering process because it helps in better understanding the parameters.

The mechanism of selecting discriminatory parameters is best explained in terms of building a decision tree that has leaf nodes corresponding to the different diagnoses **D**. The set of cases **C** is then located, or classified, on these nodes. It is important that the tree is in some sense minimal so the choice of which parameter to test at any level of the tree is critical. In ID3 this is done by selecting parameters based on their information content or discriminatory power (Quinlan, 1986). The process used in ISAC is similar to that in ID3 except that the semantics of the branching in the decision tree is slightly different because of the possibility of unknowns in the case parameters. A brief explanation of how the discrimination works is as follows:

D=$\{D_1,\ldots,D_d\}$ is the set of possible classes or diagnoses;

C=$\{C_1,\ldots,C_c\}$ is the set of cases to classify;

F=$\{F_1,\ldots,F_f\}$ is the set of expensive parameters, one of which is selected at each decision point.

The set of cases can be seen as an information source producing one of *d* messages from the set **D**. Let $|D_j|$ represent the number of cases with diagnosis D_j. Then the expected information needed to generate the appropriate message is:

$$I\left(\frac{|D_1|}{|D_1|+\ldots+|D_d|},\ldots,\frac{|D_d|}{|D_1|+\ldots+|D_d|}\right)=-\sum_{j=1}^{d}\left(\frac{|D_j|}{|D_1|+\ldots+|D_d|}\cdot\log_2\left(\frac{|D_j|}{|D_1|+\ldots+|D_d|}\right)\right)$$

Consider the complete set of matching cases (see Figure 5.2). Assume that the parameter $F\in\mathbf{F}$ is tested and that this parameter has possible values $\mathbf{V}=\{V^1,\ldots,V^n\}$. Then **V** partitions **C** into *n* groups of cases, $G^1,\ldots,G^n$; where G^i contains those cases that have value V^i for parameter *F*.

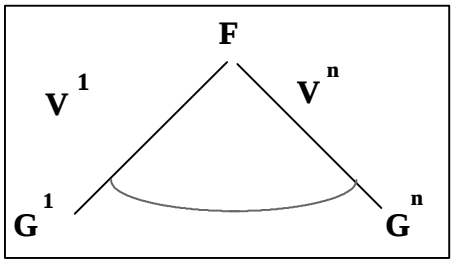

Figure 5.2: *The root classification of the cases in* ***C****.*

Let G^i contain $|D^i_j|$ cases with diagnosis D_j, that is $|D^i_j|$ instances of class D_j. The probability of a case belonging to G^i is (i.e. probability of a case having the i^{th} value for attribute *F*):

$$\frac{|D^i{}_1|+\ldots+|D^i{}_d|}{|D_1|+\ldots+|D_d|}$$

So after testing *F* the remaining information associated with the subsets, $G^1,\ldots,G^n$ is:

$$\text{Remainder}(F)=\sum_{i=1}^{n}\left(\frac{|D^i{}_1|+\ldots+|D^i{}_d|}{|D_1|+\ldots+|D_d|}\right)\cdot I\left(\frac{|D^i{}_1|}{|D^i{}_1|+\ldots+|D^i{}_d|},\ldots,\frac{|D^i{}_d|}{|D^i{}_1|+\ldots+|D^i{}_d|}\right)$$

The weight of the i^{th} subset is the proportion of cases in **C** that belong to G^i. The information gained from using *F*, or the *discriminatory power* of *F*, is:

$$DP(F)=I\left(\frac{|D_1|}{|D_1|+\ldots+|D_d|},\ldots,\frac{|D_d|}{|D_1|+\ldots+|D_d|}\right)-\text{Remainder}(F)$$

Thus the parameter that leaves the smallest remainder is the most discriminating. So, at each stage in the reduction of the set of cases, the most discriminating parameter is selected using this criterion. The user is requested to determine the value of this parameter for the target case. The cases in the candidate set that *cannot* match on this parameter are removed from

the retrieved set. This process is repeated until the set reduces to one diagnosis or the target case proves to be dissimilar to all the retrieved cases. This technique has proved remarkably successful for retrieving good matches while requiring a minimum number of expensive parameter values (Cunningham, Smyth and Bonzano, 1998). The discriminatory power depends on how specialised the solution of the cases are. The basic information formula given in (Quinlan, 1986) to calculate the discriminating power of the parameters involved in the case description is:

$$I(p,q) = -\frac{p}{p+q}\cdot\log_2\left(\frac{p}{p+q}\right) - \frac{q}{p+q}\cdot\log_2\left(\frac{q}{p+q}\right) \qquad (5.1)$$

where *p* and *q* are the probability that a case gives solution P and Q. Moreover: $p+q=1$. From the above formula it can be seen that if $p=0$ then $I(p,q)=0$ because all the cases will have solution Q. If $p=q=\frac{1}{2}$ then $I(p,q) = -\log_2\left(\frac{1}{2}\right) = 1$.

The Formula (5.1) deals only with cases with only 2 possible solutions but it can be extended to any number of possible solutions (Levine, 1971 and Nosal, 1977). For example, with three possible solutions, the information formula becomes:

$$I(p,q,r) = -\frac{p}{p+q+r}\cdot\log_2\left(\frac{p}{p+q+r}\right) - \frac{q}{p+q+r}\cdot\log_2\left(\frac{q}{p+q+r}\right) - \frac{r}{p+q+r}\cdot\log_2\left(\frac{r}{p+q+r}\right)$$

The information properties are still valid: if $p=0$ then $I(p,q,r)=I(q,r)$ returning to Equation (5.1) again. If $p=0$ and $n=0$ then $I(p,q,r)=0$. If $p=q=r=\frac{1}{3}$ then $I(p,q,r) = -\log_2\left(\frac{1}{3}\right)$ which would equal 1 if $\log_3$ is used instead of $\log_2$. In general, if there are *n* possible values, $\log_n$ should be used to keep the value of maximum information always at 1.

If the list of the most discriminatory parameters generated by C4.5 is confronted with the list obtained with ISAC's algorithm some discrepancies are evident. The root of the decision tree generated by C4.5 is the most discriminatory parameter which is different from the most discriminating parameters in the list generated by ISAC, as seen in Appendix B. This is because, as already stated, ISAC and C4.5 calculate the discriminatory power with two slightly different algorithms. The algorithm in C4.5 is more specific whereas the one used in ISAC is more general. C4.5 calculates the information carried by each parameter then it weights this value depending on the possible values that the parameter can have. ISAC's algorithm strictly calculates the information. Having the plate number of a car as a

parameter, for example, is very discriminatory because when the plate number is known the car is uniquely identified. On the other hand, the information carried by the plate number of a car is very little because there are so many different plate numbers. In this situation, C4.5 would consider the plate number very discriminatory, whereas ISAC would not.

Some more considerations on the weights of the parameters and the possibility of changing the weights to improve the performance of the system are in Chapter 7 where introspective learning of parameters weight is analysed.

5.3 Case Structure

The motivation behind the development of ISAC is to reduce the decision making burden on controllers in order to support operation in situations of increased traffic. This future scenario also implies more complex conflicts involving more than two aircraft. A key design criterion has been to develop a case representation that is extendible from two aircraft conflicts to conflicts involving three or more aircraft. This militates against having a single conflict as the basic unit of retrieval. For reasons of economy in case coverage, the solutions for two-aircraft conflicts should be reusable in multiple aircraft conflicts. To do so, a conflict should be decomposable so that the basic unit of retrieval is an individual aircraft in a conflict.

This problem of representing situations involving two conflicting entities has already been faced in the CBR literature, for example in two classical systems, Mediator (Simpson, 1985) and Persuader (Sycara, 1987), and more recently in Truth-Teller (Ashley, 1995). In these systems, perhaps because they describe interaction between humans, there is a vocabulary to characterise the “type” of conflict and this is critical in determining the solution. In the ATC domain the situation is different because the solutions depend on the arrangement of the aircraft and the context of the individual aircraft as described by their flight plans. The conflict between two aircraft can be described roughly with one or two global parameters but the final solution depends on a lot of dependent variables related to a single aircraft. For this reason the approach adopted in ISAC is somewhat different to the above systems, with an emphasis placed on the parameters that describe the aircraft on its own.

While our ultimate objective in developing ISAC is to have a single aircraft as the unit of case retrieval we have considered three case organisations in detail. Two different case representations were adopted and tested.

- The first option was to create one case for each conflict, with the description of both the aircraft in the same case. This option will be referred as “TwoInOne”, because two aircraft are in one case.
- The second option, referred to as “OneInOne”, was to create two separated cases for each conflict, each one with the description of one aircraft.

Tables 5.2 and 5.3 show the two possible case descriptions.

Table 5.2: *A conflict expressed in the “OneInOne” case representation.*

Casename	Case690(A)	Casename	Case690(B)
HorConflConf	crossing	HorConflConf	crossing
AltitudeNow	same	AltitudeNow	same
AltConfiguration	stable	AltConfiguration	stable
Speed	faster	Speed	slower
CloseToTOD	155	CloseToTOD	352
CloseToBoundaries	4.8	CloseToBoundaries	8.3
Manoeuvrability	.78	Manoeuvrability	.78
Priority	same	Priority	same
EasyToExitHorizontally	easy	EasyToExitHorizontally	possible
LevelsAvailable	yes	LevelsAvailable	yes
Faster	difficult	Faster	difficult
Slower	difficult	Slower	difficult
Solution	dow1	Solution	dow2

Table 5.3: *A conflict expressed in the “TwoInOne” case representation.*

Casename	Case690
HorConflConf	crossing
Priority	same
AltitudeNow	same
Speed	faster
AltConfiguration(A)	stable
CloseToTOD(A)	155
CloseToBoundaries(A)	4.8
Manoeuvrability(A)	.78
EasyToExitHorizontally(A)	easy
LevelsAvailable(A)	yes
Faster(A)	difficult
Slower(A)	difficult
AltConfiguration(B)	stable
CloseToTOD(B)	352
CloseToBoundaries(B)	8.3
Manoeuvrability(B)	.78
EasyToExitHorizontally(B)	possible
LevelsAvailable(B)	yes
Faster(B)	difficult
Slower(B)	difficult
Solution	dow1

5.3.1 The Canonical Form for Two-Aircraft Conflicts

Storing the description of the two conflicting aircraft in the same case is the most obvious choice because it reflects the controller’s way of examining a conflict, but it presents two problems: first, this case representation is not easily extendible to multiple aircraft conflicts, second, it has to be decided which aircraft comes first in the conflict description. This problem can be explained with an example.

Let us suppose that a conflict between two aircraft A and B is stored in the case-base in the form A-B. If the same conflict has to be solved again, HIPS will send again the description

of A and B to ISAC. ISAC could build either target A-B or, inverting the order, target B-A. If the latter happens, the probability of finding the correct case A-B in the case-base is very low.

An obvious but time and space consuming solution to this problem would be to build the two cases A-B and B-A for each conflict involving A and B. The case-base will be twice the normal size and the retrieval time will double.

Alternatively, the two targets X-Y and Y-X could be built for each conflict between X and Y and the retrieval process has to be repeated once for each target. This means a doubled retrieval time but no increase in the case-base dimension. The advantage of both these solutions is that there is no loss of knowledge. The first option is referred in the experiments as “TwoInOne.nonCanonical”.

An alternative solution is to produce a set of rules to decide which is the first and which is the second aircraft in the case description. These rules have to be used during the construction of the case-base and every time a new target problem is presented. A case filtered by these rules is said to be expressed in the “canonical form”. The advantage of this process is that neither the retrieval time nor the case-base dimension is increased. The disadvantage is a possible loss of information as can be seen from the results of the experiments. This option is referred to as “TwoInOne.canonical”. The rules for the decision of the canonical form are described in Section 4.3.

The two “TwoInOne” case descriptions are derived from the “OneInOne” case description. The only new parameter in the “TwoInOne” description was, at the beginning, “Similar”. If the four rules indicated that both aircraft could come first in the conflict description the value of the parameter “Similar” was “yes”. The utility of this parameter, redundant because extracted from other parameters, was not proved. In fact, it has been shown that the use of this parameter led to a decrease in performance (Bonzano, Cunningham and Meckiff, 1996).

In the “OneInOne” conflict representation the information about the other aircraft involved is implicit in the environment description in the form of no-go zones. This suggests a Hierarchical CBR structure (Smyth and Cunningham, 1992) where problems are represented by multiple cases. This has the big advantage that the number of aircraft that can be involved in a conflict is not limited to two. Moreover, the problem of deciding which aircraft is first is avoided. However it is more difficult to come up with a set of parameters that can capture all the details.

5.4 Hierarchical CBR for Multiple Aircraft Conflicts

In (Shively, 1984), three types of conflict sets have been identified as being the most common:

- one versus one: the two conflicting aircraft are isolated from other conflicts;
- one versus two: two separated conflicts sharing a common aircraft;
- three-at-once: three conflicts among three aircraft.

The structure of ISAC presented up to now is able to solve conflicts belonging to the first category: two aircraft conflicts (TACs). The problem of multiple aircraft conflicts (MACs) is treated in this section.

Usually, in a TAC the aircraft that is moved is the one that will have the smallest delay. In a MAC the situation is more complex. If a MAC is decomposed into TACs, there is the risk of solving the wrong pair first. An overall view is necessary to decide which aircraft has to be manoeuvred even if some old expert systems produced acceptable results with myopic strategies explained later.

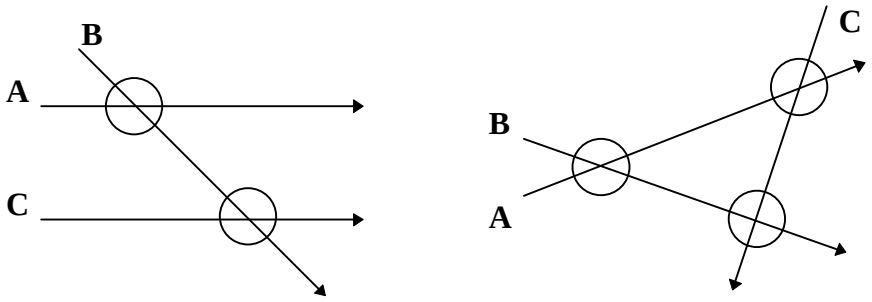

Figure 5.3: *Types of Multiple Aircraft Conflicts.*

A MAC involving *n* aircraft can be of two types: *simple* MAC and *complex* MAC. In a *simple* MAC all the *n-1* conflicts are generated by the same aircraft. On the other hand, in a *complex* MAC the conflicts are generated by different aircraft and there are at least *n* conflicts.

The Point of View of the Controllers

The different approaches to conflict resolution typical of each controller become even more evident when the conflict is a MAC. Some controllers consider only the *complex* MAC to be a "real" MAC. A *simple* MAC is only seen as a succession of TACs which are more or less interdependent.

When a *complex* MAC is decomposed into TACs, the TAC closest in time is selected. When a *simple* MAC is decomposed the aircraft that is in conflict with all the others is selected and the simplest or most desirable manoeuvre for this aircraft is chosen.

Other controllers, when solving a MAC, examine the flight plans of all the aircraft involved, then try to draw up a list of priority parameters such as: the aircraft with the longest distance to cover, the sector exit co-ordination, the impact that a level change or a course change to an aircraft would make to the other aircraft. If no particular priorities are found, then the conflict that is closest in time is solved.

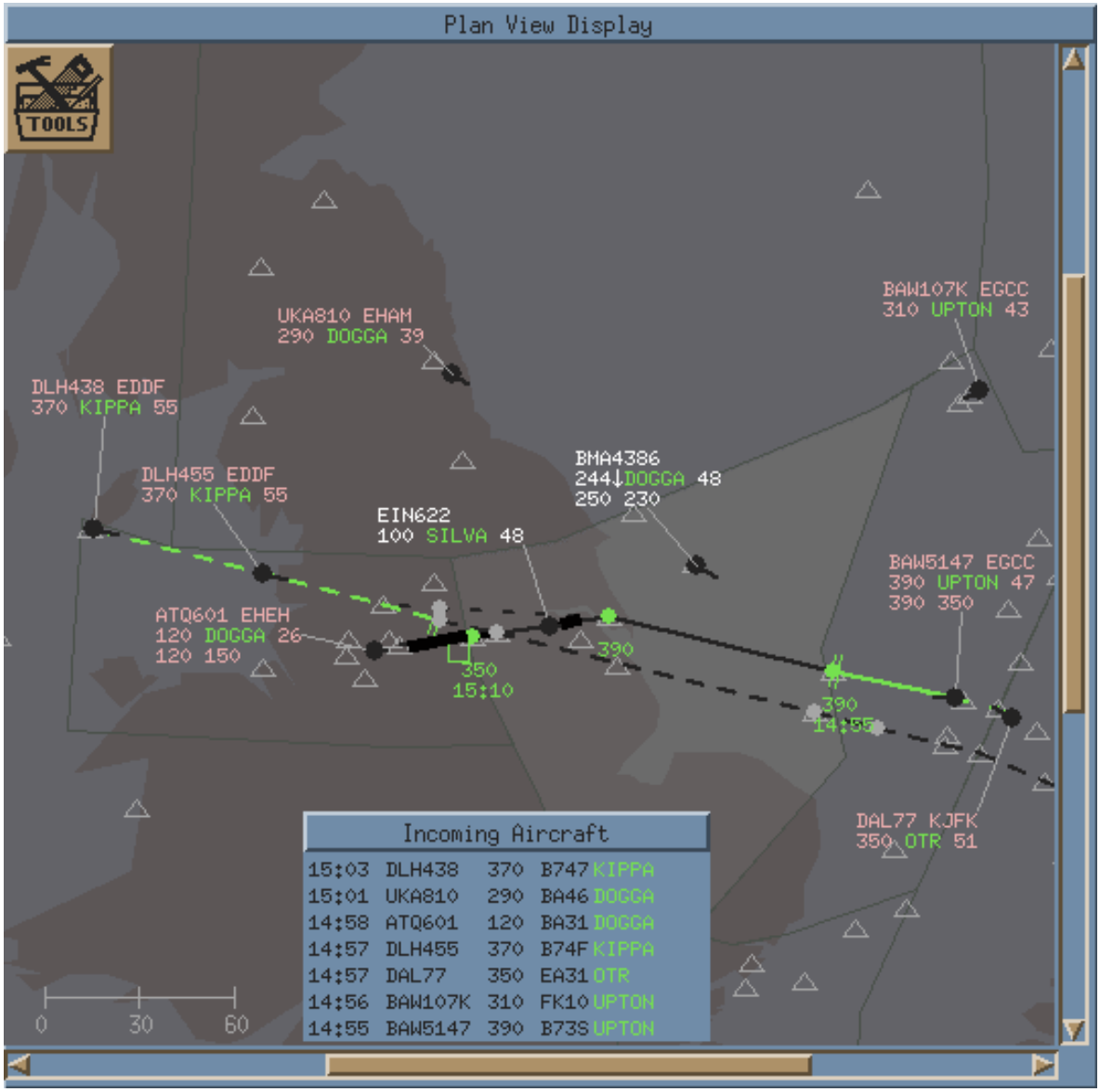

Figure 5.4: *A simple MAC.*

Independent of the method of solving the conflict, the aim of the solution is obviously the same as for TACs: the controller has to try to minimise the penalty that the solving manoeuvre will cause to the flights concerned. An evaluation of the current workload is also a determining factor for the final decision. A complex vectoring (i.e. a sequence of horizontal manoeuvres) situation may be the best solution, but a simple level change would involve far less work and concentration. Sometimes, conflicts that are distant are not solved because the situation may evolve in such a way that the conflict disappears due to altered

aircraft performance, request for a reclearance from a pilot or another conflict involving one of the original aircraft etc.

An example of a *simple* MAC is shown in Figure 5.4. The aircraft BAW5147 enters the sector at flight level 390 then it descends to level 350. While descending it gets into conflict with aircraft DLH438 coming from the opposite direction at level 370. While at level 350, BAW5147 conflicts with aircraft DAL77 that was already stable at level 350. The solutions to this conflict could be a composite manoeuvre consisting of an early descent for BAW5147 and a turn to the right for DAL77.

Hierarchical CBR

The straightforward approach to the solution of MACs would be the creation of a new case-base containing complex aircraft conflicts. This approach cannot be easily implemented because a MAC can involve 3, 4 or more aircraft and it is not possible to build a coherent structure for each possibility. Moreover, since a well covered case-base for TACs is already very big, the case-base for MACs would be larger still, making it impossible to build it in reality.

An alternative to the straightforward approach is a hierarchical structure. Three hierarchical structures for the solution of the MAC are suggested: Independent CBR, Look ahead CBR and Hierarchical CBR structure.

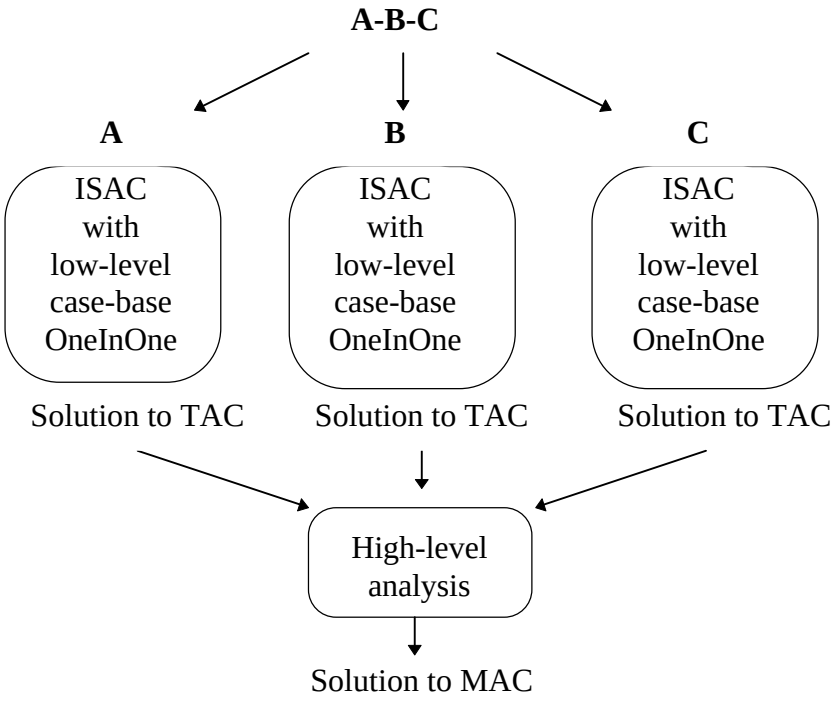

Figure 5.5: *Independent CBR.*

5.4.1 Independent CBR Structure

Let us suppose that the 3 aircraft A, B and C are involved in the MAC A-B-C where the two TACs are A-B and A-C. The considerations valid for this *simple* MAC are valid even for a *complex* MAC. As said in Section 5.3, with the "OneInOne" case representation an independent case is created for each simple aircraft involved in the conflict. No track is kept of the two TACs A-B and A-C because the conflicts are represented with no-go zones for each aircraft. This means that the MAC is not decomposed into TACs. ISAC solves the conflict for each of the aircraft involved in the MAC. The solutions found for each aircraft are then confronted and a common solution for the MAC is extracted. This structure is shown in Figure 5.5 and the name "Independent CBR" comes from the fact that the aircraft are described in independent cases.

5.4.2 Look Ahead CBR Structure

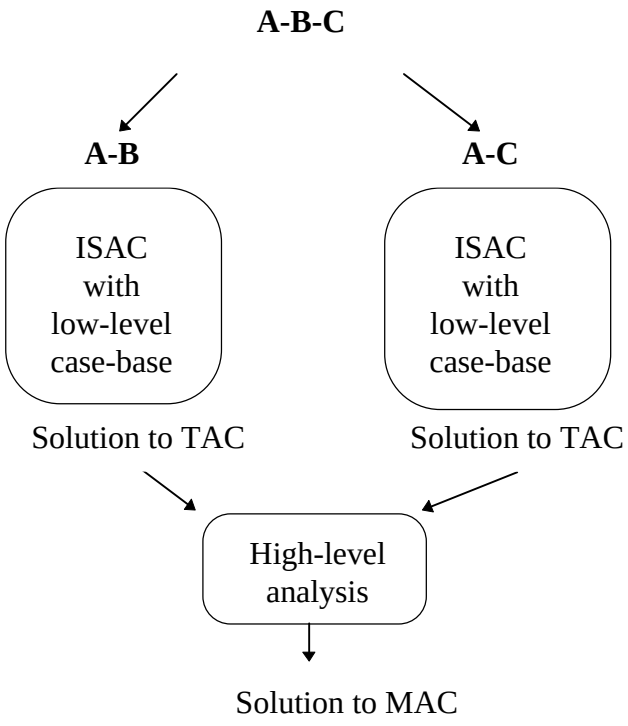

Figure 5.6: *Look Ahead CBR.*

With this structure, the MAC A-B-C is decomposed into the two TACs A-B and A-C which are solved separately by the system either with the "TwoInOne" case representation (canonical or non-canonical) or with the "OneInOne" case representation. Some heuristic rules are necessary to combine the solutions to the TACs into a coherent solution for the MAC. It should be noted that in this structure the "OneInOne" case representation is used to solve the TACs separately, whereas in the Independent CBR structure the same case

representation is immediately used to solve the MAC conflict. Figure 5.6 shows how the Look Ahead CBR structure works.

5.4.3 Hierarchical CBR Structure

This structure is the most abstract and the one that brings the biggest changes to the original structure of ISAC. The MAC A-B-C is examined at a high level to see if it is possible to immediately find a solution. A new high level case-base must be introduced for this first step. If no immediate solution is found, the high level case-base introduces some constraints or new parameters that are then used in the next step where the low level case-bases for the TACs are used. Again, the solutions found for the TACs have to be filtered to give a coherent general solution.

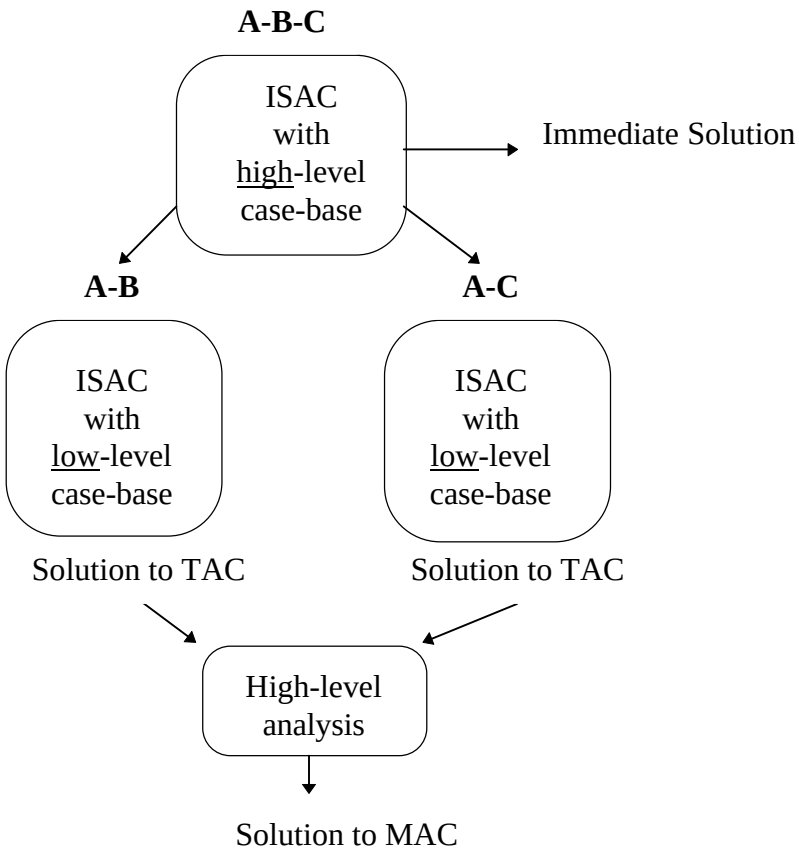

Figure 5.7: *Hierarchical CBR.*

One of the disadvantages of using a hierarchical approach is that a high level case-base becomes necessary and this case-base has to be built from scratch. Some of the parameters that might be used in the high level case-base are:

- geometrical description of the conflict (vertical view). Possible values for this parameter could be: all same level, one climbing and others stable, one descending and others climbing etc.

- geometrical description of the conflict (horizontal view). Possible parameters: two crossing, two catching up and one crossing etc.
- Is there an aircraft common to all the conflicts? (i.e., is it a *simple* MAC or a *complex* MAC?).
- If yes, some data about the aircraft which is in conflict with all the others.

The output that the case-base will give, as seen in Figure 5.7, is either the solution to the MAC or some extra constraints that can be used by the low level case-bases.

High-level analysis

In Figures 5.5, 5.6 and 5.7 the last step before the final solution has been named "high level analysis". This analysis is necessary to extract a coherent global solution from the solutions to the simple TACs. An example of a "cheap" analysis is to choose the solution of the TAC that has been retrieved with the highest activation as the solution for the MAC. In this situation the drawback is that the general view of the conflict is not taken into account.

Another example of analysis is that used in AIRPAC which chooses the first conflict in order of time and applies that solution. AIRPAC first looks for a rule able to solve all the conflicts in a co-ordinated way. If it does not find anything, it decomposes the conflict and the sub-conflicts are solved (Shively, 1984). Even if the searching algorithm is faster because only one solution for the first TAC is necessary, this analysis proves too myopic: solving the first conflict in time is not necessarily the best global solution. In the latter option the high-level analysis comes before an effective search because the first conflict must be chosen. A similar structure occurs in the Hierarchical CBR structure where the high-level case-base could be replaced by a set of rules that perform the same analysis.

Having all the solutions to the conflicts available, on the other hand, even if more time consuming, gives a broader view of the conflict and thus the high-level analysis can be more general. In Section 7.6 the structure adopted for the final version of ISAC is described with the corresponding high-level rules.

5.5 Adaptation

The three possible types of adaptation have already been mentioned in Chapter 3. Depending on the case representation adopted, different strategies are possible. As suggested in (Barletta, 1994) and as implemented in most commercial tools, the adaptation influence has been kept to the minimum. Adaptation is considered too expensive relative to

retrieval because it is not general and not easily maintained. Moreover, the types of adaptation that have been found to work in the real world are the simplest and that, in fact, is what has been done for ISAC.

For these reasons, no adaptation is used for the “TwoInOne” case representation and the solution of the retrieved case closest to the target is directly applied. Substitution Adaptation is used for the “OneInOne” case representation because this representation implies that each conflict is represented with two or more cases and a solution must be retrieved for each one. This structure requires a policy for the extraction of the final solution from the two sets of matching cases, because the two solutions could lead to an incongruous situation.

In fact, this is a very delicate issue. Let us suppose that in the case-base there are two conflicts A-B and C-D which are represented with four cases A, B, C, D. If X-Y is a new conflict very similar to the conflict A-B, ISAC will build two cases X and Y and will start the retrieval process. The retrieved cases will not necessarily be A and B, because the retrieval results depend on the individual aircraft matching. It could happen that X on its own is more similar to D and the retrieved conflict will be D-B instead of A-B. This is one of the reasons why the case-base with the “OneInOne” case description performs less well than the case-base with the “TwoInOne” case description. Our current policy is to select the highest scoring case but more experimentation is required to clarify this issue.

5.6 Summary

In this chapter all the choices inherent to the CBR aspect of ISAC have been analysed and justified. It is explained why three different case representation have been chosen for evaluation and why the possible solutions for a case have increased from 9 to 12. The use of the information carried by the parameters, shown by decision tree or the discriminatory power, proved to be useful for the refinement of the parameters necessary in the case description.

While adaptation issues have not been deeply treated because the adaptation process is almost absent in ISAC, issues concerning a hierarchical structure that could deal with multiple aircraft conflicts have been analysed in detail after having defined the problem with the classification of multiple aircraft conflicts into two categories: simple and complex.

Chapter 6

The Knowledge Engineering Problem

Having access to relevant case history in problem solving reduces the need for problem analysis because solution chunks from old problems can be reused and less in-depth analysis of the new problem is required. This suggests that developing CBR systems may require less knowledge engineering than, say, rule-based or model-based approaches. It is generally accepted among CBR researchers that this is only true to a limited extent. A CBR system that is not built on the type of domain analysis that knowledge engineering involves will probably not work very effectively (Cunningham, 1998).

The development of a knowledge-based system (KBS) involves: identifying a real world problem solving task that is to be tackled, representing the key components of this task in the KBS, and implementing the inference process that produces solutions. Thus there are two key components involved in the knowledge engineering process. There is the task of producing a representation of the problem that captures the key parameters and the task of developing an inference mechanism that describes the causal interactions involved in deriving solutions, as shown in Figure 6.1.

The inference mechanism is implemented using a case-base of solved problems and a mechanism for retrieving and adapting these cases. Many implemented CBR systems involve little or no adaptation and the reasoning mechanism is simply a retrieval system with solutions being used intact or with adaptation performed by the user.

The knowledge is encoded in the system in:

- the knowledge representation used,
- the similarity metric utilised in identifying cases to be reused,
- the mechanism for adapting solutions, if any.

This agrees with the knowledge containers model presented in (Richter, 1995). The development of the similarity metric and the adaptation mechanism is probably simpler than alternative techniques provided the adaptation mechanism does not prove too complicated (Cunningham, Finn and Slattery, 1994).

If retrieval and adaptation mechanisms are easy to implement then CBR has clear knowledge engineering advantages over "from first principles" techniques. However, this analysis will be less important if the problem analysis task that produces the problem representation should dominate in the knowledge engineering effort.

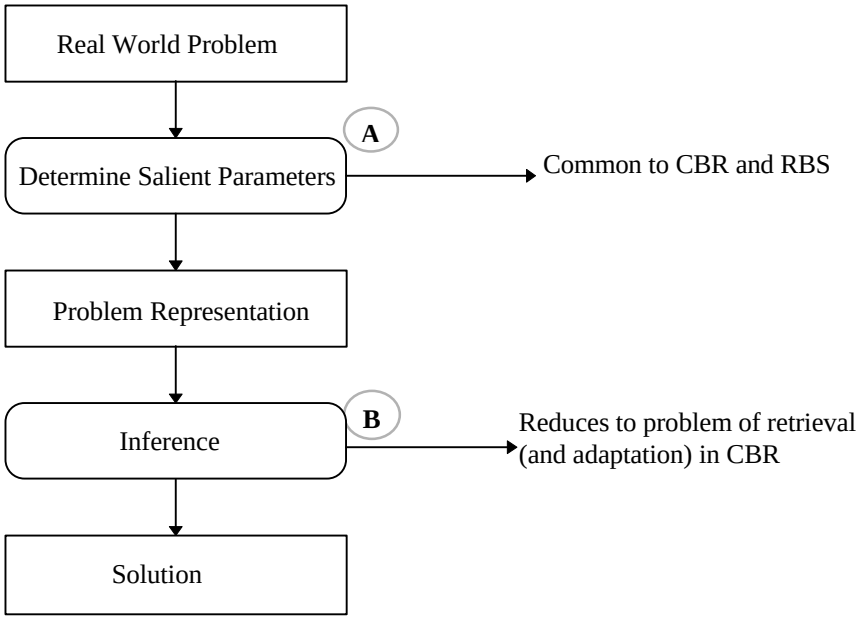

Figure 6.1: *Development of a KBS.*

The main issue remains the knowledge representation. In the next sections an iterative process of improving the representation driven by an analysis of the faulty solutions produced by ISAC is analysed.

6.1 Getting Started (April 1995)

To start understanding the ATC domain, the available literature on ATC and on expert systems has been investigated, as reported in Chapter 2. Talking to controllers and taking part in real time simulations was another important step for the understanding of the domain.

The Reduced Vertical Separation Mode (RVSM) simulation took place in the Eurocontrol Experimental Centre, Paris in May 1995. The simulated sector was in Switzerland above the Zurich airport, with Italy, France and Germany as bordering sectors. The aim of the simulation was to measure the controllers workload at that time and in the year 2000, when the traffic will be heavier, and to check if the controllers workload could be reduced with the introduction of more flight levels. The two simulated scenarios were:

- the conventional scenario, with a flight level every 1000 ft below 29,000 ft and a flight level every 2000 ft above 29,000 ft because the altitude instruments become less precise at high altitude.

- The RVSM scenario, which simulates the situation with an altitude separation of a 1000 ft everywhere because it is supposed that in the future the instruments on aircraft will be precise even above 29,000 ft.

The participation in this simulation and the discussions with the controllers suggested some initial ideas on what the parameters for the description of the conflict could be, the problems that could be encountered and the assumptions to be made. Some of these assumptions are still valid now, like the decision made to consider the zones where the weather conditions are severe (Significant Meteorological Situations: SigMetS) as no-go zones. As a consequence of this decision, there are no parameters in the conflict description mentioning the weather and it is up to the program that detects conflicts to build the no-go zones representing the SigMetS.

Another important initial decision was to assume that all the aircraft are Instruments Flight Ruled and not Visual Flight Ruled. This assumption is acceptable if it is considered that intelligent assistants will begin to help controllers in the future when aircraft will be better equipped and will be able to fly guided by instruments in any phase of flight.

6.2 Initial System Description (from May 1995 to March 1996)

The first environment tool in which HIPS was embedded was a very simple visualisation tool called “Pepsi3” representing the radar screen and the flight plan strips used by the controllers. In “Pepsi3” a lot of significant parameters were missing. The first case-base was built by showing the controllers some conflicts coming from very simple traffic samples and by generalising from what was understood from the literature. This first case description is reported in Table 6.1 with the name of the parameters and their possible values.

This first description was heavily influenced by two systems previously developed in Eurocontrol Experimental Centre, Paris: ARC2000 (Nicolaon, 1992) and PAT Problem Solver (Meckiff, 1994). Those two systems are treated in Chapter 2. The Phase of Flight’s value “pre-descent” comes from ARC2000 but it was not used in the following steps of ISAC’s development because it was considered too specific for the granularity of the description needed.

In the case-base, each case has a name, is described using the above parameters and has a solution which consists of the aircraft that has to be manoeuvred and the type of manoeuvre. Two different case representations were adopted and tested: a representation with the description of both the aircraft in the same case, referred to as “TwoInOne” and a representation referred as “OneInOne” with the description of only one aircraft in each case.

The advantages and disadvantages of each alternative have already been treated in Section 5.3.

Table 6.1: *Initial case description.*

	Name of parameter	**Possible values**
General parameters	horizontal-conflict-configuration	similar track, opposing track, crossing track
	medical-emergency	yes, no
For each aircraft involved in the conflict	size of the aircraft	light, small, medium, heavy
	absolute-speed	numeric value
	phase-of-flight-before conflict	climb, cruise, pre-descent, descent
	phase-of-flight-during conflict	climb, cruise, pre-descent, descent
	phase-of-flight-after conflict	climb, cruise, pre-descent, descent
	altitudes before, during and after the conflict	numeric value
	is the aircraft close to the sector boundaries?	yes, no
Aircraft capabilities	can the aircraft go faster?	yes, no
	can the aircraft go slower?	yes, no
	can the aircraft climb?	yes, no
	can the aircraft descend?	yes, no
	can the aircraft turn left?	yes, no
	can the aircraft turn right?	yes, no
Environment parameters	is an altitude manoeuvre possible?	above, below
	is an horizontal manoeuvre possible?	left, right
	is a speed manoeuvre possible?	acceleration, deceleration

Problems

The main problem encountered while preparing this first conflict representation was the excessive simplicity, and sometimes superficiality, of many of the components: the conflicts were badly specified, the environment was too unrealistic and the acquisition of the parameters was not accurate.

The conflicts were badly specified because too much information was missing, like the departure and arrival airports, the type of aircraft etc. The visualisation tool, “Pepsi3”, had

been written only to show the ideas behind HIPS. Being a prototype, it was very simple and, because it emphasised the geometrical aspect of the no-go zones, the initial case description was mainly geometrical, with no reference to any aircraft performance. In the subsequent steps of the knowledge acquisition process, more than one parameter that deals with aircraft performance will appear. Talking with controllers and using more accurate simulation tools showed that the performance parameters were necessary.

The acquisition of the parameters was a problem independent of the previous two. Two conditions had to be satisfied: the possibility of extracting from the available data what the controller could easily see on his radar screen and being sure that the extracted information represents what the controller is actually seeing on the radar screen.

The first issue implied the manipulation of a lot of data and the use of a lot of geometry, e.g. to find the angle to exit from a no-go zone. The second issue implied asking the controllers a lot of questions to see if the acquired parameters were expressing exactly what he/she intended. A list of all the algorithms used to acquire the parameters is in Chapter 4 with all the changes made from the first draft until the final version listed along with some alternatives.

The algorithms used in “Pepsi3” and its subsequent versions for the calculation of the no-go zones, trajectories and aircraft performances have changed during the lifetime of the project but it has always been assumed that these changes would be hidden from ISAC and its behaviour would not be affected.

For the evaluation of this version of the retrieval mechanism and of the current case representation, a case-base composed of gold-standard cases was built. Because the parameters used were so simple, it seemed possible to cover almost all the case space with gold standard cases. The “LeaveOneOUT” test on a case-base of 50 cases was adopted. The results were quite good: the system found the same solution as the one given by the controller in 95% of the conflicts. The issue of using either gold standard cases or cases specific to a certain sector has already been discussed in Section 5.1.2.

These results were not reliable for two reasons: one, already mentioned, is that the domain was too simplistic, but the main reason is that the solutions to the conflicts in the case-base had been given by a non-expert, whose knowledge in the domain was limited to a small set of empirical rules learned watching the controller solving the conflicts.

A CBR system, to work well, needs coherency in the solutions. When it seemed that the controller was not coherent, a set of rules generating the solution was used to have an “artificial” coherency. In reality, the “incoherent” solutions given by the controller were due

to some parameters that had not been considered in the case description but that the controller was automatically assuming by default depending on the conflict and on the sector.

Summarising, the first step in the construction of the case-base was necessary to set all the structures up and to prepare the retrieval engine, but the obtained system was a toy system that did not have the robustness needed to work in the real world.

6.3 Interim Refinements Description (from April 1996 to June 1996)

The second step towards reaching a consistent case description was to gradually eliminate the unrealistic factors. It was first decided to build a completely new case-base by using traffic samples coming from real time simulations. Secondly, a new environment visualisation tool called GHMI was introduced. This tool does not change the way of displaying the three HIPS windows but simulates an actual radar screen for the visualisation of the sector.

With this new environment the controllers felt more comfortable and the solutions given became more precise. More complex parameters were used and the number of possible values for each parameter augmented. The possibility of having more than one acceptable solution was introduced for two reasons: either some parameters, whose variation could lead to a different solution, were not considered or, more simply, the conflict could be solved in more than one way.

The case structure and the case-base had to be completely rewritten. The traffic samples available did not contain conflicts, and the creation of a conflict by slightly modifying the flight plan was difficult: some of these slight changes seemed illogical to the controller, even if, for a non-expert eye, there was nothing wrong (e.g., an aircraft far from destination which is descended one level with the purpose of creating a conflict seems a plausible manoeuvre to a non-expert eye. The same manoeuvre was illogical for the controller because, to save fuel, an aircraft far from its destination should not descend).

A controller examined the entire set of new and more realistic conflicts and gave some “non-artificial” solutions. It seemed that two problems had been solved: the conflict description was more realistic compared to the previous tool “Pepsi3” and the solution had been given by an expert. The new case structure with the parameters and values is in Table 6.2. Most of the parameters imply the existence of a “first” and a “second” aircraft. The set of rules described in Section 4.3 decides which is the first aircraft and the “canonical” case description, as discussed in Section 5.3.1, depends on those rules.

Table 6.2: *Interim case description.*

Name of parameter	Possible values
horizontal-conflict-configuration	crossing, catching, joining, facing
altitude-intention (profile of the two aircraft)	StableStable, StableDesc, StableClimb, DescStable, DescDesc, DescClimb, ClimbStable, ClimbDesc, ClimbClimb
altitude-now (altitude of the first aircraft compared to the second)	higher, lower, same
speed (speed of the first aircraft compared to the second)	faster, slower, same
horizontal-intention (where are the aircraft turning)	LeftLeft, LeftStr, LeftRight, StrLeft, StrStr, StrRight, RightLeft, RightStr, RightRight
performance (performance of this aircraft compared to the other)	better, same, worse
miles-done (how many miles has the aircraft done)	numeric value
miles-to-do (how many miles has the aircraft to do)	numeric value
close-to-boundaries (is the aircraft far from the sector boundaries?)	yes, no
same-destination (are the two aircraft going to the same destination?)	yes, no
left-exit-no-go (angle to exit the no-go zone by turning left, in degrees)	numeric value
right-exit-no-go (angle to exit the no-go zone by turning right, in degrees)	numeric value
right-available (space available on the right, in degrees)	numeric value
left-available (space available on the left, in degrees)	numeric value
in-front-direct (does the aircraft pass in front of the other one if it goes directly?)	yes, no
in-front-more-space (does the aircraft pass in front of the other one if it goes where there is more space?)	yes, no
requested-level-free (which levels are available)	None, YesRequested, YesOverInit, YesBelowInit, Above&Below
faster (percentage of speed increase to exit the no-go zone)	numeric value
slower (percentage of speed decrease to exit the no-go zone)	numeric value
dimension-altitude-zone (dimension of the no-go zone in the altitude display)	big, small

The most evident change to the previous description is that there are less parameters dealing with the geometric description of the conflict and more parameters dealing with the performance of the aircraft. The parameters implying a direct route ("InFrontDirect" and "InFrontMoreSpace") will not be used in the next steps because horizontal manoeuvres are not very common. Parameters like "Horizontal Intention" that might seem essential to a non-expert for understanding the geometry of the conflict were present in the initial description but were discarded afterwards: the controller is not interested in knowing whether the aircraft is turning right or left, but is only interested in knowing whether the aircraft is turning or not.

From a practical point of view, the presence of the sector boundaries caused a lot of problems in the automatic acquisition of the parameters.

The overall complexity affected the performance of the system. The case-base, constituting of 60 "realistic" conflicts extracted from the available traffic samples, was tested with the "LeaveOneOUT" method. The results, presented to the 1996 European Workshop on Case-Based Reasoning (Bonzano, Cunningham and Meckiff, 1996), were worse than the results of the toy system: only 70% of the solutions suggested by ISAC matched the solutions given by the controller. The main reason for this was the lack of coverage of the space of all the possible cases. A case-base of 60 cases was too small to cover the huge case space of more than 4 million possible cases; this value is obtained by multiplying all the possible symbolic values of all the parameters. Even if a large part of these cases would never appear in the real world, 60 cases were not enough. Moreover, the introduction of the constraints reduced the performance (Bonzano, Cunningham and Meckiff, 1996).

All the tests were performed using "HorConflConf" as the only constraint. This caused some problems when particular geometries of conflict were encountered: for example, in the case-base there were not enough "head-on" conflicts, so the solutions of most of the cases where the constraint's value was equal to "head-on" were solved incorrectly.

6.4 Third System Description (from July 1996 to September 1996)

After the toy system and the first attempt to work with a real world system, the need for a bigger case-base was evident. Because of the lack of time, it was not possible to continue acquiring conflicts from the traffic samples to build a bigger case-base. The 60 conflicts coming from the real world simulation that constituted the case-base in the previous step were kept as a test set. A new case-base was built from scratch by giving the controllers the description of a general conflict and asking for its solution, then changing the parameters

one by one and recording how the solution would change accordingly (see Appendix A). This approach implies that a case is now hand written and not generated from a real traffic sample, whereas before the case was automatically written by the environment visualisation tool as soon as a conflict was displayed. In the meantime a new case representation was introduced as reported in Table 6.3.

Table 6.3: *Third case description.*

Name of parameter	**Possible values**
horizontal-conflict-configuration	crossing, converging, head-on, diverging
altitude-intention (altitude profile of the aircraft at the beginning of the conflict)	stable, descending, climbing
easy-to-exit-right (how easy it is to exit the horizontal no-go zone by turning to the right)	veryEasy, easy, possible, difficult
easy-to-exit-left (how easy it is to exit the horizontal no-go zone by turning to the left)	veryEasy, easy, possible, difficult
manoeuvrability (this depends on the aircraft type and on its fuel load)	high, medium, low
close-to-TOD (how many miles there are between the aircraft and the Top Of Descent)	numeric value
close-to-boundaries (how many minutes from the sector boundaries the action point is)	numeric value
levels-available (which levels are available)	none, yes, above, below, withSpaces
faster (is it possible to exit the no-go zone by increasing speed?)	easy, possible, difficult
slower (is it possible to exit the no-go zone by decreasing speed?)	easy, possible, difficult
agreements (agreements with next sector)	sequencing, notSequencing
priority (a commercial aircraft has higher priority than a business or a military aircraft)	higher, same, lower
altitude-now (altitude of the first aircraft compared to the second)	different, same
speed (speed of the first aircraft compared to the second)	faster, slower, same
similar (are the parameters AltitudeIntention, CloseToTOD, Performance and Priority equal?)	yes, no

A drastic reduction of the parameters represented with numerical values is evident. It is easier to store the controller's knowledge with symbolic values because the controller usually has a quick overview of a conflict and can give a qualitative description of it which is better described with symbolic values. The acquisition interface uses all the data supplied

by HIPS to create some symbolic values that express exactly what the controller thinks. The performance of the system with this case representation is examined in more detail in Chapter 8.

During this analysis, it was discovered that the controller's workload heavily influences the solution of the conflict even if it is not directly connected to the conflict description. Depending on the workload, a controller could alter his behaviour. If the workload is low the controller has time to choose a complex solution that will be less expensive for the aircraft; on the other hand if the workload is high there is time only for a very simple but sometimes expensive solution that does not need any monitoring. A way of calculating the workload could be to count the number of aircraft that are visible on the radar screen and if more than a certain percentage of the aircraft are climbing or descending than the workload is considered high. This percentage threshold varies depending on the controller and each controller could suggest different ways for the calculation of the workload. A futuristic alternative could be to measure the workload on a biological basis: by measuring the stress of the controllers with electrodes, the workload can be evaluated (Caloo, 1997). The workload, as a parameter, has not yet been used in ISAC.

A new policy for the consideration of the NIL values was tried as explained in Section 5.1.4. With this new policy a lot of the conflicts that were different by non-significant parameters were reduced into only one by assigning to the non-significant parameters a NIL value. For this reason the 150 conflicts stored in the new case-base were representing, in reality, many more conflicts.

The last change introduced during this stage was the evaluation strategy. The "LeaveOneOUT" strategy was substituted by a more realistic test on the case-base using as a test set the 60 conflicts coming from the real traffic samples as mentioned above.

6.5 Fourth System Description (from October 1996 to June 1997)

The case-base was once more judged as not being representative after some tests with controllers. Moreover some parameters had to be changed and usually when a new parameter is added to the case description its value is NIL for all the cases that were already present in the case-base. But in ISAC's particular situation this was not possible because the new parameters were substituting some old ones. A new case-base was built with a lot of *numeric* parameters coming directly from "raw" data with the aim of reducing to a minimum the manipulation of the data. The final and current case description is reported in Table 6.4.

This is the final version for the non hierarchical structure. An example of a case expressed in the "OneInOne" and "TwoInOne" case representation is in Tables 6.5 and 6.6. The parameter "performance" is calculated with the help of the BADA[5] database (Bos, 1997). During the previous steps of ISAC's engineering process, the performance categories had been decided by a controller. This approach often implied that two aircraft belonging to the same category were considered different by the controller but ISAC could not realise it. By using a continuous parameter from BADA any ambiguity is eliminated.

Table 6.4: *Final case description.*

Name of parameter	**Possible values**
horizontal-conflict-configuration	crossing, converging, head-on, diverging
priority	higher, lower, same
altitude-now	different, same
speed	faster, slower, same
altitude-configuration	climbing, descending, stable
close-to-TOD	numeric value
close-to-boundaries	numeric value
manoeuvrability	numeric value
easy-to-exit-horizontally	veryEasy, easy, possible, difficult
levels-available	yes, none, above, below, withSpaces
faster	easy, possible, difficult
slower	easy, possible, difficult

After realising that the most frequently used manoeuvre is the altitude manoeuvre and that the horizontal and speed manoeuvres are not frequently used (see Figure 8.3), there was no longer a need to discriminate between the two "easy-to-exit-right" and "easy-to-exit-left" parameters: the more general "easy-to-exit-horizontally" parameter was introduced. For the same reason, the "altitude" solution given by the system was changed into a more precise "climbing" or "descending" solution (see Section 5.1.3).

The function that calculates the spreading activation was modified to shift the range of the activation from "0 to 1" to "-1 to1". With this new convention a parameter with a NIL value has activation 0 which is intuitively more correct than having activation 0.5. For the

[5] The Base of Aircraft Data (BADA) provides a set of ASCII files containing performance and operating procedure coefficients for 165 different aircraft types. The coefficients include those used to calculate thrust, drag and fuel flow and those used to specify nominal cruise, climb and descent speeds.

symbolic parameters, if the value of the target is the same as the case's value, the activation is +1, otherwise it is -1.

For the numeric parameters, the activation is calculated with this formula:

$$w \cdot \left(2 \left(1 - \frac{|v_t - v_c|}{v_{\max} - v_{\min}} \right) - 1 \right)$$

where $\mathbf{v_c}$ and $\mathbf{v_t}$ are the case and target values and $\mathbf{v_{max}}$ and $\mathbf{v_{min}}$ are the maximum and minimum values for that parameter in the case-base. This gives an activation that can vary from ***-w*** to ***+w*** continuously instead of having a discrete value (+1, +0.75, +0.5 or 0) as it was in the previous development step. Since the activation can be smaller than zero, the system gives a solution only if the highest activation is bigger than zero, otherwise a message saying "Unable to give solution" is prompted. If the highest activation of all the cases in the case-base is smaller than zero it would mean that even the most similar case is too far from the target to have an acceptable solution.

After some discussion with an expert on cognitive psychology experiments, a new policy for the evaluation of the system was introduced and is described in Chapter 8. During this development step, some experiments on introspective learning of local and global weights have been performed and are discussed in Chapter 7.

Table 6.5: *A conflict expressed in the "OneInOne" case representation.*

Casename	Case690(A)	Casename	Case690(B)
HorConflConf	crossing	HorConflConf	crossing
AltitudeNow	same	AltitudeNow	same
AltConfiguration	stable	AltConfiguration	stable
Speed	faster	Speed	slower
CloseToTOD	155	CloseToTOD	352
CloseToBoundaries	4.8	CloseToBoundaries	8.3
Manoeuvrability	.78	Manoeuvrability	.78
Priority	same	Priority	same
EasyToExitHorizontally	easy	EasyToExitHorizontally	possible
LevelsAvailable	yes	LevelsAvailable	yes
Faster	difficult	Faster	difficult
Slower	difficult	Slower	difficult
Solution	dow1	Solution	dow2

Table 6.6: *A conflict expressed in the "TwoInOne" case representation.*

Casename	Case690
HorConflConf	crossing
Priority	same
AltitudeNow	same
Speed	faster
AltConfiguration(A)	stable
CloseToTOD(A)	155
CloseToBoundaries(A)	4.8
Manoeuvrability(A)	.78
EasyToExitHorizontally(A)	easy
LevelsAvailable(A)	yes
Faster(A)	difficult
Slower(A)	difficult
AltConfiguration(B)	stable
CloseToTOD(B)	352
CloseToBoundaries(B)	8.3

```
Manoeuvrability(B)        .78
EasyToExitHorizontally(B) possible
LevelsAvailable(B)        yes
Faster(B)                 difficult
Slower(B)                 difficult
Solution                  dow1
```

6.6 Hierarchical System (from June 1997 to October 1997)

The knowledge engineering steps seen until now focused on the resolution of two aircraft conflicts (TACs). The last step to complete the system was the implementation of a structure for multiple aircraft conflicts (MACs). The interface between GHMI and ISAC has been changed to make it possible to acquire the description of more than two aircraft. ISAC's code, too, had to be changed.

As already said in the Case Structure section (5.3), a lot of choices made for the system when solving TACs have been influenced by the fact that it was known that ISAC would have had to solve MAC. The main issue was to reuse the case-base of TACs without having to create from scratch a case-base of 3 aircraft conflicts, then 4 aircraft conflicts etc.

Of the three options for the resolution of a multiple aircraft conflict that have been introduced in Section 5.4, the only one that has been implemented so far is a simplified version of the "Look Ahead CBR". The "Independent CBR" option has been discarded because the performance of the system when using the "OneInOne" case representation was not as good as the performance with the "TwoInOne" case representation. The "Hierarchical CBR" option has not been implemented for reasons of time: the construction of a new case-base implies finding from scratch new parameters and new cases to fill the new case-base.

The heuristic rules used for the high level analysis have been suggested by controllers and can easily be changed depending on the controller's preferences. The rules should change depending on the hierarchical structure used. For the "Look Ahead CBR", they are:

- check if, among the solutions to the TACs, there is a solution common to all the TACs. If yes, this common solution becomes the solution to the MAC.
- If no common solution is found, an aircraft manoeuvred in all the TACs is searched for. If found, the solution valid for that aircraft is given as solution to the MAC.
- If no common aircraft is found, the TAC closest in time is solved and that solution becomes the solution of the MAC.

For lack of time it has not been possible to implement all of these rules and in the evaluation of the hierarchical structure presented in Chapter 8 the high level analysis consists only of the first rule.

6.7 Conclusions

In the previous sections the knowledge engineering process for the construction of ISAC has been shown. The first stage involved an analysis of the problem that produced a representation that can be manipulated by the reasoning system. The second stage involved developing the reasoning mechanism that manipulates the problem representation to produce a solution.

The second step was the easiest to accomplish. The coding of the retrieval algorithm and adaptation, when present, was done without any major problems and, apart from difficulties with the portability of some libraries (e.g. Motif), the system is able to work with any case-base containing both numeric and symbolic parameters and no speed problems have been encountered.

The first step was the most problematic. As described in (Bayles et al., 1993), a lot of hours have been spent interviewing specialists and reading literature. As a starting point, an available system was taken (Meckiff, 1994) and from that system the lengthy job of acquiring the case-base began. Even if a lot of conflicts were available, their solutions were not, making it impossible to build a case-base from the existing data. Different options have been tried and the problem of having a representative case-base is not yet completely solved.

When the number of cases and the methodology for acquiring them were first discussed, it seemed that a case-base of 30-50 conflicts would have been big enough to start the tests and that these conflicts could be hand crafted. As already explained, both of these assumptions were wrong due to the complexity of the domain.

The absence of an adaptation mechanism made it necessary to have a case-base with good coverage. Second, the complexity of the domain implied that the case-base contained lots of cases. Finally, having a lot of conflicts in a case-base is not enough: each conflict needs a solution, too. Moreover, the solutions must be coherent and must satisfy the controller. Two conditions have to be respected in order to have an effective CBR system:

1. There must be enough cases drawn *from the same sector*. If cases are not from the same sector and the case-base is used to solve conflicts on the same sector, the chances that a similar conflict is already in the case-base is higher. Having cases belonging to the same sector will reduce the complexity of the domain and the size of the case-base.

2. The solutions to the conflicts that are stored in the case-base must be given by the controllers that usually work on that sector. This will avoid the situation where controllers give different solutions to the same conflict either because they have different background or because they use the tools in a different way. Practices in use in individual sectors will ensure that controllers working on the same sector will give coherent solutions.

The tool used for displaying the conflicts influenced heavily the choice of the parameters and the solutions of the conflicts. The more realistic the tool, the more reliable the solutions given by the controller. The decision whether to use gold standard cases or noisy cases depends on the way the case-base is acquired: gold standard cases will be used if the case-base is built by hand but, on the other hand, the case-base will contain more noisy data if the case-base is directly acquired from the sector.

Some data had to be entered by hand but in an operational system all the data should be acquired electronically because the controllers will have neither time, nor inclination, to enter all the data by hand.

It was anticipated that ISAC would not have had to deal with incomplete data in the traffic samples used, but this was not true: the acquisition of some data was quite difficult and, often, the data that the controller was acquiring very easily could not be translated so easily into parameters for ISAC. Introspective Learning techniques could help in reducing the negative effect of the lack of cases.

It can be said that it is true that CBR does not eliminate the knowledge engineering problem but it does reduce it. With CBR, the parameters that describe a problem have to be found, but how these parameters influence the decisions does not have to be discovered (Thompson, 1997).

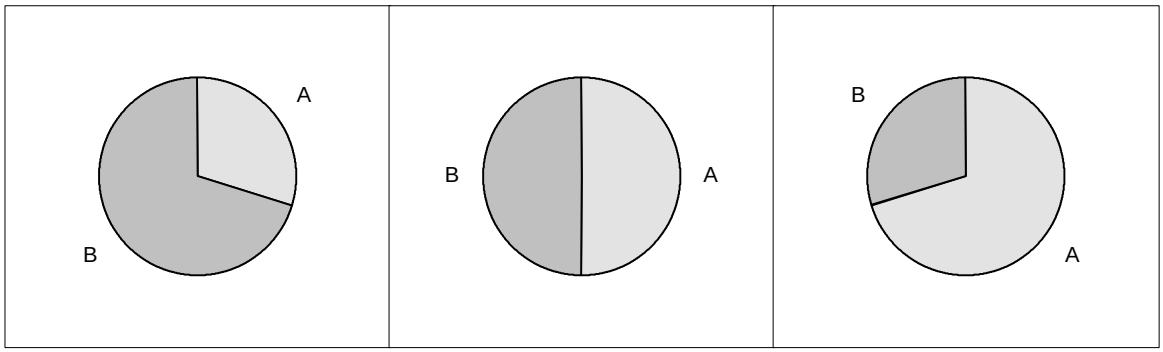

Figure 6.2 (a,b,c): *Different structures for the knowledge engineeing process.*

The model of KE requirements described in Figure 6.1 can result in various specific scenarios. The characteristic of specific applications will dictate the balance of effort

between tasks A and B in Figure 6.2 where “A” represents the determination of salient parameters, whereas “B” represents the inference mechanism. CBR is very effective in a situation like in Figure 6.2(a), where the acquisition of the case-base and the decision of the parameters is not as relevant as the retrieval component of the system. When the acquisition of the case-base and the decision of the parameters becomes more dominant, like in Figures 6.2 (b) and (c), the advantages of CBR over RBS are less evident.

The main conclusion from those considerations, which will be more deeply analysed in the last chapters, is this:

CBR can be used in the ATC domain ***iff*** *an adequate case-base is available and if all the cases come from the same sector with solutions given by controllers trained on that sector.*

Chapter 7

Introspective Learning of Parameter Weights

7.1 Introduction

As seen in the previous chapters, the descriptive parameters usually have different discriminatory power. In this chapter the weight issues relating to the choice of weights are analysed in more detail. When a *k*-Nearest Neighbour (*k*-NN) technique is used for case retrieval, the accuracy depends on the weights assigned to the parameters. Recent research in Machine Learning and Case-Based Reasoning has shown that Introspective Learning (IL) of parameter weights can improve accuracy (Saltzburg, 1991; Fox and Leake, 1995; Wetterschereck and Aha, 1995; Muñoz-Avila and Hüllen, 1996).

Developing the *k*-NN retrieval system used in ISAC has been problematic because not only have the relevant parameters been difficult to determine but because the relative importance of parameters has been difficult to gauge. Moreover some parameters were highly context sensitive: i.e. parameters that were very predictive in some conflicts were not relevant in others.

Two types of weights were analysed: local and global weights. If a parameter has a global weight, its weight will be the same for all the cases in the case-base, i.e. its importance is the same in all the cases. On the other hand, if a weight is local, its value could change depending on the case under examination and on the values assumed by other parameters. This option is more flexible if a parameter is context sensitive.

In this chapter we will present our experiences with introspective learning and describe the lessons learned. We present four central findings:

- How weights should be adjusted.
- What cues should drive learning.
- When to use local and global weights.
- Introspective learning does not work well with pivotal cases.

We begin with a general review of introspective learning in the next section, then we present the learning policies in Section 7.3. The updating policies are in Section 7.4 for local

weights and in Section 7.5 for global weights. Section 7.6 is the evaluation section with the performance comparison between global and local weights.

7.2 Background

Introspective learning refers to an approach to learning problem solving knowledge by monitoring the run-time progress of a particular problem solver (Fox and Leake, 1995; Leake, Kinley and Wilson, 1995; Oehlman, Edwards and Sleeman, 1995). In particular, we have investigated the problem of learning parameter weights by monitoring the retrieval performance of ISAC, work that is related to similar research in the machine learning community (Saltzburg, 1991; Wettschereck and Aha, 1995; Wettschereck, Aha and Mohri, 1997).

Traditionally, Artificial Intelligence research has focused on the acquisition of domain knowledge in order to provide basic problem solving competence and performance. However, even when a reasoner has a correct set of knowledge it may still experience reasoning failures. This can be explained as an inability of the reasoner to properly access and apply its knowledge. For this reason researchers have looked at how monitoring problem solving performance might lead to new learning opportunities that can improve the way in which available knowledge is used. This form of introspective reasoning and learning has become more and more important in recent years as AI systems have begun to address real world problem domains, characterised by a high degree of complexity and uncertainty. In such domains, where determining the necessary world knowledge is difficult, it is also difficult to determine the correct reasoning approach to manipulate this knowledge effectively. Hence the need for introspective learning, and its increasing popularity across a range of AI problem solving paradigms, from planning to case-based reasoning.

Meta-planning was an early model of introspective reasoning found in the MOLGEN planning system (Stefik, 1981). MOLGEN could, to some extent, reason about its own reasoning processes. Meta-planning provided a framework for partitioning knowledge into layers, separating planning knowledge (domain knowledge and planning operators) from meta-knowledge (planning strategies). Introspective reasoning is implemented as planning within the meta-knowledge layer.

SOAR (Laird, Rosenbloom and Newell, 1986; Laird, Newell and Rosenbloom, 1987) also employs a form of introspective reasoning. It learns “meta-rules” which describe how to apply rules about domain tasks and acquire knowledge. SOAR’s meta-rules are created by

chunking together existing rules and learning is triggered by sub-optimal problem solving results rather than failures.

Case-based reasoning researchers have also begun to understand the importance of introspective reasoning. Fox and Leake (1995) describe a case-based system called ROBBIE which uses introspective reasoning to model, explain, and recover from reasoning failures. Building on ideas first put forward by Birnbaum et al. (1990), Fox and Leake take a model-based approach to recognising and repairing reasoning failures. Their particular form of introspective reasoning focuses on retrieval failures and case index refinement. Work by Oehlmann, Edwards and Sleeman (1995) addresses the related topic of re-indexing cases, through introspective questioning, to facilitate multiple viewpoints during reasoning. Leake, Kinley, and Wilson (1995) describe how introspective reasoning can also be used to learn adaptation knowledge in the form of adaptation cases.

Many case-based reasoning systems use the *k*-nearest neighbour (*k*-NN) classifier (or a derivative) to retrieve cases. One of the problems with this approach is that the standard *k*-NN similarity function is extremely sensitive to irrelevant, interacting, or noisy parameters. The typical solution has been to parameterise the similarity function with parameter weights so that, for example, the influence of irrelevant parameters can be de-emphasised through the assignment of a low weight. However, suitable weight vectors are not always readily available. This has lead to a number of parameter-weight learning algorithms which attempt to introspectively refine parameter weights on the basis of problem solving successes or failures.

7.3 Learning Policies

The basic idea behind the introspective learning of parameter weights is to increase or decrease the weights of selected case parameters on the basis of problem solving performance. Parameter weighting methods differ in terms of their learning criteria as well as in terms of their update models.

There are four distinct policies that can drive learning (i.e. that trigger the change of a parameter weight). Two basic learning criteria are used, failure-driven and success-driven. Failure-driven methods only update parameter weights as a result of a retrieval failure, and conform to the “if it’s not broken do not fix it” school of thought. Success-driven approaches seek to update parameter weights as a result of a retrieval success. For each approach the weights of matching and unmatching parameters are increased or decreased accordingly.

By changing the weights, we move the cases in the case space. We want the cases that led to a correct solution to be "pulled" closer to the target and the cases that were retrieved incorrectly to be "pushed" away from the target as can be seen in Figure 7.1.

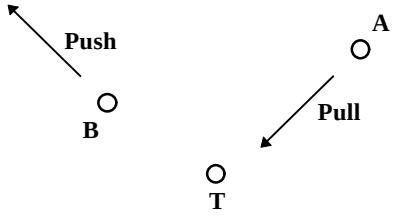

Figure 7.1: *Pushing and pulling a case.*

There are four possible learning policies; two cause a "push" and two cause a "pull":

- GUM, Good Up Matching: the case retrieved from the case-base has the same solution as the target in the training set (Good retrieval). We increase (Up) the weights of the parameters that have the same value as the target (Matching values). By doing this we increase even more the case's activation, i.e. we "pull" the case towards the target.
- GDU, Good Down Unmatching: the case retrieved from the case-base has the same solution as the target in the training set (Good retrieval). We decrease (Down) the weights of the parameter that have a different value from the target (Unmatching values). The non-matching parameters decrease the case activation even if we want this case to be retrieved, so by decreasing their weights we again "pull" the case towards the target.
- BUU, Bad Up Unmatching: the case retrieved from the case-base has a different solution from the target (Bad retrieval) and the weights of the parameters that have a different value from the target (Unmatching values) are increased (Up). By doing this we "push" the case away from the target because by subtracting an increased weight we reduce even more the activation of the case.
- BDM, Bad Down Matching: the case retrieved from the case-base has a different solution from the target (Bad retrieval) and the weights of the parameters that have the same value as the target (Matching values) are decreased (Down) because these weights contribute too much to the activation that we want to be low. So we are again "pushing" the case away from the target.

Different parameter learning algorithms employ different combinations of these techniques. By far the most common strategy is to use all four update policies (e.g., Salzberg, 1991, Wettschereck and Aha, 1995). However, more focused strategies have also been adopted.

For example, Muñoz-Avila and Hullen (1996) use the BUU and GDU policies to increase or decrease the weights of unmatched parameters after a retrieval failure or success respectively.

The way in which a parameter's weight value is changed during learning, the update policy, is also critical. One of the simplest approaches is to increase or decrease parameter weights by a fixed amount. For example, this method is used in EACH (Salzberg, 1991) where all four of the above learning policies are used to increase or decrease parameter weights by some fixed amount Δf. Salzberg reported that the benefits associated with the weight learning depended on the value of Δf, and that different values of Δf worked better on different data-sets. Muñoz-Avila and Hullen (1996) use a decaying update policy so that the magnitude of weight changes decreases over time.

In general, the relationship between the learning policy, the update policy, and the application domain is not at all clear and requires further work (this point is emphasised in Wettschereck, Aha and Mohri, 1997). In particular, different policies have been reported to give very different performance results. Moreover, the sensitivity of the learning algorithm to noise and parameter interactions needs to be further studied.

7.4 Update Policies for Local Weights

Assigning global weights by hand to the parameters requires a deep domain knowledge but it is still possible. On the other hand it is impossible to assign a local weight to all the parameters of all the cases in a case-base. The alternative is to start with all the weights at the same initial value and to use an introspective learning algorithm to update them.

When a parameter is symbolic, the activation of the case is increased by the weight ***w*** if the values are matching, decreased by ***w*** if the values are non-matching and left as it is if one of the two values is unknown. The activation increase for continuous parameters is proportionate to the proximity of the parameter values: very different values get a negative activation while similar values get a positive activation. The actual activation increase is calculated as follows:

$$w \cdot \left(2 \left(1 - \frac{|v_t - v_c|}{v_{\max} - v_{\min}} \right) - 1 \right)$$

where $\mathbf{v_c}$ and $\mathbf{v_t}$ are the case and target values and $\mathbf{v_{max}}$ and $\mathbf{v_{min}}$ are the maximum and minimum values for that parameter in the case-base. This gives an activation that can vary

from -***w*** to +***w*** as before. The objective for introspective learning is to determine local values for these weights for each parameter in each case in the case-base.

The weight can be updated by an update policy which modifies the existing weight by either adding or multiplying by a constant. This weight change itself can be constant or it can decay as the learning proceeds (Muñoz-Avila and Hüllen, 1996). We use a decay policy. The formulæ for the increase and decrease adding option are as follows:

$$w_i(t+1) = w_i(t) + \Delta i \frac{F_c}{K_c}$$

$$w_i(t+1) = w_i(t) - \Delta i \frac{F_c}{K_c}$$

where K_c indicates the number of times that a case has been correctly retrieved and F_c reports the number of times that a case has been incorrectly retrieved. The ratio F_c/K_c reduces the influence of the weight update as the number of successful retrievals increases and is called the decay function.

The formulæ for the increase and decrease multiplying option are as follows:

$$w_i(t+1) = w_i(t) \cdot \left(1 + \Delta i \frac{F_c}{K_c}\right)$$

$$w_i(t+1) = \frac{w_i(t)}{1 + \Delta i \frac{F_c}{K_c}}$$

We evaluated both the alternatives of adding and multiplying and found little difference between them - adding proved slightly better. The value Δi determines the initial weight change. We tested values of Δi between 0.1 and 2 and settled on Δi=1.0. There was little to choose between values from 0.5 to 2 because the weight change decreases anyway.

When all the weights in a case have been updated they are normalised so that the maximum activation remains the same for all cases in the case-base. This is done as follows:

$$w_{ik} = w_{ik} \frac{Number\ of\ Features}{\sum w_{ik}}$$

The normalisation is important to prevent popular cases becoming dominant attractors in the case-base.

7.5 Update Policies for Global Weights

The global weight updating policies are derived from the local weight ones. Four global update policies have been tested:

S1. Each global weight is updated by adding/subtracting the constant quantity 0.1.

If the weight is to be increased:

$$w_i(t+1) = w_i(t) + 0.1$$

If the weight is to be decreased:

$$w_i(t+1) = w_i(t) - 0.1$$

This policy does not use a decay function, so it is necessary to keep the increment small, otherwise a case that is retrieved too often will have big weights.

S2. Each global weight is updated by adding/subtracting the quantity $\frac{F_c}{K_c}$.

If the weight is to be increased:

$$w_i(t+1) = w_i(t) + \frac{F_c}{K_c}$$

If the weight is to be decreased:

$$w_i(t+1) = w_i(t) - \frac{F_c}{K_c}$$

Note that K_c and F_c belong to the individual case and not to the parameter. This formula is the same as for the local weights with the parameter $\Delta i=1.0$.

S3. Each global weight is the average of all the corresponding local weights after they have been trained with the policy shown in section 7.4. All the weights are considered to calculate the average, even the weights that have not been updated (i.e. all the weights that remain initialised at 1).

S4. Each global weight is the average of all the corresponding local weights, as in strategy S3, but the average is calculated without considering the weights that have not been changed during learning.

The strategies S3 and S4 are time consuming because a previous training of the local weights would be necessary, but are useful to test if the global weights can carry as much information as the local ones.

7.6 Evaluation

For all the experiments we used a case-base of 126 cases coming from the ATC domain, a training set of 40 cases and a test set of 27 cases. Each case has 23 parameters, of which 19

were symbolic and 4 numeric. The system iterated 20 times on the training set to extract the best weights. The points in Figures 7.3 and 7.4 are the average of 50 experiments with different combinations of test set and training set.

All the experiments have been repeated for both local and global weights and for the eleven different combinations of learning policies:

- A*llFour* (GUM + GDU + BUU + BDM) where learning is driven by all the four policies;
- *onlyBad* (BUU + BDM) where learning is driven only by the badly retrieved cases (failure driven);
- *onlyGood* (GUM + GUU) where learning is driven only by the correctly retrieved cases;
- *onlyGUM* and *onlyGDU* where learning is driven only by the cases that are correctly retrieved;
- *withoutGUM* (GDU + BUU + BDM) and *withoutGDU* (GUM + BUU + BDM) where the learning is driven by all the policies except from respectively GUM and GDU;
- *onlyBUU* and *onlyBDM* where the learning is driven only by the cases that are badly retrieved;
- *withoutBUU* (GUM + GDU + BDM) and *withoutBDM* (GUM + GDU + BUU) where the learning is driven by all the policies except from respectively BUU and BDM. (Bonzano, Cunningham and Smyth, 1997,b).

7.6.1 Training the Case-Base

For the evaluation purposes we use three sets of cases: a **case-base** where the cases will have their local weights adjusted during the introspective learning, a **training set** for training the weights in the case-base and a **test set** for testing the error of the case-base. The steps to train and verify the effectiveness of introspective learning are as follows (see Figure 7.2):

- We calculate the initial error on the test set and on the training set when all the weights in the case-base are still set to 1: we call these error figures E_{ts} and E_{tr}.

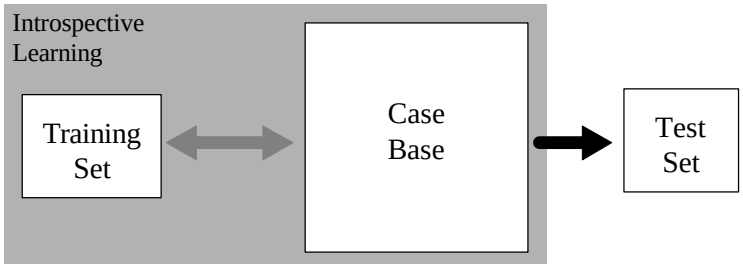

Figure 7.2: *The components in the introspective learning process.*

- We train the case-base by retrieving the *k*-Nearest Neighbours for each case in the training set. The weights of the *k* cases are adjusted based on the various learning and update policies. The values for K_c and F_c are also updated for these cases.
- This training step is repeated several times. E_{tr} and E_{tr} are calculated after each step.

7.6.2 Overfitting

In Figure 7.3 it can be seen that after each iteration E_{tr} decreases, but not monotonically. This was found for all the eleven learning policy alternatives. In all evaluations the best figure for E_{tr} was found within 30 iterations. As might be expected the weights start to over-fit the training data by the time this best error is reached and the error on the test data improves. For this reason we stop the training after 20 iterations and select the weight set corresponding to the best value for E_{tr}.

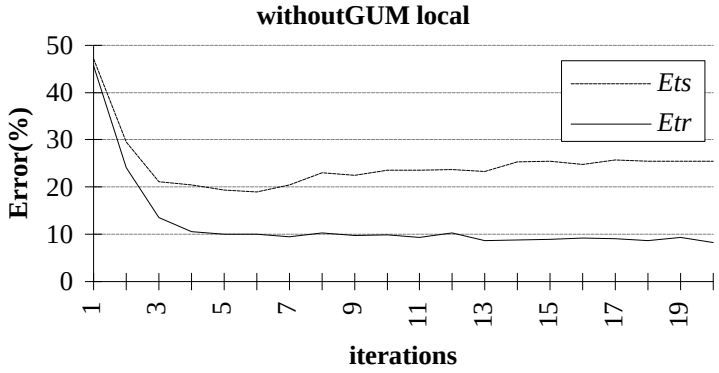

Figure 7.3: E_{tr} *and* E_{ts} *for the "Without GUM" policy for local weights.*

The overfitting phenomenon happens when the weights become too specialised for the *training* set and they loose the generality needed to solve the *test* set. The error on both the training set and the test set decreases during the first iterations, but the more the case-base learns about the training set, the more specific solutions it gives, so the error on the training set keeps decreasing, but the error on the test set starts increasing again.

This phenomenon had been found on both local and global weights but only with some policies (e.g., the combination "WithoutGUM" shown in Figures 7.4 a and b).

The graph shows that this is a well behaved learning process but there is evidently a need to stop learning early. In practice this can be achieved using a separate validation set as mentioned already.

In Figure 7.4(a) we show the behaviour of the case-base when the global weights are updated with the strategy S1: after a few iterations, the global weights saturate and the

error increases because there is no update decay. In Figure 7.4(b) we show the same behaviour with strategy S2: it can be seen that the saturation process is slightly less strong because of the presence of the decay function.

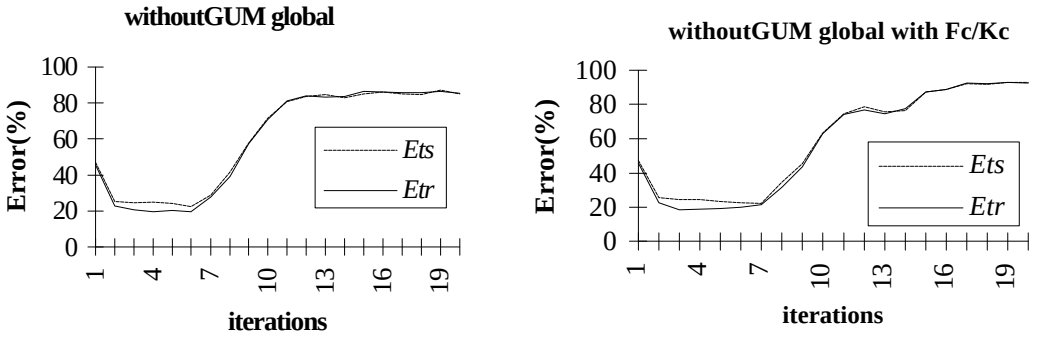

Figure 7.4 (a, b): E_{tr} *and* E_{ts} *for the combination "Without GUM" (global weights).*

The policy of iterating 20 times and keeping the weights that generated the smallest error is time consuming. A better policy would be to use a validation set to determine when to stop training but at present there are not sufficient cases in ISAC for this to be feasible. In this situation we could use a technique called *k*-fold cross-validation for early stopping as presented in (Hjorth, 1994) and below.

7.6.3 K-fold Cross-validation

The use of a validation set is useful to determine when it is the right time to stop iterating. When a simple validation (hold out validation) approach is used, some of these cases have to be withhold from the training set to be used in the validation set.

For example in a case-base of 150 cases, 100 cases could be used for training and 50 for validation. I.e. we stop training with the 100 cases when the error on the 50 cases starts to rise or we select the weights corresponding to the point when the error is smallest. This approach is fine if there are loads of cases for training but if cases are scarce then 50 cases are wasted.

In *k*-fold cross-validation all cases are used for training and for validation. One approach to cross validation is to try and guess the training error that will produce the lowest test error. The training data is divided into *k* sets and trained with *k-1* sets. The training is stopped when the error on the k^{th} set is minimised. The training error corresponding to this is noted. This is repeated *k* times and *k* estimates of test error and the *k* training errors corresponding to these points would be available. The training is stopped when the average of these training errors is reached. Thus all the data in training have been used.

Let us imagine this situation: there are 100 cases and a five fold cross-validation is done with 20 cases in the validation set each time. Let us suppose that for the five folds the following results are obtained:

Iteration	12	9	14	13	17
Training Set Error	12%	15%	12%	14%	12%
Test Set Error	18%	20%	19%	25%	22%

e.g. for the 1st fold the best error on the Test Set is 18%, this occurs at the 12th iteration. When all the data is used, training is stopped when the Training Set Error is (12+15+12+14+12)/5. Alternatively it could be stopped when iteration (12+9+14+13+17)/5 is reached.

7.7 Results

We tested the effectiveness of learning local weights with the combinations of the updating policies that we introduced previously; the results are shown in Table 7.1. All the 11 updating policies show a performance increase. The best increase of performance was recorded with the combination "WithoutGUM". On average, it seems that the combinations where the failure driven policies are dominant are more effective than the combinations where cues come from successful retrievals.

Table 7.1: *Error on the Test Set.*

Learning Policy	Before Learning	After Learning
Without GUM	47.2 %	22.1 %
Without GDU	47.2 %	23.7 %
Without BDM	47.2 %	26.7 %
Only BDM	47.2 %	29.7 %
Without BUU	47.2 %	29.9 %
Only Bad	47.2 %	31.8 %
All Four	47.2 %	31.9 %
Only BUU	47.2 %	32.7 %
Only Good	47.2 %	37.9 %
Only GDU	47.2 %	42.1 %
Only GUM	47.2 %	42.4 %

To test the robustness of the learning we also initialised the local weights with random values from 0.5 to 1.5 instead of having the starting weights all equal to 1. The performance increase was the same.

7.7.1 Local versus Global

We repeated the same experiments with the global weights and the four different strategies presented in Section 7.5. We were expecting a smaller scale increase in performance than with the local weights. This was true on average, but, sometimes, for strategy S1 the performance was better than the local weights. The results are shown in Figures 7.5 and 7.6.

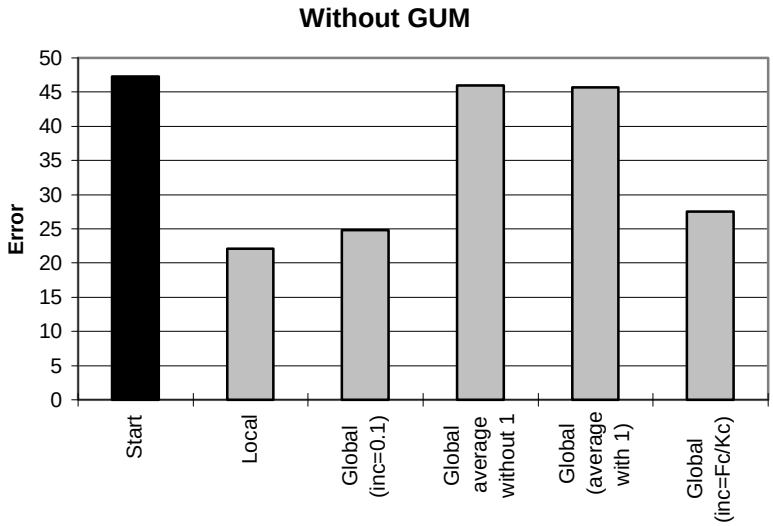

Figure 7.5: *Error of global and local weights for the "withoutGUM" combination.*

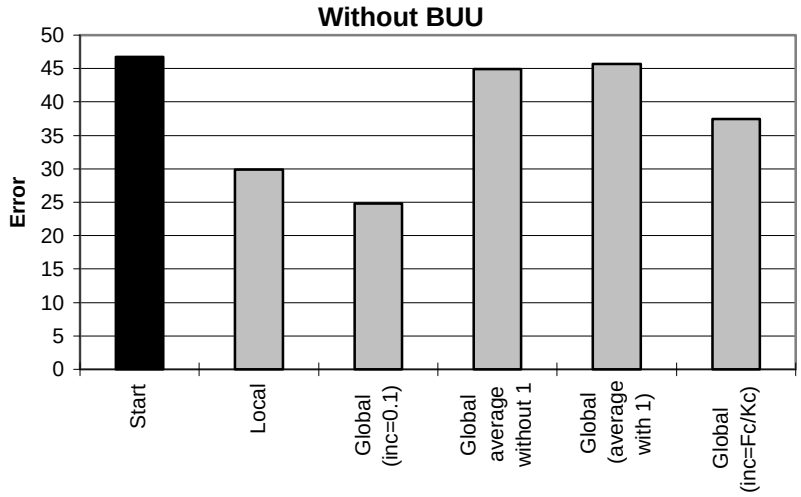

Figure 7.6: *Error of global and local weights for the "WithoutBUU" combination.*

Our best results occur with the "WithoutGUM" learning policy for local weights. This reduces the error from 47% to 22%. The best result with global weights is the 25% shown in Figure 7.6. We would expect the difference between the best local result and the best global result to be greater as more data becomes available for training.

7.7.2 Analysis of Context Sensitivity

Initial development on ISAC suggested that the parameters were quite context sensitive and an examination of the learned weights confirms this to be the case. The histograms in Figures 7.7 and 7.8 show distributions of weight values in the trained case-base for two specific parameters, "LevelsAvailable" and "CloseToBoundaries". In each case the range of weights has been divided into 10 intervals and the frequencies of weights in each interval are shown. Weights that remained unchanged at 1 have been removed. (Bonzano, Cunningham and Smyth, 1997,a).

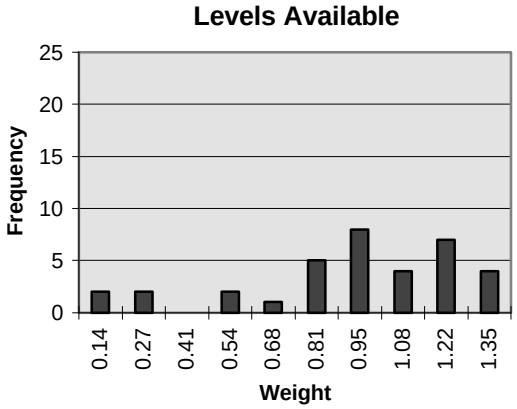

Figure 7.7: *The distribution of learned weights for the "LevelsAvailable" parameter.*

In Figures 7.7 and 7.8, the Y-axis "Frequency" indicates the number of cases that had the weight falling in the range reported in the X-axis.

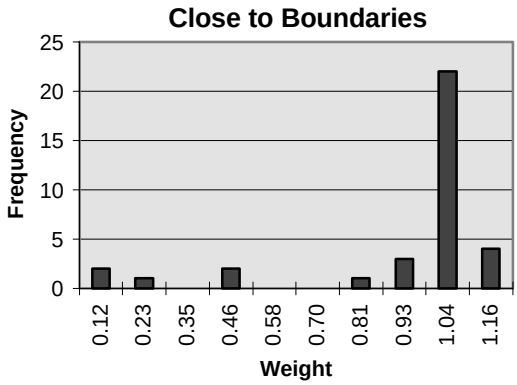

Figure 7.8: *The distribution of learned weights for the "CloseToBoundaries" parameter.*

The situation for the "LevelsAvailable" parameter shown in Figure 7.7 is the most typical, showing quite a spread in weight values across the case-base. Thus the relative importance of this parameter clearly changes from case to case and hence the parameter is of local importance across the case-base. This accords with the semantic of this parameter because it indicates whether other altitude levels are free and is only important when an altitude

manoeuvre is being considered. By comparison the "CloseToBoundaries" parameter shown in Figure 7.8 is evidently more global and again this makes sense in the problem domain. If an aircraft is close to the boundary of the controller's sector then this is always an important consideration.

7.8 Introspective Learning with Pivotal Cases

Smyth and Keane (1995) show that a case-base can be reduced in size without losing competence provided pivotal cases are not removed. A pivotal case is one that provides coverage not provided by other cases in the case-base. This is related to the idea of having a case-base of 'clean' cases where cases are hand picked to be of good quality and to cover particular areas of the problem domain.

It might be expected that a case-base composed of pivotal or 'clean' cases will not benefit much from introspective learning of parameter weights. Introspective learning depends on having adjacent cases so that the relevance of parameters can be determined. However, this redundancy will not exist in a pivotal case-base.

To verify this hypothesis we ran two experiments: one with a toy case-base where cases could be verified to be pivotal and one with the ISAC cases. Tests on the toy case-base supported the hypothesis. The cases available in ISAC are specially prepared clean cases so our hypothesis suggests that introspective learning will not work with these. From this 126 we prepared a case-base of 86 cases and a training set of 40 pivotal cases. For comparison we also prepared a training set of 40 cases taken from real traffic samples. After training the case-base with the training sets extracted from the case-base, we tested it with a test set also taken from real traffic samples. This experiment was repeated 22 times with different training sets. The results showed that training with pivotal (or clean) cases only produced an improvement of just 7% while training with random cases produced an improvement of 18% (see Table 7.2).

Table 7.2: *Pivotal versus non-pivotal Training Set.*

Training Set	E_{ts} (before)	E_{ts} (after)
pivotal	39 %	32 %
real	39 %	21 %

This supports our hypothesis that introspective learning of parameter weights exploits redundancy in the case-base and there is little redundancy in a case-base of pivotal or clean cases.

7.9 Conclusions

Learning local parameter weights greatly improves retrieval in ISAC. Our central conclusions are:

- Because of the context sensitivity of parameters, local parameter weights are more effective than global weights. We have shown that, for many parameters in ISAC, the learned local weights vary considerably. This is predicted by our understanding that the importance of many parameters in this domain is context sensitive. Presumably this varies from problem to problem, however using local rather than global weights has definitely been helpful here.
- Failure driven learning is most effective and the best policy is "WithoutGUM". This learning policy reduces the error in ISAC from 47% to 22%. It appears that failure driven rather than success driven learning contributes most to this improvement. This effect is not reported elsewhere so we need to determine why this is the case with ISAC.
- The learning process can overfit to the training set so an early stopping policy is needed. A validation set can be used to achieve this.

We have also verified that introspective learning of parameter weights does not work well when the cases used for training are pivotal. This is predicted by our understanding of the need for redundancy in the case-base for introspective learning. So this finding should be true in general.

In the future we propose to explore whether these findings generalise to other domains. We also propose to explore any variation in performance between global and local weights as the size of the case-base increases.

Chapter 8

Results and Evaluation

One of the most controversial steps in the development of ISAC has been its evaluation. In the air traffic control domain there is the saying - “ask *six* controllers to solve a conflict and you will get *seven* different answers”. This is obviously an exaggeration but it gives an idea of how subjective the evaluation of a solution given either by a controller, or the system itself, is. Consequently, the evaluation of this expert system is difficult.

The program of research between Trinity College, Dublin and Eurocontrol Experimental Centre, Paris was intended to investigate the use of CBR to augment the capability of an aircraft to carry out elaborate manoeuvres to avoid conflicting with others. One of the research themes that was considered the most important was the validation of the method used in view of the safety-critical nature of the overall problem and the definition of confidence figures for solutions given by the system. These issues are treated in this chapter with an analysis of the performance of the system. The evaluation recommended by an experimental psychologist is explained and the different steps to evaluate ISAC are discussed.

8.1 The Tests

The tests done with the controllers are intended to evaluate the performance of ISAC from two different points of view: the correctness of the solution suggested by the system and the reduction of the controller’s workload with the system implemented. The working tool that the controllers use is HIPS, which is embedded in the GHMI environment. ISAC does not change the global behaviour of the system, apart from a slight speed reduction which is acceptable at this prototypical level.

All the tests done take into consideration the three case representations introduced earlier. The case-base and the set of cases used for evaluation are based on the knowledge and preference of only one controller and not from a collective decision of all the controllers. The case structure does not have any constraints even if this possibility is available. All parameters have the same weight unless otherwise stated. All the traffic samples came from

en-route sectors, heavily conditioning the parameters used for the case description and the solutions of the conflicts. CBR can be made to work in any kind of sector, but the parameters describing a conflict and the solution to a conflict change with the type of sector.

8.1.1 The People that Evaluated the System

The system has been evaluated by air traffic controllers of different nationalities working in the Eurocontrol Experimental Centre in Paris. The typical career of these air traffic controllers starts with an ATC course, then it continues with 15-20 years experience on different airfields before joining Eurocontrol. Usually, in Eurocontrol, they work in real time simulations or in human-machine interface.

The work experience of a controller influences heavily the solutions that he gives. A controller who worked for a long period in a sector where the aircraft are usually cruising will use radar vectoring more often than a controller who worked in an airport sector where usually the safest and by far most common manoeuvre is a change in altitude.

For this reason, some questions have been asked to the controllers that took part to the simulation.

- For how long have they been a controller.
- For how long have they been approach, TMA or radar controller.
- Which was the last type of controlling that they did.

The answers to these questions could give an insight into the relationship between a controller's background and the solutions he gives.

The way a conflict is solved nowadays is heavily influenced by the fact that the only way of communicating is via voice messages. Moreover, the transmission is not always good. The absence of a datalink often forces the controller to reduce to the minimum the number of manoeuvres communicated to the pilot. This difficulty in communication is bad for two reasons. First, the controller often suggests a manoeuvre bigger than the one strictly necessary to avoid any further corrections that would mean a loss of time for the controller. Unfortunately, the oversized manoeuvre causes delays in the flight plan of the aircraft. Secondly, sometimes the controller waits for the conflict to evolve before taking a decision and often, what seems to be a conflict is not so in the end.

The approach that controllers had toward ISAC and the possibility of having a computer generated suggestion was almost always positive, even if they were sometimes a bit sceptical because controllers are aware of the complexity of the domain. Some controllers

were enthusiastic about the idea of a computer aiding the controller's decisions. They suggested further improvements that in some cases have been implemented, like the use of the BADA database for the acquisition of the performance parameters.

8.2 Initial Tests

The initial tests were not reliable because the solutions of the cases had been generated with an artificial set of rules and, as already said, were coherent but not realistic. Moreover there were only 50 cases in the case-base and no test cases were available, so the "LeaveOneOUT" evaluation technique had to be used. The results were very good: the system gave the correct solution in more than the 90% of the cases but as said they were not reliable as explained in Chapter 6.

The most important and helpful feedback from this evaluation came from verbal comments made by controllers during the testing sessions. Moreover, this evaluation was useful for the verification of the speed, efficiency and robustness of the tool in the hands of controllers.

In (Bonzano, Cunningham and Meckiff, 1996), it has been shown that the constraints are useful in speeding up the system but do not have any significant effect on the system competence as shown in Figure 8.1. The conclusion that can be drawn, i.e. that the use of constraints not only reduces the retrieval time but it increases the performance too, has to be tested with other case-bases before being confirmed and generalised. It should be noted that the better performance of the "OneInOne" case representation will not be repeated in the next steps of evaluation with more elaborated case-bases.

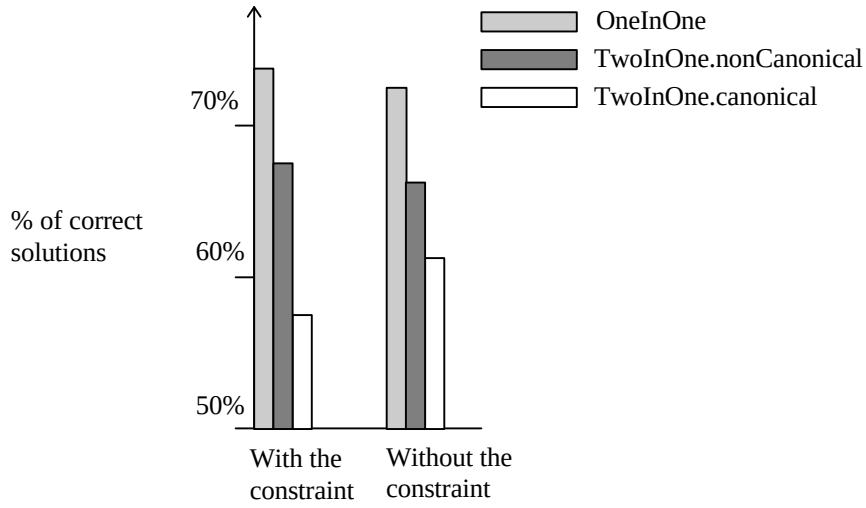

Figure 8.1: *The effectiveness of the constraints on the performance of the system.*

During this first step in the evaluation, some tests have been done in parallel to evaluate the speed performance of the spreading activation algorithm, with the results shown in Chapter 4. Other tests involved the construction of a decision tree based on the same data used by ISAC. In Chapter 5 it has been shown that ISAC performs better than its corresponding decision tree generated by C4.5.

8.3 Interim Step

A simplified traffic sample was used with the controllers for training and familiarisation purposes. All the traffic samples have been engineered to include a significant number and variety of conflicts. Tests took place in Summer 1996. All the sessions were individual and a different traffic sample was used for each run. Each solution given by a controller was recorded and compared with the solution given by ISAC and with the solution given by the other controllers. The controller could either accept or discard the solution suggested by ISAC.

The case-base used for the tests has been constructed by trying to put into a "case" form some of the rules learned during the sessions with the controllers. The 150 cases that constitute the case-base represent the knowledge of a particular controller and the solutions are generated from a set of rules. The output of a rule usually does not depend on all the parameters needed for the case description. For this reason, in a case-base generated from some basic rules, a lot of parameters will be set to a NIL value. This case-base had not been built for the traffic samples used for the tests, but was designed to be able to solve any kind of conflict in any type of sector. It will be highlighted later how naive this assumption was.

The test set consisted of 67 conflicts extracted from real traffic samples, but not all the conflicts had been solved by all the controllers for reasons of time. Only one controller solved all of the 67 conflicts and at least two controllers solved 42 conflicts. Four different situations have been tested.

- The "OneInOne" case representation was used and the solutions given by ISAC have been compared to the solutions given by the only controller who solved all the conflicts.
- The same case representation, "OneInOne", was used, but the solutions given by ISAC have been compared to the solutions given by all the controllers.
- The "TwoInOne" case representation has been used, and the solutions given by ISAC have been compared to the solutions given by the only controller who solved all the conflicts.

- The same case representation, “TwoInOne”, was used, but the solutions given by ISAC have been compared to the solutions given by all the controllers.

The system performance is reported in Table 8.1. The conflicts solved by only one controller have been identified with “One” whereas the tests done on the set of conflicts that have been solved by all the controllers are indicated by “All”. For the “One” situation, a suggestion was considered correct if the solution of the controller and the solution given by ISAC were the same. In the “All” situation, ISAC’s solution was considered correct if at least one of the controllers gave the same solution.

Table 8.1: *ISAC’s performance.*

Case Representation	Controller	% of correct solutions
OneInOne	One	49%
TwoInOne	One	71%
OneInOne	All	83%
TwoInOne	All	94%

It can be seen that the performance of the system with the “OneInOne” case representation is in general worse than the performance with the “TwoInOne” case representation. This trend, opposite to the one in the previous evaluation step, is confirmed in the final evaluation step and is supported by the intuitive consideration that the “OneInOne” case representation is less effective because less information about the global conflict and the other aircraft is stored in the case. For this reason, the “OneInOne” case representation will not be used in the final evaluation of the system. It will be possible to use it only when a realistic and well covered case-base will be made available. The case-base used in the final version, even if more complete, is still too small and oversimplified.

From a more accurate analysis of the results it was discovered that the majority of the errors made by ISAC were due to a wrong choice of the aircraft to manoeuvre but not to the incorrect type of manoeuvre. This was encouraging because the case-base used for the evaluation contained very little knowledge about the choice of the aircraft.

From the performance, it was clear that a lot of work still had to be done on extending the case-base, because 150 conflicts were not enough to characterise all the possible ATC conflicts, and on the parameters acquisition, because it was not always obvious how to convert into numbers what the controller sees on the radar screen.

8.4 Final Evaluation Step

The structure of the final test has been defined with the help of an expert on psychological experiments with the aim of gaining a better understanding of how the controller can interact with ISAC and how the system performs.

The tests consist of three steps:

1. A conflict is shown to the controller.
2. The solution for the conflict is requested from the controller. This step could be skipped if it would have been possible to pre-classify the bias of each controller by using conflicts that had already been solved and stored.
3. ISAC gives, on purpose, either a good or a bad solution to the conflict. The controller has to rank the given solution from 0 (very bad) to 7 (very good). The wrong solution is a random solution chosen from the solutions that were not selected as good solutions and it must be really bad, otherwise the results will not be reliable. Moreover, the controller is asked why does he think that it is a good/bad solution and what changes would he make to the solution to improve it. These questions are useful for building the adaptation function.

Step 2 is necessary because controllers sometimes accept sub-optimal solutions, as the controllers themselves confirm. By previously asking the controller for his solution, the risk of the controller passively accepting the solution suggested by ISAC is avoided. Moreover, with step 2, it is possible to evaluate whether or not the controller is biased: the percentage of altitude, speed, and horizontal manoeuvres in the controller's solutions is recorded. If a controller gives more than one possible solution, the weight of each solution is reduced by the number of solutions given.

Step 3, i.e. giving on purpose some bad and some good solutions, is necessary for different reasons:

- to make sure that ISAC gives the correct solution. All the marks that the controller gives to the solutions suggested by ISAC are averaged. The marks to the solutions that ISAC gives wrongly on purpose are averaged together, the same is done for the marks to the solutions that ISAC gives correctly on purpose. The greater the difference between the average of the good and the average of the bad marks, the better ISAC performed. The difference is visualised by the slope of the two lines in Figure 8.2: the steeper the lines, the better. Obviously, the average of the good solutions must be bigger than the average of the bad ones.

- Step 3 is also necessary to examine if some types of conflicts are solved more effectively than others. This could happen because either the controllers are biased or because some problems are simpler than others.
- Finally, step three could be necessary to check against the bias. For some conflicts ISAC would give a solution consistent with the bias whereas in other cases it would give a solution not consistent with the bias.

An introductory page was given to all the controllers that took part to the simulations. The way the tests were presented to the controllers was important because even a single misleading word could have influenced the controllers and nullified the results. The first part was intended to give a general background to the controller by explaining how ISAC works. The second part is reported below:

"Some conflicts will be shown to the controller. When a conflict is detected, the system will automatically display a solution. The controller will be asked to:

- *rate the correctness of the solution given by ISAC with a mark from 0 (very bad) to 7 (very good);*
- *say what he dis/liked about the given solution;*
- *if he would have given a different solution, and to specify which.*

Some of the solutions proposed may be deliberately incorrect. The duration of the evaluation will not take more than 30 minutes per controller."

The best thing would have been *not* to tell to the controllers that ISAC gives on purpose some of the wrong decisions, but there was either the risk of the controllers loosing confidence in a system with a low rate of good solutions or the possibility of the controllers giving good marks to bad solutions purely to give us encouragement. In both cases, the results of the evaluation would not have been valid. During the tests, the percentage of bad solutions given by ISAC on purpose was 50%.

A problem arises if a controller does not use the full range of marks, i.e. from 0 to 7. If this happens, there are two alternatives: either the controller's results are discarded, which is not possible, considering how difficult it was to get the assistance of a controller, or the marks that he gave have to be normalised to fill the interval from 0 to 7. During the tests, all the controllers made use of the full interval 0-7, eliminating the problem.

8.4.1 Results

The case-base used for the final round of tests has around 700 conflicts, i.e., 1400 cases in the "TwoInOne.nonCanonical" case representation, which has the best performance of all

three. This is the final case-base which also contains some conflicts stored with the purpose of solving some multiple aircraft conflicts.

Because of time restrictions, the bias has not been used to calculate the performance of ISAC. The controllers have been considered not biased and only the difference between correct and wrong solutions has been calculated.

Table 8.2: *How the solutions given by each controller are stored.*

Conflict	mark	solution
e1	4b	vector
e2	7g	dowBAW
e3	0b	dowIEA
e4	7g	dowDLH
e5	7g	dowBAW
e6	0b	dowEIN or vector
e7	7g	uppCOA
e8	0b	dowCPA
e9	7g	for dowBAL
“	0g	for horBoth
e10	7g	dowAFL
e11	7b	vector or descend any
e12	6g	dowEIN
e13	0b	dowSAS
e14	7g	for dowATQ
“	0g	for horBAW
e15	0b	vector

The wrong solutions that ISAC had to give on purpose was decided in advance and stored in a different file for each controller. A shell in which ISAC was embedded was taking the decision whether to give the wrong or correct solution depending on the name of the controller and on the name of the conflict. This shell always gave the correct solution to two conflicts that did not have any really wrong solution.

A table like Table 8.2 was created for each controller during the tests. The mark “4b” on the first line of the table means that the controller gave a mark “4” to a “Bad” solution given on purpose by ISAC. The mark “7g” on the second line, means that the controller gave a mark “7” to a “Good” solution. In this table, the Conflicts “e9” and “e14” have different marks for the two possible solutions that had been suggested by ISAC.

The results of the evaluation are in Figure 8.3. It can be seen that the mark that all the controllers gave to the wrong solutions suggested by ISAC was, on average, $\frac{2.03}{7}$, i.e. 29%, whereas the mark given to the good solutions was $\frac{5.49}{7}$, i.e. 78%. The mark given by the controller that generated the solutions for the cases in the case-base are respectively $\frac{2}{7}$, i.e. 28%, for the bad solutions and $\frac{7}{7}$, i.e. 100%, for the good solutions. This discrepancy in marks is due to the different preferences of each controller.

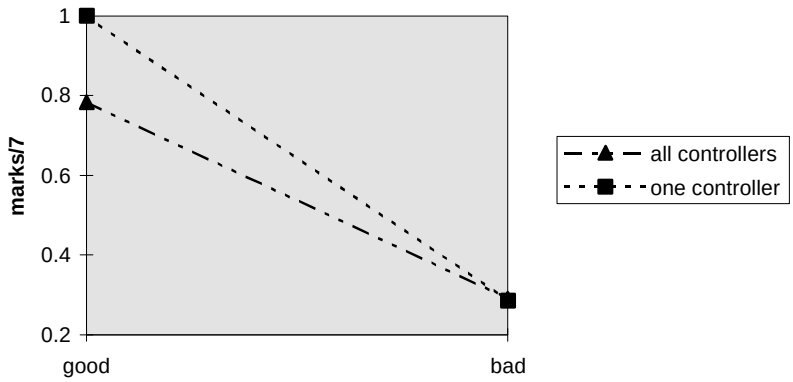

Figure 8.2: *Results of the evaluation.*

Figure 8.3(a) shows nine piecharts, one for each controller involved in the final tests, that report which are the preferences of the controller. Of the 15 conflicts that were shown to the controller the percentage of times that a certain type of manoeuvre chosen was recorded: horizontal manoeuvre (hor), vertical manoeuvre (alt) and speed manoeuvre (spe). Figure 8.3(b) shows the averaged percentage of preferred manoeuvres for all controllers.

From Figure 8.3(b) it can be seen that controllers prefer to use a vertical manoeuvre because it is the safest and the fastest to be communicated and implemented. From Figure 8.3(a) it can be seen that the preferences of each controller vary a lot. For example, Controller Two and Controller Eight have opposite habits in the use of vertical and horizontal manoeuvres. Some of these differences are due to the background of the controllers or to the attitude they have towards HIPS. For instance, Controller Two had been working for a long period in an overflying sector where the most common manoeuvre is the horizontal one because a lot of aircraft are cruising, whereas Controller Eight had been working for longer time in an approach sector where the most common manoeuvre is the vertical one.

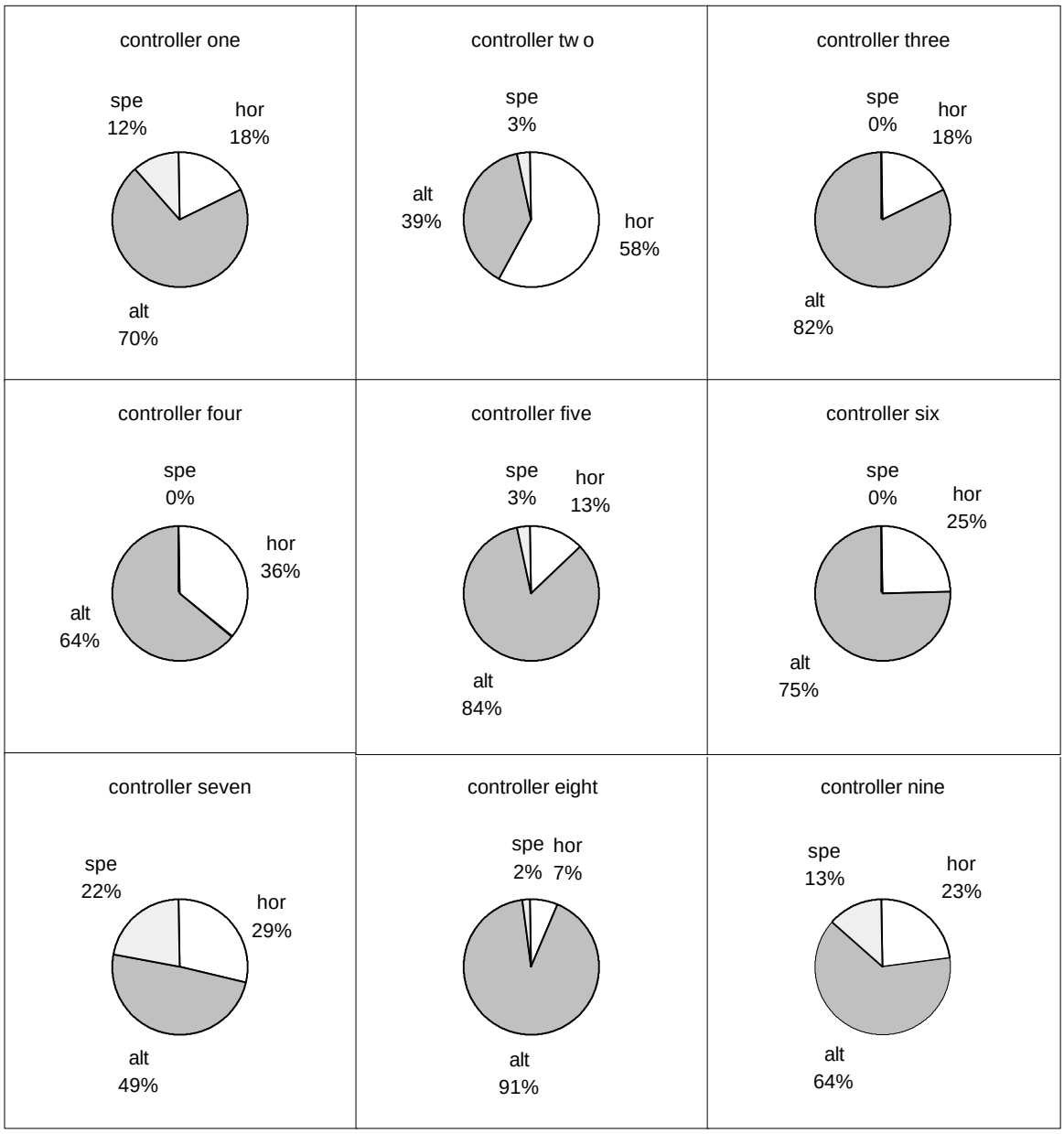

Figure 8.3(a): *Types of manoeuvres used by controllers to solve the test conflicts.*

As mentioned above, the attitude of the controllers toward HIPS influenced the results, too. Some controllers, sceptical about HIPS, were solving conflicts without using the help that HIPS could have provided and in this situation the most common manoeuvre was, again, a vertical manoeuvre because it is the one that needs the least visualisation. On the other hand, the controllers that liked "playing" with HIPS used a higher percentage of horizontal and speed manoeuvres because with HIPS, which has a superior graphical display of the conflict, more possible solutions are shown.

Some tests on introspective learning of the parameters weights have been done in parallel to the evaluation tests, during this final knowledge engineering step. These tests and the results obtained have already been described in Chapter 7.

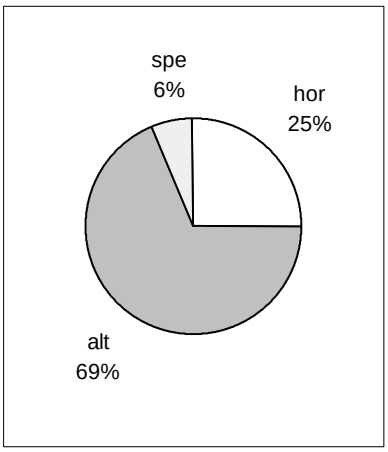

Figure 8.3(b): *Types of manoeuvres used by controllers in general.*

8.5 Multiple Aircraft Conflicts Tests

Because no MACs were available in the traffic samples used, some conflicts had to be built from scratch from already existing TACs and their consistency had to be checked by a controller. The problem, already present with TACs, of creating realistic conflicts is even more evident with MACs.

The evaluation tests for multiple aircraft conflicts have been done with the Look Ahead structure for MACs described in Section 6.6, chosen from the alternative structures presented in Chapter 5. In a Look Ahead structure, the MAC is decomposed into the constituent TACs that are solved independently, then a high-level analysis extracts from the solutions of the TACs the best solution for the MACs.

We are now going to show how the Look Ahead structure works when applied to a real MAC, shown in Figure 8.4. In this conflict, the aircraft FIN1121 is crossing the trajectory of the two aircraft SAS611 and SPAR64. At the same time, the aircraft SPAR64, behind, is catching the SAS611, which is in front and slower. All the three aircraft are flying at the same level.

The first step of the Look Ahead structure involves the resolution of the 3 constituent TACs: SAS611-SPAR64, FIN1121-SAS611 and FIN1121-SPAR64. The solutions found by ISAC for the three conflicts are, respectively, “lock the speed of SAS611 and SPAR64”, “climb FIN1121” and “climb “FIN1121”.

The second step of the Look Ahead algorithm consists of a high-level analysis of the three TACs solutions found and the extraction of a coherent one. Because there are three TACs, at least two solutions have to be extracted. Because the solutions for the two TACs FIN1121-SAS611 and FIN1121-SPAR64 are the same, this will be the final solution for the

MACs altogether with the solution for the SAS611-SPAR64 conflict. The Look Ahead structure for this MAC is shown in Figure 8.5.

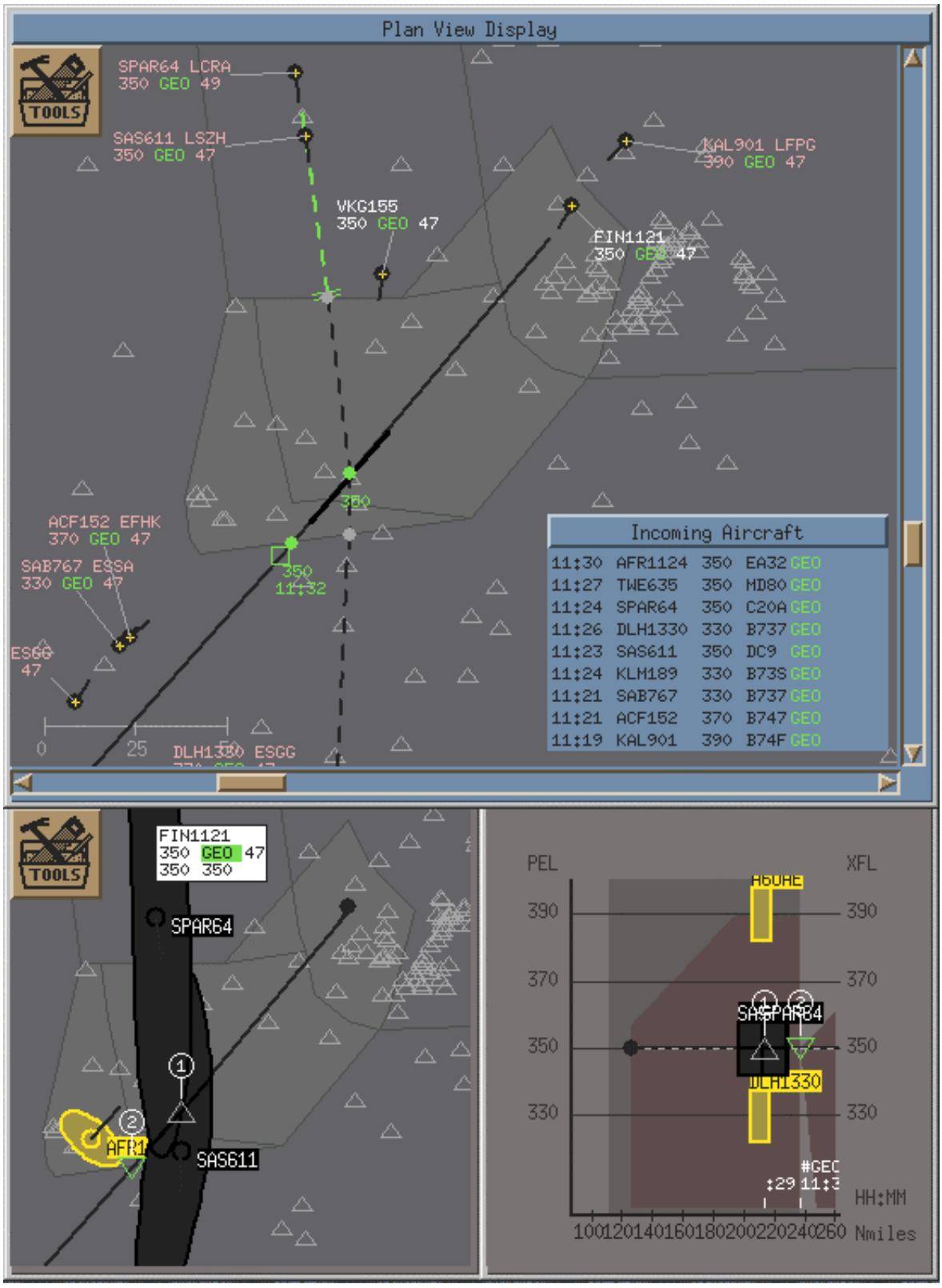

Figure 8.4: *A multiple aircraft conflict.*

If no solution in common to all the TACs was found, ISAC would have suggested the solution of the TAC closest in time as solution for the MAC.

As already said, the high-level analysis could be refined with the introduction of either more rules or a high-level case-base containing more general parameters. For example, a rule stating that the aircraft which is in conflict with all the others should be moved, could be added.

The conflict shown in Figure 8.4 is a *complex* MAC because there are 3 aircraft involved in 3 conflicts. The Look Ahead structure, and also the other two introduced in Chapter 5, works for both *simple* and *complex* MACs.

The MACs used for the evaluation have been displayed on a web page. The possible use of a browser to reduce time of the tests and to give the same treatment to all the controllers has been essential. The use of HIPS gives a range of choices to the controller, which is good in the TACs situation, but it is not as good in the more complex situation of the MACs where too many solutions would be available, making it impossible to test ISAC.

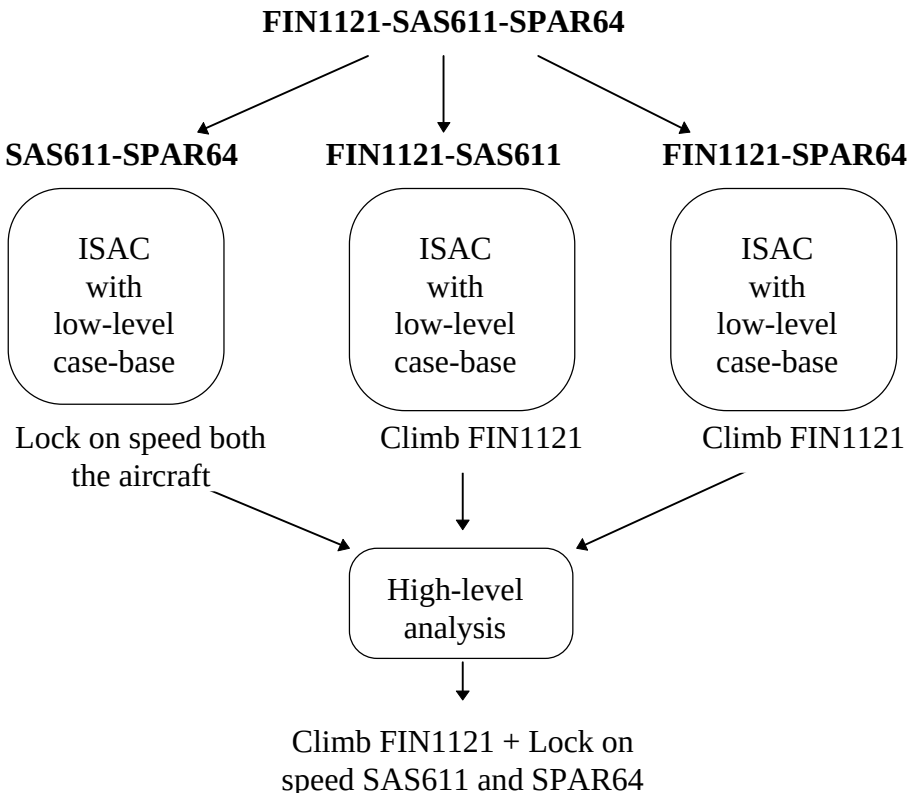

***Figure 8.5**: Look Ahead CBR for the sample MAC.*

In conclusion, we can say that the mechanism for solving MACs works, but we cannot say anything concrete about the coverage on MACs offered by the case-base.

8.6 Conclusions

In this chapter we analysed the performance of ISAC in solving conflicts, both TACs and MACs. Results regarding introspective learning of parameter weights, speed of the retrieval algorithm and comparison with decision trees have been treated earlier on.

The performance of a CBR system in general and of ISAC in particular depends on how well the case is described and on how densely and homogeneously the case space is populated. The results of the evaluation take account of how happy the controller is and not whether the correct solution has been chosen. The “correct” solution is a subjective decision and would vary from controller to controller. Because the solutions in the case-base have been given by a single controller, they reflect his preference.

The way of evaluating ISAC should change depending on its function. ISAC could be a training tool for controllers not experienced on a new sector, or it could be a standardising tool to homogenise the biased solutions that controllers might give.

As already mentioned, a controller usually has to train for more than one year on a certain sector before beginning to work on it. This training is necessary to teach the controller the optimal solutions for that particular sector, but it will influence his preferences and his behaviour when he will change sector. For example, if a controller has worked for some years in the approaching sector of a busy airport, where usually conflicts are solved immediately with a vertical manoeuvre, when this controller will change sector, he will be biased and will solve conflicts with a vertical manoeuvre.

Another factor that could influence the controller’s decision is the attitude towards the tools used in the simulation: some controllers examine very deeply the conflict, some others do not. Moreover some controllers already know the sector used for the tests, so they have an advantage over controllers who had never seen the sector.

A solution to avoid the controllers’ biases would be to build a case-base containing conflicts that happened in the same sector and to ask for their solutions from controllers who work on that sector. Biases among controllers working on the same sector are less influencing because, having learnt the same patterns, controllers will make the same assumptions on the conflicts.

Asking a group of controllers to come up with a globally accepted solution, one of the initial options, would take too much time. It could be assumed that the solutions generally given by a group of controllers working on the same sector could be synthesised by one of them, saving a lot of time.

Initial reaction to the work from controllers was positive, with the feeling that it is an appropriate line of research. The strongest point in favour of the tool was undoubtedly the fact that the controller remained entirely in control of the resolution process, while benefiting from the information provided by the HIPS displays and ISAC's suggestions.

Chapter 9

Conclusions and Future Work

The basic assumption underlying much of the work undertaken in the ATC research centres such as the Eurocontrol Experimental Centre in Paris, is that air traffic will continue to increase at a significant rate. Since most air traffic control facilities use practices and equipment which were developed at least 20 years ago, it is natural to assume that new approaches are needed for future scenarios with higher aircraft populations. Improvements due to the reorganisation of route structures will rapidly reach a limit in airspace at which point some fundamental changes will be needed. First of all, the utility of computer assistance will increase due also to increased precision in predicted trajectories of aircraft. The research presented in this thesis was intended to investigate the benefits from using CBR in order to help controllers in aircraft conflict resolution. Different research themes have been treated:

- the definition of the parameters that describe a conflict;
- the definition of an appropriate structure for the case-base that takes into account the real time nature of the problem;
- the possibility of solving two aircraft conflicts and multiple aircraft conflicts;
- the development of a retrieval mechanism and of the evaluation of a prototypical system;
- the validation of the method used in view of the safety-critical nature of the overall problem.

The system as it is now is integrated with HIPS which is embedded in a specific simulation environment for evaluation purposes, but it could be in theory integrated in any ATC tool, provided that this tool can supply ISAC with the necessary data for the conflict description. It is our opinion that only minor modifications would be needed to the *structure* of ISAC to be used in any type of sector with any ATC tool. The existence of a reliable case-base for the specific sector is a different and more fundamental problem.

9.1 Lessons Learned

During this research, several lessons have been learned. Some are typical of all CBR systems, whereas others are related to the knowledge engineering process and the problems of building a system that has to work in the real world.

A Reliable Case-Base is Essential

When the number of cases and the methodology for acquiring them were first discussed, it seemed that a case-base of 30-50 conflicts would have been big enough to start the tests and that these conflicts could be hand crafted. As already explained, both of these assumptions were wrong due to the complexity of the domain.

As described in (Leake, 1996, p.34), the most important component of a CBR system is its library of cases. This was particularly true for ISAC. First, the absence of an adaptation mechanism made it necessary to have a case-base with good coverage. Second, the complexity of the domain implied that the case-base contained lots of cases. Finally, having a lot of conflicts in a case-base is not enough: each conflict needs a solution, too. Moreover, the solutions must be coherent and must satisfy the controller.

Two conditions have to be respected in order to have an effective CBR system:

1. there must be enough cases drawn from the same sector. If cases are from the same sector and the case-base is used to solve conflicts on the same sector, the chances that a similar conflict is already in the case-base is higher. Having cases belonging to the same sector will reduce the complexity of the domain and the size of the case-base.
2. the solutions to the conflicts that are stored in the case-base must be given by the controllers that usually work on that sector. This will avoid the situation where controllers give different solutions to the same conflict either because they have different background or because they use the tools in a different way. Practices in use in individual sectors will ensure that controllers working on the same sector will give coherent solutions.

The tool used for displaying the conflicts heavily influenced the choice of the parameters and the solutions of the conflicts. The more realistic the tool, the more reliable the solution given by the controller. The decision whether to use gold standard cases or noisy cases depends on the way the case-base is acquired: if the case-base is built by hand, gold standard cases will be used, on the other hand, if the case-base is directly acquired from the sector, the case-base will contain more noisy data.

Some data had to be entered by hand but in an operational system all the data should be acquired electronically because the controllers will have neither time, nor inclination, to enter all the data by hand.

It was anticipated that ISAC would not have had to deal with incomplete data in the traffic samples used, but this was not true: the acquisition of some data was quite difficult and, often, the data that the controller was acquiring very easily could not be translated so easily into parameters for ISAC. Introspective learning techniques, presented earlier on, could help in reducing the negative effect of the lack of cases.

CBR is Better than RBS, but with Caveats

CBR reduces the knowledge engineering problem in comparison to RBS. The claim that CBR systems can be implemented faster than model-based systems is supported by different sources. For example, a study stated that it took two weeks to develop a case-based version of a system that took four months to build in rule-based form (Watson, 1994). Also, and more recently, developers confirmed that a rule-based system took more than eight times longer to develop than a case-based system with the same functionality. They also claim that the maintenance of the RBS is continual whereas the CBR system needs almost no maintenance (Watson, 1994). The time to effectively build the structure that handles the knowledge base in ISAC was short and almost no maintenance was necessary. Adding cases to the case-base when a conflict was not correctly solved was also simple.

The time to construct ISAC is shorter than the time that would have been necessary to build the equivalent rule-based system, but no comparison between the two algorithms could be done from the point of view of the performance. In fact, from the available literature on expert systems for ATC, it seems that the existing RBS are able to help the controllers only in certain situations but are not reliable in a general context. Moreover, their maintenance and update is very difficult.

The idea of using a cost function for estimating the effectiveness of a solution was considered but discarded because it would have implied building a complete rule-based system as complex and expensive as ISAC with the sole purpose of estimating the cost.

A very simple set of rules (2 rules) has been used in the hierarchical structure of ISAC. Some rules are also used in the adaptation step, which is very simple at this stage but could be increased if a more detailed solution had to be implemented. For these reasons, it has to be said that CBR should be complemented with some other systems such as RBS to build successful applications (Bayles et al., 1993).

The Knowledge Engineering Problem

At the beginning of the project, a report with some hypotheses on important CBR issues (Bonzano and Cunningham, 1995) was produced before having acquired a deep understanding of the problem of ATC. There were hypotheses on the structure of the system, on the programming language that could have been used, on the possible technical and theoretical issues and their corresponding solutions, etc. Some of these hypotheses were later revealed to be correct, whilst, others were not. For example, the speed of the system in giving real time solutions was considered one of the biggest issues at the beginning, but at the end it was not so. Moreover it was thought that the case-base acquisition would have been one of the easiest tasks, but, on the other hand it revealed to be one of the most difficult. These changes are just an indicator of how complex the process has been.

The structure of ISAC changed numerous times. Several decisions had to be taken and they did not only depend on the CBR nature of the problem, but also on its ATC nature. Moreover, not only the restrictions coming from the ATC domain had to be taken into consideration, but also the preferences of the controllers.

The Evaluation of ISAC is a Complex Issue

The performance of a CBR system in general and of ISAC in particular depends on how well the case is described and on how densely and homogeneously the case space is populated. The results of the evaluation take account of how happy the controller is and not whether the correct solution has been given. The “correct” solution is a subjective decision and could vary from controller to controller. Because the solutions in the case-base have been given by a single controller, they reflect his preference.

The way of evaluating ISAC should change depending on its function. ISAC could be a training tool for controllers not experienced on a new sector, or it could be a standardising tool to homogenise the biased solutions that controllers might give.

The initial approach was to build a very general system able to solve any kind of conflict in any sector, but the solutions are often strongly dependent on the sector where the conflict happens. ISAC could easily store the experience that each controller needs on a particular sector, but if it is kept too general it would loose this efficiency. Although controllers will always have to approve the suggested solutions, ISAC could become a means of giving a sort of standardised decision even if its main purpose remains the reduction of the controller’s workload.

Different Controllers can Give Different Solutions to the Same Conflict

As already mentioned, a controller usually has to train for more than one year on a certain sector before beginning to work on it. This training is necessary to teach the controller the preferred solutions for that particular sector, but it will not alter his preferences and his behaviour when he changes sector. For example, let us consider a controller that has worked for some years in the approaching sector of a busy airport where usually conflicts are solved immediately with a vertical manoeuvre because it is the kind of manoeuvre that needs the least monitoring. When this controller changes sector, he will always be biased and will solve conflicts with a vertical manoeuvre.

Another factor that could influence the controller's decision is the attitude towards the tools used in the simulation: some controllers examine very deeply the conflict, some other do not. Moreover some controllers already know the sector used for the tests, so they are advantaged to respect to the controllers who had never seen the sector.

Also the separation minima adopted in HIPS for visualising the no-go zones could change the solution given. If the separation minima is too big, HIPS will visualise conflicts that do not exist in reality and the shape of the no-go zone will change, nullifying the solutions. During a simulation, for example, there was a 27% increase in "speed" solutions when the horizontal separation was reduced from 10 to 6 nautical miles.

A solution to avoid the controllers' biases would be to build a case-base containing conflicts that happened in the same sector and to ask for their solutions to controllers who work on that sector. Biases among controllers working on the same sector are less influential because, having learnt the same patterns, controllers will make the same assumptions on the conflicts.

Asking a group of controllers to come up with a globally accepted solution, one of the initial options, would take too much time. It could be assumed that the solutions generally given by a group of controllers working on the same sector could be synthesised by one of them, saving a lot of time.

CBR can be Useful in the ATC Domain

The current version of ISAC is a working prototype that has been used to evaluate the performance of the case-base, the interactions with the controllers and the interface protocol with HIPS. This version has already been fielded and it is under refinement. It proved to be effective in helping controllers in doing their job and helped in suggesting a possible working scenario: "even if the structure of ISAC does not need substantial

alterations, the case-base will have to be specific for each particular sector. It could be built by acquiring all the conflicts that will happen on that particular sector for a period that could be as long as 6 months. Lots of conflicts will be very similar, but the case space of the conflicts that are typical to a specific sector will be well covered. This is the only way to avoid the problem of not having a real and good knowledge base. The case-base could contain a small core of generic cases common to all the sectors, and it could be then augmented with the sector specific cases over time."

The air traffic controllers in Eurocontrol Experimental Centre and in Dublin airport have been an integral part of designing ISAC and have been involved since the beginning of the project. We had no difficulty getting co-operation from the controllers, and the only negative point was our lack of ability in extracting and analysing their knowledge more qualitatively.

9.2 Directions for Further Research

The current version of ISAC has accomplished most of the initial goals. Nevertheless there still remain several unexplored paths, some already envisaged at the beginning of this research, but even more had been discovered during the knowledge engineering process.

Obviously, the most immediate necessity for ISAC is the construction of an effective case-base: big enough and with coherent solutions. To date we have tried to build a generic CBR reasoner that could work in any situation. Now that the prototypical version is ready, a system more specialised on a particular sector could be considered.

Some tests could be done to reduce the number of parameters used in the case description. This could be obtained either with introspective learning techniques or by eliminating the least discriminating parameters one by one and recalculating the performance each time.

More work on the hierarchical structure for multiple aircraft conflicts has to be done. The rules used in the high level analysis could be substituted with a small and more general case-base especially conceived for MACs. The parameters used in this case-base would be different, even if still related, to the ones used in the case-base for TACs.

Other tests could calculate the effective bias of the controllers. If done with the suggested new case-base, this could be useful to verify our assumption that the controllers working on the same sector are not biased, or at least are all biased in the same way.

Some interdisciplinary work could be done. For example, some available databases such as BADA, already used by ISAC, could provide more accurate parameters. Moreover, the

parameter that indicates the workload could be measured with biological data that check the controller's stress.

The web of pointers with symbolic values, for the implementation of the retrieval mechanism, was introduced with the purpose of having a fast system for real time simulations. If the speed problem occurs again, it might be solved with the help of Neural Networks (Naughton, 1995). An alternative solution could be the implementation of the web of pointers for the numeric values. This would require the division into ranges of the numeric values, which is not necessarily the best option and it would require a lot of calculations.

Finally, some lines of research have been suggested on the implementation of methods for deciding if a case space is well covered with the introduction of the two parameters AVE, the average distance of each case from all the others, and SMA, the distance of a case from the closest case. It would be interesting to evaluate the hypothesis that these two parameters do not work well in a non homogeneously covered case-base.

References

Aamodt A., Plaza E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variation and System Approaches, *AICOM*, Vol.7, No.1, pp.39-58.

Ashley K.D., McLaren B.M. (1995). Reasoning with Reasons in Case-based Comparisons, *Proceedings of the 1995 International Conference on Case-Based Reasoning*, Case-based Reasoning Research and Development, Lecture Notes in Artificial Intelligence, M. Veloso and A. Aamodt Eds., Springer Verlag, pp.133-144.

Bareiss E.R., Porter B.W., Murray K.S. (1989). Supporting Start-to-Finish Development of Knowledge Bases, *Machine Learning*, Vol.4, pp.259-283.

Barletta E.R., Hennessy C. (1989). Case Adaptation in Autoclave Layout Design, *Case-Based Reasoning: Proceedings of a Workshop on Case-Based Reasoning*, San Mateo, CA: Morgan, pp.203-207.

Barletta E.R. (1994). A Hybrid Indexing and Retrieval Strategy for Advisory CBR Systems Built with ReMind, *Proceedings of the 1994 European Workshop on Case-Based Reasoning*.

Bayles et al. (1993). Using Artificial Intelligence to Support Traffic Flow Management Problem Resolution, *Mitre Corporation*, MTR93W245.

Bhansali S., Harandi M.T. (1993). Synthesis of UNIX Programs Using Derivational Analogy, *Machine Learning*, Vol.10, pp.7-55.

Birnbaum L., Collins G., Brand M., Freed M., Krulwich B., Prior L. (1991). A Model-Based Approach to the Construction of Adaptive Case-Based Planning Systems, *Proceedings of the Case-Based Reasoning Workshop,* Washington D.C., USA, pp.215-224.

Bonzano A., Cunningham P. (1995). A review of CBR for use in Air Traffic Control, *Eurocontrol Experimental Centre Internal Report*, April 1995.

Bonzano A., Cunningham P., Meckiff C. (1996). ISAC: A CBR System for Decision support in Air Traffic Control, Advances in Case-Based Reasoning, *Proceedings of the 1996 European Workshop on Case-Based Reasoning*, I. Smith and B. Faltings Eds., Springer Verlag Lecture Notes in Artificial Intelligence, pp.44-57.

Bonzano A., Cunningham P., Smyth B. (1997,a). Using introspective learning to improve retrieval in CBR: A case study in air traffic control, Case-Based Reasoning Research and Development, *Proceedings of the 1997 International Conference on Case-Based Reasoning*, D.B. Leake and E. Plaza Eds., Springer Verlag, Lecture Notes in Artificial Intelligence, pp.291-302.

Bonzano A., Cunningham P., Smyth B. (1997,b). Learning feature weights for CBR: Global versus Local, AIIA 97: Advances in Artificial Intelligence, *Proceedings of AIIA'97*, M. Lenzerini Ed., Lecture Notes in Artificial Intelligence, Springer Verlag, pp.417-426.

Bos A. (1997). User Manual for BADA, Revision 2.5, *Eurocontrol Experimental Centre Note*, January 1997.

Caloo F. (1997). Elaboration of a method to assess psycho-physiological states of Air Traffic Controllers in Simulation, Laboratoire d'Anthropologie Appliquée de l'Université Réné Déscartes Paris en collaboration avec le Centre d'expertise IND et des controleurs du CEE, *Eurocontrol Experimental Centre Report*, December 1997.

Carbonell J.G., Veloso M. (1988). Integrating Derivational Analogy into a General Problem Solving Architecture, *Proceedings of the First Workshop on Case-Based Reasoning*, pp.104-124.

Cunningham P., Finn D., Slattery S. (1994). Knowledge Engineering Requirements in Derivational Analogy, Topics in Case-based Reasoning, Lecture Notes in Artificial Intelligence, S. Wess, K.-D. Althoff, M.M. Richter Eds., Springer Verlag, pp.234-245.

Cunningham P., Smyth B., Bonzano A. (1998). An Incremental Retrieval Mechanism for Case-Based Electronic Fault Diagnosis, to be published in the *Knowledge-Based Systems Journal*.

Cunningham P. (1998). CBR: Strengths and Weaknesses, to be presented at *The 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Castellón, Spain, June 1998.

Domeshek, E. (1992). Using Cases for Design Aiding, *AID Workshop 1992*.

Domeshek E., Kolodner J. (1992). Toward a Case-based Aid for Conceptual Design, *International Journal of Expert Systems*, Vol.4, No.2.

Domeshek E., Kolodner J. (1993). Using the Points of Large Cases, *AI EDAM 1993*, Vol.7, No.2, pp.87-96.

Doyle M. (1997). Web-based CBR in Java, B.A. (Mod.) Computer Science, Linguistic and German, Final Year Project, *Trinity College Dublin*.

Field A. (1985). International Air Traffic Control Management of the World's Airspace, Pergamon Press.

Fox S., Leake D.B. (1995). Using Introspective Reasoning to Refine Indexing, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp.391-397.

Goel A., Chandrasekaran B. (1989). Use of Device Models in Adaptation of Design Cases, *Proceedings of the Second Workshop on Case-Based Reasoning*, K. Hammond Ed., Morgan Kaufman.

Goel A. (1991). A Model-Based Approach to Case-Adaptation, *Proceedings of the 13th Annual Conference of Cognitive Science society*.

Goel A. (1992). Integrating Case-Based Reasoning and Model-Based: a Computational Model of Design Problem Solving, *AI Magazine,* Vol.13, No.2.

Gotteland J.B. (1995). Résolution Automatisée de conflicts en route, *Ecole Nationale de l'Aviation Civile,* S92, June 1995.

Hamrick L.Y., Arthur W.C., Reierson J.D. (1991). Advanced AERA Concepts: Proposed Problem Prediction and Problem Resolution Algorithms for the Automated Separation Function (ASF), *Mitre Corporation*, MTR-90W00140, July 1991.

Hanney K., Keane M., Smyth B., Cunningham P. (1995). Systems, tasks and adaptation knowledge: Revealing some revealing dependencies, Case-Based Reasoning Research and Development, *Proceedings of the 1995 International Conference on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, M. Veloso and A. Aamodt Eds., Springer Verlag, pp.461-470.

Hanney K., Keane M. (1996). Learning adaptation rules from a Case-Base, Advances in Case-Based Reasoning, *Proceedings of the 1996 European Workshop on Case-Based Reasoning*, I. Smith and B. Faltings Eds., Springer Verlag Lecture Notes in Artificial Intelligence, pp.179-192.

Hansen L.K., Larsen J., Fog T. (1997). Early Stop Criterion from the Bootstrap Ensemble, *Proceedings of ICASSP'97*, Munich, Germany, April 1997.

Hennessy D., Hinkle D. (1991). Initial Results from Clavier: A Case-Based Autoclave Loading Assistant, *Proceedings of the Third Workshop on Case-Based Reasoning*.

Hennessy D., Hinkle D. (1992). Applying case-based reasoning to autoclave loading, *IEEE Expert,* Vol.7, No.5, pp.21-26.

Hikle D., Toomey C.N. (1994). Clavier: Applying Case-Based Reasoning to Composite Part Fabrication, *Proceedings of the Sixth Innovative Applications of Artificial Intelligence Conference*, pp.55-61.

Hinrichs T.R. (1988). Towards an Architecture for Open World Problem Solving, *Proceedings of the First Workshop on Case-Based Reasoning*.

Hinrichs T.R., Kolodner J. (1991). The Roles of Adaptation in Case-Based Design, *Proceedings of the Third Workshop on Case-base Reasoning*.

Hjorth U. (1994). Computer Intensive Statistical Methods, Chapman and Hall, London.

Holl Nagel (1993). Human Reliability Analysis Context and Control, Academic Press, London.

ICAO (1994). Air Traffic Services, Annex 11, 10th edition, *International Civil Aviation Organisation*.

ICAO (1996). Rules of the Air and Air Traffic Services, Doc. 4444-RA/501, 13th edition, *International Civil Aviation Organisation*.

Irvine R. (1997). The GEARS Conflict Resolution Algorithm, *Eurocontrol Experimental Centre*, Report 321, November 1997.

Kambhampati S. (1989). Integrating Planning and Reuse: a framework for flexible plan reuse, *Proceedings of the Second Workshop on Case-Based Reasoning*, K. Hammond Ed., Morgan Kaufman.

Kambhampati S. (1989). Representational requirements for plan reuse, *Proceedings of the Second Workshop on Case-Based Reasoning*, K. Hammond Ed., Morgan Kaufman.

Kambhampati S. (1993). Supporting Flexible Plan Reuse, *Machine Learning for Planning*, S. Minton Ed.

Kambhampati S., Hendler J.A. (1992). A Validation-structure-based theory of plan modification and reuse, *Artificial Intelligence*, Vol.55, pp.193-258.

Kitano H. (1996). Nausicaä and the Sirens: A Tale of Two Intelligent Autonomous Agents, *IEEE Expert*, December 1996.

Kolodner J.L. (1988). Extending Problem Solver Capabilities through Case-based Inference, *Proceedings: Case-Based Reasoning Workshop*, J.L. Kolodner Ed.

Kolodner J.L. (1991). Improving Human Decision Making Through Case-Based Decision Aiding, *AI Magazine*, Vol.12, No.2, Summer 1991, pp.52-68.

Kolodner J.L. (1993). Case-Based Reasoning, Morgan Kaufmann Publishers.

Koton P. (1988). Reasoning about Evidence in Causal Explanation, *Proceedings of the First Workshop on Case-Based Reasoning*.

Koton P. (1988). Integrating Case-based and Causal Reasoning, *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: Erlbaum.

Laird J.E, Rosenbloom P.S., Newell A. (1986). Chucking in Soar: The Anatomy of a General Learning Mechanism, *Machine Learning*, Vol.1, No.1.

Laird J.E., Newell A., Rosenbloom P.S. (1987). Soar: An Architecture for General Intelligence, *Artificial Intelligence*, Vol.33, No.1.

Leake D.B., Kinley A., Wilson D. (1995). Learning to Improve Case Adaptation by Introspective Reasoning and CBR, Case-Based Reasoning Research and Development, *Proceedings of the First International Conference on Case-Based Reasoning*, M. Veloso and A. Aamodt Eds., Springer-Verlag, pp.229-240.

Leake D.B. (1996). Case-Based Reasoning: Experiences, Lessons and Future Directions, Chapter 2, AAAI Press, The MIT Press, 1996.

Levine A. (1971). Theory of Probability, Addison-Wesley.

Lewino F. (1995). Sécurité Aérienne, Comment réduire les risques, *Le Point*, N.1192.

Ljungberg M., Lucas A. (1992). The OASIS air traffic management system, *Australian Artificial Intelligence Institute*, Tech. Rep. 28, Melbourne, Australia, Aug 1992, also available at http://www.aaii.oz.au/research/techreports/abstracts/tn28.html.

Ly S. (1987). Prémières études en intelligence artificielle appliquées à la circulation aérienne: PLATONS, *Rapport CENA/R87-19*, December 1987.

Maes P. (1994). Agents that Reduce Work and Information Overload, *Communications of the ACM*, July 1994, Vol.37, No.7, pp.31-40.

Mark W. (1989). Case-Based Reasoning for Autoclave Management, *Proceedings of the Second Workshop on Case-Based Reasoning*, K. Hammond Ed., Morgan Kaufman.

Meckiff C., Gibbs P. (1994). PHARE Highly Interactive Problem Solver, *Eurocontrol Experimental Centre*, Report 273/94.

Meckiff C. (1994). Proposal for PATs Problem Solver Front-end Processing, *Eurocontrol Experimental Centre*, Internal Document, July 1994.

Micarelli A., Sciarrone F. (1996). A Case-Based System for Adaptive Hypermedia Navigation, *Proceedings of the 1996 European Workshop on Case-Based Reasoning*, Advances in Case-Based Reasoning, I. Smith and B. Faltings Eds., Springer Verlag Lecture Notes in Artificial Intelligence, pp.266-279.

Model Development Group (1995), RAMS System Overview Document, *Eurocontrol Experimental Centre*, December 95.

Muñoz-Avila H., Hüllen J. (1996). Parameter Weighting by Explaining Case-Based Planning Episodes, *Proceedings of the 1996 European Workshop on Case-Based Reasoning*, Advances in Case-Based Reasoning, I. Smith and B. Faltings Eds., Springer Verlag Lecture Notes in Artificial Intelligence, pp.280-294.

Naughton S., Cunningham P. (1995). Neural Networks for Case Retrieval in Case-Based Reasoning, *Proceedings of the Fifth Irish Neural Networks Conference*, Cardinal Press.

Nicolaon J., Tumelin J. (1992). ARC2000: Specification of the real time simulation, *Eurocontrol Experimental Centre*, Task AS06, December 1992.

Nosal M. (1977). Basic Probability and Application, W.B. Sainders Company.

Oehlmann R., Edwards P., Sleeman D. (1995). Changing the Viewpoint: Re-Indexing by Introspective Question, *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, Lawrence-Erlbaum and Associates, pp.381-386.

Planchon P., Angerand L., Ly S. (1988). Rapport de mission aux USA sur l'utilisation de l'Intelligence Artificielle pour l'ATC et conséquences pour les projets français, *CENA*, Report 88-16.

Perry T.S. (1997). In search of the future of Air Traffic Control, *IEEE Spectrum*, August 1997, pp.19-35.

Quinlan J.R. (1986). Induction of Decision Trees, *Machine Learning*, Vol.1, pp.81-106.

Quinlan J.R. (1993). C4.5 Programs for machine learning, Morgan Kaufmann Publishers.

Quinlan J.R. (1994). Comparing Connectionist and Symbolic Learning Methods, *Computational Learning Theory and Natural Learning Systems*, Vol.1, S.J. Hansen, G.A. Draftel and R.L. Rivest Eds., MIT Press, pp.445-456.

Quinlan J.R. (1997). C5.0, available at http://www.rulequest.com/see5-info.html

Ram A., Arkin R., Moorman K., Clark, R. (1992). Case-Based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters, *Autonomous Robotic Systems*, GIT-CC-92/57.

Richter M. (1995). The Knowledge Contained in Similarity Measures. Presented at the *1997 International Conference on Case-Based Reasoning.* http://wwwagr.informatik.uni-kl.de/~lsa/CBR/RichterSlides.ps

Rougegrez-Loriette S. (1994). Prediction de processus a partir de comportements observes: Le systeme REBECAS, *These de Doctorat de l'Université de Paris*.

Saltzburg S.L. (1991). A Nearest Hyperrectangle Learning Method, *Machine Learning*, Vol.1.

Sharma S., Sleeman D. (1988). REFINER: a Case-Based Differential Diagnosis Aid for Knowledge Acquisition and Knowledge Refinement, *Proceedings of EWSL-88*, D. Sleeman Ed., Ritman: London, pp.201-210.

Shively C., Schwamb K.B. (1984). AIRPAC: Advisor for the Intelligent Resolution of Predicted Aircraft Conflicts, *Mitre Corporation*, MTR-84W164, October 1984.

Simpson R.L. (1985). A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute mediation, Ph.D. Thesis, *Georgia Institute of Technology*, School of Information and Computer Science, GIT-ICS-85/18.

Smyth B., Cunningham P. (1992). A Blackboard Based, recursive case-based reasoning system for software development, *Proceedings of 5th Irish Conference on Artificial Intelligence and Cognitive Science*, pp.179-194.

Smyth B., Cunningham P. (1992). Déjà-Vu: A Hierarchical Case-Based Reasoning System for Software Design, *Proceedings of 10th European Conference on Artificial Intelligence*, Vienna, B. Neumann Ed., Wiley & Son, pp.587-589.

Smyth B., Cunningham P. (1993). Complexity of Adaptation in Real-World Case-Based Systems, *Proceedings of the 6th Irish Conference on Artificial Intelligence and Cognitive Science*, pp.229-240.

Smyth B., Keane M. (1994). Retrieving Adaptable Cases, M. Richter, S. Wess and K.-D. Dieter Eds., *Topics on Case-Based Reasoning*, Lecture Notes on Artificial Intelligence, Springer Verlag, pp.209-220.

Smyth B., Keane M.T. (1995). Remembering to Forget: A Competence Preserving Case Deletion Policy for CBR Systems, *Proceedings of IJCAI-95*, Montreal, Canada, pp.377-382.

Stefik M. (1981). Planning and Meta-Planning, *Artificial Intelligence*, Vol.16, pp.141-170.

Sycara E.P. (1987). Resolving Adversarial Conflicts: An Approach to Integrating Case-Based and Analytic Methods, Ph.D. Thesis, *Georgia Institute of Technology*, GIT-ICS-87/26.

Sycara E.P., Navinchandra D. (1989). A Process Model of Experience-Based Design, *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*.

Sycara E.P., Navinchandra D. (1991). Influences: A Thematic Abstraction for Creative Use of Multiple Cases, *Proceedings of the Third Workshop on Case-Based Reasoning*.

Thompson V. (1997). New reasoning engines and intelligent agents help companies manage their enterprise-wide knowledge resources, *Byte*, September 1997, also available at http://www.byte.com/art/9709/sec17/art1.htm

Tumelin J. (1990). ASTA, *Eurocontrol Experimental Centre*, Note 02/90.

Veloso M., Carbonell J.G. (1989). Learning Analogies by Analogy-The Closed Loop of Memory Organisation and Problem Solving, *Proceedings of the Second Workshop on Case-based Reasoning*, K. Hammond Ed., Morgan Kaufman.

Veloso M. (1991). Efficient Non-linear Problem Solving using Casual Commitment and Analogical Replay, *Proceedings Thirteenth Annual Conference of Cognitive Science society*.

Veloso M., Carbonell J.G. (1991). Variable-Precision Case Retrieval in Analogical Problem Solving, *Proceedings of the Third Workshop on Case-based Reasoning*.

Veloso M. (1992). Learning by Analogical Reasoning in General Problem Solving, Ph.D. Thesis, CMU-CS-92-174, School of Computer Science, *Carnegie Mellon University*, Pittsburgh, USA.

Watson I.D. (1994), The Case for Case-Based Reasoning, *Proceedings of EPSRC/DRAL*, November 1994, pp.55-64.

Watson I.D. (1996) Case-Based Reasoning Tools: An Overview, *Proceedings of 2nd. UK CBR Workshop*, Progress in Case-Based Reasoning, I.D. Watson Ed., University of Salford, pp.71-88, also available at http://146.87.176.38/ai-cbr/Papers/cbrtools.doc.

Wetterschereck D., Aha D.W. (1995) Weighting Parameters, Case-Based Reasoning Research and Development, *Proceedings of The 1st International Conference on Case-Based Reasoning*, M. Veloso and A. Aamodt Eds., Springer-Verlag, pp.347-358.

Wettschereck D., Aha D.W., Mohri T. (1997). A review and empirical evaluation of parameter weighting methods for a class of lazy learning algorithms, to appear in *Artificial Intelligence Review*, also available at http://www.aic.nrl.navy.mil/~aha/.

Wiener E.L., Nagel D.C. (1988). Human Factors in Aviation, London, Academic Press, Chapter 19.

Zeghal K. (1994). Towards the Logic of an Airborne Collision Avoidance System which Ensures Coordination with Multiple Cooperative Intruders, International Council of Aeronautical Sciences, *Aircraft Systems Conference*, Anaheim.

Appendix A

Acquisition of the Case-Base

As said in Chapters 6 and 8, one of the biggest problems in the project of ISAC has been the acquisition of the case-base. As a temporary solution to the impossibility of creating a case-base from the real traffic samples, an HTML form has been prepared to add by hand the cases to the case-case. This appendix contains a short description of how this has been done.

A.1 Structure

The HTML form, whose code is below, gives the list of parameters that constitute the case-base with radio buttons for the possible values of the parameters. When the "submit" button is pressed the form sends its data to the PERL script `process_form.cgi` that analyses the data and sends it to the program `Convert`. This program reads the data and writes it into the case-bases stored as text files.

A.2 The Form for the Acquisition of the Case-Base

```
<HTML>
<TITLE>CBR</TITLE>
<BODY bgcolor=white>

<strong><FONT SIZE=5>Add this case to the CaseBase</FONT></strong>
<FONT SIZE=2>

<FORM METHOD="POST" ACTION="process_form.cgi">

<strong>CaseName  </strong> (WITHOUT SPACES)
<INPUT TYPE=text NAME=CaseName SIZE=20 MAXLENGTH=60> (optional)<BR>

<strong>HorConflConf</strong>
<INPUT TYPE=radio NAME=HorConflConf VALUE="crossing">crossing
<INPUT TYPE=radio NAME=HorConflConf VALUE="converging">converging
<INPUT TYPE=radio NAME=HorConflConf VALUE="headon">headon
<INPUT TYPE=radio NAME=HorConflConf VALUE="diverging">diverging
<INPUT TYPE=radio NAME=HorConflConf VALUE="NIL" checked>NIL
<BR>
<strong>AltitudeNow</strong>
<INPUT TYPE=radio NAME=AltitudeNow VALUE="different">different
<INPUT TYPE=radio NAME=AltitudeNow VALUE="same" >same
<INPUT TYPE=radio NAME=AltitudeNow VALUE="NIL" checked>NIL
<BR>
<strong>Priority</strong>
<INPUT TYPE=radio NAME=Priority VALUE="higher">higher
<INPUT TYPE=radio NAME=Priority VALUE="same" >same
<INPUT TYPE=radio NAME=Priority VALUE="lower">lower
<INPUT TYPE=radio NAME=Priority VALUE="NIL" checked>NIL
```

```
<BR>
<strong>Speed</strong>
<INPUT TYPE=radio NAME=Speed VALUE="faster">faster
<INPUT TYPE=radio NAME=Speed VALUE="same" >same
<INPUT TYPE=radio NAME=Speed VALUE="slower">slower
<INPUT TYPE=radio NAME=Speed VALUE="NIL" checked>NIL
<BR>
<table >
<td>
<FONT SIZE=2>
<strong>CloseToTOD(A)</strong>
<INPUT TYPE=text NAME=CloseToTOD SIZE=4 MAXLENGTH=20>
<BR>
<strong>CloseToBoundaries(A)</strong>
<INPUT TYPE=text NAME=CloseToBoundaries SIZE=4 MAXLENGTH=20>
<BR>
<strong>Manoeuvrability(A)</strong>
<INPUT TYPE=text NAME=Manoeuvrability SIZE=4 MAXLENGTH=20>
<BR>
<strong>AltIntention(A)</strong>
<BR>
     
<INPUT TYPE=radio NAME=AltIntention VALUE="stable">stable
<INPUT TYPE=radio NAME=AltIntention VALUE="descending">descending
<INPUT TYPE=radio NAME=AltIntention VALUE="climbing">climbing
<INPUT TYPE=radio NAME=AltIntention VALUE="NIL" checked>NIL
<BR>
<strong>EasyToExitHorizontally(A)</strong>
<BR>
     
<INPUT TYPE=radio NAME=EasyToExitHorizontally VALUE="veryEasy">veryEasy
<INPUT TYPE=radio NAME=EasyToExitHorizontally VALUE="easy" >easy
<INPUT TYPE=radio NAME=EasyToExitHorizontally VALUE="possible" >possible
<INPUT TYPE=radio NAME=EasyToExitHorizontally VALUE="difficult" >difficult
<INPUT TYPE=radio NAME=EasyToExitHorizontally VALUE="NIL" checked>NIL
<BR>
<strong>LevelsAvailable(A)</strong>
<BR>
     
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="none">none
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="yes" >yes
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="above">above
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="below">below
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="withSpaces">withSpaces
<INPUT TYPE=radio NAME=LevelsAvailable VALUE="NIL" checked>NIL
<BR>
<strong>Faster(A)</strong>
<INPUT TYPE=radio NAME=Faster VALUE="easy">easy
<INPUT TYPE=radio NAME=Faster VALUE="possible" >possible
<INPUT TYPE=radio NAME=Faster VALUE="difficult">difficult
<INPUT TYPE=radio NAME=Faster VALUE="NIL" checked>NIL
<BR>
<strong>Slower(A)</strong>
<INPUT TYPE=radio NAME=Slower VALUE="easy">easy
<INPUT TYPE=radio NAME=Slower VALUE="possible" >possible
<INPUT TYPE=radio NAME=Slower VALUE="difficult">difficult
<INPUT TYPE=radio NAME=Slower VALUE="NIL" checked>NIL
<BR>
</FONT>
</td>

<td>
<FONT SIZE=2>
<strong>CloseToTOD(B)</strong>
<INPUT TYPE=text NAME=CloseToTODB SIZE=4 MAXLENGTH=20>
<BR>
<strong>CloseToBoundaries(B)</strong>
<INPUT TYPE=text NAME=CloseToBoundariesB SIZE=4 MAXLENGTH=20>
<BR>
<strong>Manoeuvrability(B)</strong>
<INPUT TYPE=text NAME=ManoeuvrabilityB SIZE=4 MAXLENGTH=20>
<BR>
<strong>AltIntention(B)</strong>
<BR>
     
```

```
<INPUT TYPE=radio NAME=AltIntentionB VALUE="stable">stable
<INPUT TYPE=radio NAME=AltIntentionB VALUE="descending">descending
<INPUT TYPE=radio NAME=AltIntentionB VALUE="climbing">climbing
<INPUT TYPE=radio NAME=AltIntentionB VALUE="NIL" checked>NIL
<BR>
<strong>EasyToExitHorizontally(B)</strong>
<BR>
     
<INPUT TYPE=radio NAME=EasyToExitHorizontallyB VALUE="veryEasy">veryEasy
<INPUT TYPE=radio NAME=EasyToExitHorizontallyB VALUE="easy" >easy
<INPUT TYPE=radio NAME=EasyToExitHorizontallyB VALUE="possible" >possible
<INPUT TYPE=radio NAME=EasyToExitHorizontallyB VALUE="difficult" >difficult
<INPUT TYPE=radio NAME=EasyToExitHorizontallyB VALUE="NIL" checked>NIL
<BR>
```

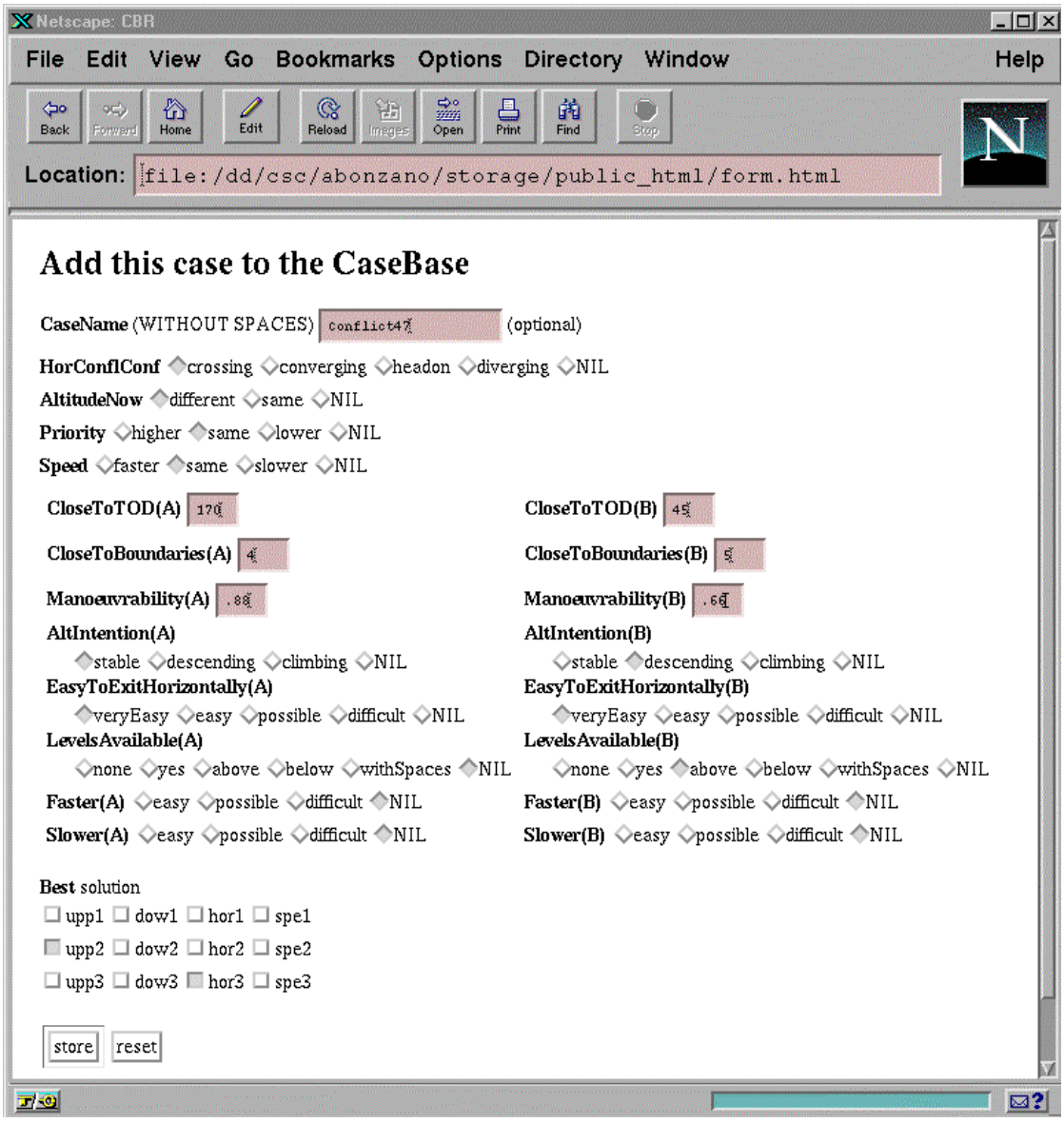

Figure A.1: *The form as shown by the browser.*

```
<strong>LevelsAvailable(B)</strong>
<BR>
     
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="none">none
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="yes" >yes
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="above">above
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="below">below
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="withSpaces">withSpaces
<INPUT TYPE=radio NAME=LevelsAvailableB VALUE="NIL" checked>NIL
<BR>
<strong>Faster(B)</strong>
```

```
<INPUT TYPE=radio NAME=FasterB VALUE="easy">easy
<INPUT TYPE=radio NAME=FasterB VALUE="possible" >possible
<INPUT TYPE=radio NAME=FasterB VALUE="difficult">difficult
<INPUT TYPE=radio NAME=FasterB VALUE="NIL" checked>NIL
<BR>
<strong>Slower(B)</strong>
<INPUT TYPE=radio NAME=SlowerB VALUE="easy">easy
<INPUT TYPE=radio NAME=SlowerB VALUE="possible" >possible
<INPUT TYPE=radio NAME=SlowerB VALUE="difficult">difficult
<INPUT TYPE=radio NAME=SlowerB VALUE="NIL" checked>NIL
<BR>
</FONT>
</td>
</table>
<BR>
<dt><strong>Best </strong>solution
     
<input type=checkbox name=Supp1 > upp1
<input type=checkbox name=Sdow1 > dow1
<input type=checkbox name=Shor1 > hor1
<input type=checkbox name=Sspe1 > spe1 ||
<input type=checkbox name=Supp2> upp2
<input type=checkbox name=Sdow2> dow2
<input type=checkbox name=Shor2> hor2
<input type=checkbox name=Sspe2> spe2 ||
<input type=checkbox name=Supp3> upp3
<input type=checkbox name=Sdow3> dow3
<input type=checkbox name=Shor3> hor3
<input type=checkbox name=Sspe3> spe3
<BR>
<INPUT TYPE=submit VALUE=store>
<INPUT TYPE=reset VALUE=reset>
</FORM>
<hr>
</BODY>
</HTML>
```

A.3 The PERL file `process_form.cgi`

```
#!/opt/perl5/bin/perl -w
unshift @ISA, ".";
use CGI;
$query = new CGI;

$CaseName= $query->param('CaseName');
$HorConflConf= $query->param('HorConflConf');
$AltitudeNow= $query->param('AltitudeNow');
$Priority= $query->param('Priority');
$Speed= $query->param('Speed');
$CloseToTOD= $query->param('CloseToTOD');
$CloseToTODB= $query->param('CloseToTODB');
$CloseToBoundaries= $query->param('CloseToBoundaries');
$CloseToBoundariesB= $query->param('CloseToBoundariesB');
$Manoeuvrability= $query->param('Manoeuvrability');
$ManoeuvrabilityB= $query->param('ManoeuvrabilityB');
$AltIntention= $query->param('AltIntention');
$AltIntentionB= $query->param('AltIntentionB');
$EasyToExitHorizontally= $query->param('EasyToExitHorizontally');
$EasyToExitHorizontallyB= $query->param('EasyToExitHorizontallyB');
$LevelsAvailable= $query->param('LevelsAvailable');
$LevelsAvailableB= $query->param('LevelsAvailableB');
$Faster= $query->param('Faster');
$FasterB= $query->param('FasterB');
$Slower= $query->param('Slower');
$SlowerB= $query->param('SlowerB');

$Supp1= $query->param('Supp1');
$Sdow1= $query->param('Sdow1');
$Shor1= $query->param('Shor1');
$Sspe1= $query->param('Sspe1');
$Supp2= $query->param('Supp2');
$Sdow2= $query->param('Sdow2');
$Shor2= $query->param('Shor2');
$Sspe2= $query->param('Sspe2');
```

```
$Supp3= $query->param('Supp3');
$Sdow3= $query->param('Sdow3');
$Shor3= $query->param('Shor3');
$Sspe3= $query->param('Sspe3');

if (!$CaseName ) { $CaseName="NoName"; }
if (! $CloseToTOD) { $CloseToTOD="-999"; }
if (!$CloseToTODB) { $CloseToTODB="-999"; }
if (!CloseToBoundaries) { $CloseToBoundaries="-999"; }
if (!$CloseToBoundariesB) { $CloseToBoundariesB="-999"; }
if (!$Manoeuvrability) { $Manoeuvrability="-999"; }
if (!$ManoeuvrabilityB) { $ManoeuvrabilityB="-999"; }
if (!$Supp1 ) { $Supp1="off"; }
if (!$Sdow1 ) { $Sdow1="off"; }
if (!$Shor1 ) { $Shor1="off"; }
if (!$Sspe1 ) { $Sspe1="off"; }
if (!$Supp2 ) { $Supp2="off"; }
if (!$Sdow2 ) { $Sdow2="off"; }
if (!$Shor2 ) { $Shor2="off"; }
if (!$Sspe2 ) { $Sspe2="off"; }
if (!$Supp3 ) { $Supp3="off"; }
if (!$Sdow3 ) { $Sdow3="off"; }
if (!$Shor3 ) { $Shor3="off"; }
if (!$Sspe3 ) { $Sspe3="off"; }

system "/home/ist/bnz/public_html/CreateCB/g $CaseName $HorConflConf
$AltitudeNow $Priority $Speed $CloseToTOD $CloseToTODB $CloseToBoundaries
$CloseToBoundariesB $Manoeuvrability $ManoeuvrabilityB $AltIntention
$AltIntentionB $EasyToExitHorizontally $EasyToExitHorizontallyB
$LevelsAvailable $LevelsAvailableB $Faster $FasterB $Slower $SlowerB $Supp1
$Sdow1 $Shor1 $Sspe1 $Supp2 $Sdow2 $Shor2 $Sspe2 $Supp3 $Sdow3 $Shor3 $Sspe3";
```

A.4 The program `Convert`

This is a C++ program that first checks that a new case with exactly the same parameters' values as one already in the case-base has not been submitted. Then the priority of the two aircraft is calculated using the rules in Section 4.3. Afterward, all the parameters that usually are calculated by the system GHMI when a case description is passed to ISAC have to be calculated because `Convert` substitutes itself to GHMI. For example, the parameter "Similar", usually calculated by GHMI when a conflict is detected, has to be calculated exactly in the same way by `Convert` using the data of the case description.

Finally the three case-bases, one for the "OneInOne" case representation and two for the "TwoInOne" case representation (canonical and non-canonical) are written with the names: `cb1in1`, `cbcanonical` and `cbnonCan`.

The code for `Convert`

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>

char lista[12][5]=
 {"upp1", "dow1", "hor1", "spe1", "upp2", "dow2", "hor2", "spe2", "upp3",
"dow3", "hor3", "spe3"};

char* DelEOL(char* tok)
{
 int i=0;
 while((tok[i]!=' ')&&(tok[i] != '\n')&&tok[i])
  i++;
```

```
 tok[i] = '\0';
 return tok;
}

char* other(char* value)
{
 if(strcmp(value,"different")==0)
  return "different";
 if(strcmp(value,"same")==0)
  return "same";
 if(strcmp(value,"better")==0)
  return "worse";
 if(strcmp(value,"worse")==0)
  return "better";
 if(strcmp(value,"higher")==0)
  return "lower";
 if(strcmp(value,"lower")==0)
  return "higher";
 if(strcmp(value,"faster")==0)
  return "slower";
 if(strcmp(value,"slower")==0)
  return "faster";
 if(strcmp(value,"NIL")==0)
  return "NIL";

 cout << "<BR> Error in function \"other\": not found " << value;
 return "nothing";
}

char* otherSol(char* value)
{
 if(strcmp(value,"upp1")==0)
  return "upp2";
 if(strcmp(value,"dow1")==0)
  return "dow2";
 if(strcmp(value,"upp2")==0)
  return "upp1";
 if(strcmp(value,"dow2")==0)
  return "dow1";
 if(strcmp(value,"hor1")==0)
  return "hor2";
 if(strcmp(value,"hor2")==0)
  return "hor1";
 if(strcmp(value,"spe1")==0)
  return "spe2";
 if(strcmp(value,"spe2")==0)
  return "spe1";
 return value;
}

main(int argc,char** argv)
{
 int i;
 char CaseName[32],Similar[5];
 FILE *cb1in1,*nonCan,*canonical,*cspace;

 cout << "Content-type:text/html\n\n";
 cout << "<HTML> <TITLE> form results </TITLE> <BODY>";

 // building string with all the values
 char AllTheValues[333];
 strcpy(AllTheValues,argv[2]);
 strcat(AllTheValues,"*");
 for(i=3;i<argc;i++)
  {
   strcat(AllTheValues,argv[i]);
   strcat(AllTheValues,"*");
  }
 cout << "<BR>Values passed from the form: " << DelEOL(AllTheValues) << "<BR>";

 // checking whether this string is already present in the case space
 char CaseInside[333];
 int AlreadyThere=0;
 cspace=fopen("CreateCB/CaseSpace","r");
 int aux=0;
```

```
while(fgets(CaseInside,332,cspace))
 {
  aux++;
  DelEOL(CaseInside);
  if(strcmp(CaseInside,AllTheValues)==0)
   AlreadyThere=1;
 }
fclose(cspace);

if(AlreadyThere)
 {
  cout << "<BR><BR><H1> Case ALREADY in the Case-base " <<
         " </H1> (not added)</BODY></HTML>";
  exit(0);
 }
else // case not in the case space: writing it
 {
  cspace=fopen("CreateCB/CaseSpace","a");
  fprintf(cspace,"%s\n",AllTheValues);
  fclose(cspace);
 }

cb1in1=fopen("CreateCB/cb1in1","a");
nonCan=fopen("CreateCB/cbnonCan","a");
canonical=fopen("CreateCB/cbcanonical","a");

// choosing first aircraft for canonical description:
// it is the one with the lower priority

int mark1=0,mark2=0,first=1;
// the higher the mark, the higher the priority
strcpy(Similar,"no");

// see if it is stable
if(strcmp(argv[12],"stable")==0)
 mark1++;
if(strcmp(argv[13],"stable")==0)
  mark2++;

// see if close to dest (TOD)
double aux1=atol(argv[6]),aux2=atol(argv[7]);
if((aux1!=-999)&&(aux2!=-999))
 {
  if((aux1-aux2)>10)
   mark1++;
  if((aux2-aux1)>10)
   mark2++;
 }

// see the manoeuvrability
aux1=atol(argv[10]),aux2=atol(argv[11]);
if((aux1!=-999)&&(aux2!=-999))
 {
  if(aux1<aux2-0.1)
   mark1++;
  if(aux2<aux1-0.1)
   mark2++;
 }

// see the priority due to the category
if(strcmp(argv[4],"higher")==0)
 mark1++;
if(strcmp(argv[4],"lower")==0)
 mark2++;

if(mark1<mark2)
 first=1;
else if(mark2<mark1)
 first=0;
else // if the aircraft are similar the first one becomes
     // AC1 because already under exam (less workload)
 {
  strcpy(Similar,"yes");
  first=1;
 }
```

```
double cTOD=atol(argv[6]),cTODb=atol(argv[7]);
double cBound=atol(argv[8]),cBoundb=atol(argv[9]);
double Man=atol(argv[10]),Manb=atol(argv[11]);

// beginning HTML page and finding the name of the case
if(strcmp(argv[1],"NoName")!=0)
 strcpy(CaseName,argv[1]);
else
 {
  int IntNumber;
  char number[9];
  FILE *Fnumber;
  Fnumber=fopen("CreateCB/number","r");
  fscanf(Fnumber,"%s",number);
  fclose(Fnumber);
  Fnumber=fopen("CreateCB/number","r");
  fscanf(Fnumber,"%d",&IntNumber);
  fclose(Fnumber);
  Fnumber=fopen("CreateCB/number","w");
  fprintf(Fnumber,"%d",IntNumber+1);
  fclose(Fnumber);
  strcpy(CaseName,"Case");
  strcat(CaseName,number);
 }

cout << "<hr> <FONT SIZE=19> <strong> " << CaseName
     << " </strong></FONT>added to the CaseBase<BR>";

// Calculating Solution
int FirstToPut=1;
char sol[200],sol1[200];
for(i=22;i<34;i++)
 if(strcmp(argv[i],"on")==0)
  if(FirstToPut)
   {
    FirstToPut=0;
    strcpy(sol,lista[i-22]);
    strcpy(sol1,otherSol(lista[i-22]));
    // otherSol calculates the solution for the second aircraft
   }
  else
   {
    strcat(sol,"&");
    strcat(sol1,"&");
    strcat(sol,lista[i-22]);
    strcat(sol1,otherSol(lista[i-22]));
   }

// cb1in1 (A)
fprintf(cb1in1,"@n %s(A)",CaseName);
fprintf(cb1in1,"\nHorConflConf %s",argv[2]);
fprintf(cb1in1,"\nAltitudeNow %s",argv[3]);
fprintf(cb1in1,"\nAltConfiguration %s",argv[12]);
fprintf(cb1in1,"\nSpeed %s",argv[5]);

if(cTOD==-999)
 fprintf(cb1in1,"\nCloseToTOD NIL");
else
 fprintf(cb1in1,"\nCloseToTOD %s",argv[6]);

if(cBound==-999)
 fprintf(cb1in1,"\nCloseToBoundaries NIL");
else
 fprintf(cb1in1,"\nCloseToBoundaries %s",argv[8]);

if(Man==-999)
 fprintf(cb1in1,"\nManoeuvrability NIL");
else
 fprintf(cb1in1,"\nManoeuvrability %s",argv[10]);

fprintf(cb1in1,"\nPriority %s",argv[4]);
fprintf(cb1in1,"\nEasyToExitHorizontally %s",argv[14]);
fprintf(cb1in1,"\nLevelsAvailable %s",argv[16]);
fprintf(cb1in1,"\nFaster %s",argv[18]);
```

```
fprintf(cb1in1,"\nSlower %s",argv[20]);
fprintf(cb1in1,"\n@s %s\n\n",sol);

// cb1in1 (B)
fprintf(cb1in1,"@n %s(B)",CaseName);
fprintf(cb1in1,"\nHorConflConf %s",argv[2]);
fprintf(cb1in1,"\nAltitudeNow %s",other(argv[3]));
fprintf(cb1in1,"\nAltConfiguration %s",argv[13]);
fprintf(cb1in1,"\nSpeed %s",other(argv[5]));

if(cTODb==-999)
 fprintf(cb1in1,"\nCloseToTOD NIL");
else
 fprintf(cb1in1,"\nCloseToTOD %s",argv[7]);

if(cBoundb==-999)
 fprintf(cb1in1,"\nCloseToBoundaries NIL");
else
 fprintf(cb1in1,"\nCloseToBoundaries %s",argv[9]);

if(Manb==-999)
 fprintf(cb1in1,"\nManoeuvrability NIL");
else
 fprintf(cb1in1,"\nManoeuvrability %s",argv[11]);

fprintf(cb1in1,"\nPriority %s",other(argv[4]));
fprintf(cb1in1,"\nEasyToExitHorizontally %s",argv[15]);
fprintf(cb1in1,"\nLevelsAvailable %s",argv[17]);
fprintf(cb1in1,"\nFaster %s",argv[19]);
fprintf(cb1in1,"\nSlower %s",argv[21]);
fprintf(cb1in1,"\n@s %s\n\n",sol1);

// NonCanonical (1)
fprintf(nonCan,"@n %s_1",CaseName);
fprintf(nonCan,"\nHorConflConf %s",argv[2]);
fprintf(nonCan,"\nPriority %s",argv[4]);
fprintf(nonCan,"\nAltitudeNow %s",argv[3]);
fprintf(nonCan,"\nSpeed %s",argv[5]);
fprintf(nonCan,"\nAltConfiguration(A) %s",argv[12]);

if(cTOD==-999)
 fprintf(nonCan,"\nCloseToTOD(A) NIL");
else
 fprintf(nonCan,"\nCloseToTOD(A) %s",argv[6]);

if(cBound==-999)
 fprintf(nonCan,"\nCloseToBoundaries(A) NIL");
else
 fprintf(nonCan,"\nCloseToBoundaries(A) %s",argv[8]);

if(Man==-999)
 fprintf(nonCan,"\nManoeuvrability(A) NIL");
else
 fprintf(nonCan,"\nManoeuvrability(A) %s",argv[10]);

fprintf(nonCan,"\nEasyToExitHorizontally(A) %s",argv[14]);
fprintf(nonCan,"\nLevelsAvailable(A) %s",argv[16]);
fprintf(nonCan,"\nFaster(A) %s",argv[18]);
fprintf(nonCan,"\nSlower(A) %s",argv[20]);

fprintf(nonCan,"\nAltConfiguration(B) %s",argv[13]);

if(cTODb==-999)
 fprintf(nonCan,"\nCloseToTOD(B) NIL");
else
 fprintf(nonCan,"\nCloseToTOD(B) %s",argv[7]);

if(cBoundb==-999)
 fprintf(nonCan,"\nCloseToBoundaries(B) NIL");
else
 fprintf(nonCan,"\nCloseToBoundaries(B) %s",argv[9]);

if(Manb==-999)
 fprintf(nonCan,"\nManoeuvrability(B) NIL");
else
```

```
 fprintf(nonCan,"\nManoeuvrability(B) %s",argv[11]);

fprintf(nonCan,"\nEasyToExitHorizontally(B) %s",argv[15]);
fprintf(nonCan,"\nLevelsAvailable(B) %s",argv[17]);

fprintf(nonCan,"\nFaster(B) %s",argv[19]);
fprintf(nonCan,"\nSlower(B) %s",argv[21]);
fprintf(nonCan,"\n@s %s\n\n",sol);

// NonCanonical (2)
fprintf(nonCan,"@n %s_2",CaseName);
fprintf(nonCan,"\nHorConflConf %s",argv[2]);
fprintf(nonCan,"\nPriority %s",other(argv[4]));
fprintf(nonCan,"\nAltitudeNow %s",other(argv[3]));
fprintf(nonCan,"\nSpeed %s",other(argv[5]));
fprintf(nonCan,"\nAltConfiguration(A) %s",argv[13]);

if(cTODb==-999)
 fprintf(nonCan,"\nCloseToTOD(A) NIL");
else
 fprintf(nonCan,"\nCloseToTOD(A) %s",argv[7]);

if(cBoundb==-999)
 fprintf(nonCan,"\nCloseToBoundaries(A) NIL");
else
 fprintf(nonCan,"\nCloseToBoundaries(A) %s",argv[9]);

if(Manb==-999)
 fprintf(nonCan,"\nManoeuvrability(A) NIL");
else
 fprintf(nonCan,"\nManoeuvrability(A) %s",argv[11]);

fprintf(nonCan,"\nEasyToExitHorizontally(A) %s",argv[15]);
fprintf(nonCan,"\nLevelsAvailable(A) %s",argv[17]);
fprintf(nonCan,"\nFaster(A) %s",argv[19]);
fprintf(nonCan,"\nSlower(A) %s",argv[21]);

fprintf(nonCan,"\nAltConfiguration(B) %s",argv[12]);

if(cTOD==-999)
 fprintf(nonCan,"\nCloseToTOD(B) NIL");
else
 fprintf(nonCan,"\nCloseToTOD(B) %s",argv[6]);

if(cBound==-999)
 fprintf(nonCan,"\nCloseToBoundaries(B) NIL");
else
 fprintf(nonCan,"\nCloseToBoundaries(B) %s",argv[8]);

if(Man==-999)
 fprintf(nonCan,"\nManoeuvrability(B) NIL");
else
 fprintf(nonCan,"\nManoeuvrability(B) %s",argv[10]);

fprintf(nonCan,"\nEasyToExitHorizontally(B) %s",argv[14]);
fprintf(nonCan,"\nLevelsAvailable(B) %s",argv[16]);
fprintf(nonCan,"\nFaster(B) %s",argv[18]);
fprintf(nonCan,"\nSlower(B) %s",argv[20]);
fprintf(nonCan,"\n@s %s\n\n",sol1);

// canonical
if(first)
 {
fprintf(canonical,"@n %s",CaseName);
fprintf(canonical,"\nHorConflConf %s",argv[2]);
fprintf(canonical,"\nPriority %s",argv[4]);
fprintf(canonical,"\nAltitudeNow %s",argv[3]);
fprintf(canonical,"\nSpeed %s",argv[5]);
fprintf(canonical,"\nAltConfiguration(A) %s",argv[12]);

if(cTOD==-999)
 fprintf(canonical,"\nCloseToTOD(A) NIL");
else
 fprintf(canonical,"\nCloseToTOD(A) %s",argv[6]);
```

```
if(cBound==-999)
 fprintf(canonical,"\nCloseToBoundaries(A) NIL");
else
 fprintf(canonical,"\nCloseToBoundaries(A) %s",argv[8]);

if(Man==-999)
 fprintf(canonical,"\nManoeuvrability(A) NIL");
else
 fprintf(canonical,"\nManoeuvrability(A) %s",argv[10]);

fprintf(canonical,"\nEasyToExitHorizontally(A) %s",argv[14]);
fprintf(canonical,"\nLevelsAvailable(A) %s",argv[16]);
fprintf(canonical,"\nFaster(A) %s",argv[18]);
fprintf(canonical,"\nSlower(A) %s",argv[20]);

fprintf(canonical,"\nAltConfiguration(B) %s",argv[13]);

if(cTODb==-999)
 fprintf(canonical,"\nCloseToTOD(B) NIL");
else
 fprintf(canonical,"\nCloseToTOD(B) %s",argv[7]);

if(cBoundb==-999)
 fprintf(canonical,"\nCloseToBoundaries(B) NIL");
else
 fprintf(canonical,"\nCloseToBoundaries(B) %s",argv[9]);

if(Manb==-999)
 fprintf(canonical,"\nManoeuvrability(B) NIL");
else
 fprintf(canonical,"\nManoeuvrability(B) %s",argv[11]);

fprintf(canonical,"\nEasyToExitHorizontally(B) %s",argv[15]);
fprintf(canonical,"\nLevelsAvailable(B) %s",argv[17]);
fprintf(canonical,"\nFaster(B) %s",argv[19]);
fprintf(canonical,"\nSlower(B) %s",argv[21]);
fprintf(canonical,"\n@s %s\n\n",sol);
 }
else
 {
fprintf(canonical,"@n %s",CaseName);
fprintf(canonical,"\nHorConflConf %s",argv[2]);
fprintf(canonical,"\nPriority %s",other(argv[4]));
fprintf(canonical,"\nAltitudeNow %s",other(argv[3]));
fprintf(canonical,"\nSpeed  %s",other(argv[5]));
fprintf(canonical,"\nAltConfiguration(A) %s",argv[13]);

if(cTODb==-999)
 fprintf(canonical,"\nCloseToTOD(A) NIL");
else
 fprintf(canonical,"\nCloseToTOD(A) %s",argv[7]);

if(cBoundb==-999)
 fprintf(canonical,"\nCloseToBoundaries(A) NIL");
else
 fprintf(canonical,"\nCloseToBoundaries(A) %s",argv[9]);

if(Manb==-999)
 fprintf(canonical,"\nManoeuvrability(A) NIL");
else
 fprintf(canonical,"\nManoeuvrability(A) %s",argv[11]);

fprintf(canonical,"\nEasyToExitHorizontally(A) %s",argv[15]);
fprintf(canonical,"\nLevelsAvailable(A) %s",argv[17]);
fprintf(canonical,"\nFaster(A) %s",argv[19]);
fprintf(canonical,"\nSlower(A) %s",argv[21]);

fprintf(canonical,"\nAltConfiguration(B) %s",argv[12]);

if(cTOD==-999)
 fprintf(canonical,"\nCloseToTOD(B) NIL");
else
 fprintf(canonical,"\nCloseToTOD(B) %s",argv[6]);

if(cBound==-999)
```

```
 fprintf(canonical,"\nCloseToBoundaries(B) NIL");
else
 fprintf(canonical,"\nCloseToBoundaries(B) %s",argv[8]);

if(Man==-999)
 fprintf(canonical,"\nManoeuvrability(B) NIL");
else
 fprintf(canonical,"\nManoeuvrability(B) %s",argv[10]);

fprintf(canonical,"\nEasyToExitHorizontally(B) %s",argv[14]);
fprintf(canonical,"\nLevelsAvailable(B) %s",argv[16]);
fprintf(canonical,"\nFaster(B) %s",argv[18]);
fprintf(canonical,"\nSlower(B) %s",argv[20]);
fprintf(canonical,"\n@s %s\n\n",sol1);
 }

// writing values on HTML file
cout << "<FONT SIZE=3> <BR> <TABLE border>";
for(i=1;i<argc;i++)
 {
  if((i%5)==0)
   cout << "<TR>";
  cout << "<TD>" << i << ": " << argv[i] << "</TD>";
 }
cout << "</TABLE> <BR>SOL: " << sol;

cout << "</FONT> <P> <A HREF=\"form.html\">Add another case</A><BR>";
cout << "</BODY></HTML>";
fprintf(cb1in1,"\n\n");
fprintf(nonCan,"\n\n");
fprintf(canonical,"\n\n");

fclose(cb1in1);
fclose(nonCan);
fclose(canonical);
}
```

Appendix B

Decision Trees and Discriminatory Power

As said in Chapter 5, ISAC can convert the case-base from the ISAC format into a format readable by C4.5. This has been useful for the comparison of the performance of ISAC and C4.5. The first section of this appendix treats the issues related to the construction of the decision tree by C4.5.

Moreover, ISAC gives the option of calculating the discriminatory power of the parameters involved in the case description. This helps in deciding which parameters to use in the retrieval process. The discriminatory power of the parameters involved in the final case description is treated in the second section of this appendix.

B.1 Decision Tree

C4.5 (Quinlan, 1993) is a classifier system written in C for the UNIX environment. C4.5 starts with a large set of cases that already have a solution and scrutinise them for patterns that allow the solutions to be reliably discriminated. In C4.5, the case-base is read with the command "`c4.5 -f namefile`", then the corresponding decision tree is built. For each "`namefile`", C4.5 will read 4 files:

- `namefile.names` that contains the parameters and the possible values;
- `namefile.data` with the case-base;
- `namefile.test` with the case description of a conflicts;
- `namefile.sol` with the solutions, one for each line, corresponding to the `namefile.test` file.

These 4 files are created by ISAC that automatically translates the files containing the case-base and the case structure into a format readable by C4.5. When the decision tree has been built, the case-base is not necessary anymore. The command "`consult -f namefile`" is used to test the tree built and all the cases in the file `namefile.test` are solved. The correct solutions and the one retrieved by C4.5 are stored into a file called "`results`" whose format can be read by the function "`analyse`" that gives the percentage of correct solutions.

ISAC automatically does all the “LeaveOneIN” and the “LeaveOneOUT” experiments by using system calls to “`c4.5`” and “`consult`”. The code of the user interface has been modified to make the system able to read the data and solutions directly from the test file. For simplicity, the parameters that have a NIL value, which is represented by a “?” in C4.5, are not used.

The decision trees shown in this appendix are generated with the default windowing and pruning parameters. For a guide on how to use the C4.5 system see pp.81-91 in (Quinlan, 1993). The output given by C4.5 while working with the case-base of 51 cases used in the first step of the knowledge engineering process is reported below. In this case-base, each case is described by 38 parameters.

```
C4.5 [release 5] decision tree generator
----------------------------------------
    Options:
      File stem <SymNum>
Read 51 cases (36 attributes) from cbase.data
Decision Tree:
RightExitNoGo(A) <= 3.16331 : spe3 (13.0/2.0)
RightExitNoGo(A) > 3.16331 :
|   TimeBefore(A) <= 17.9524 :
|   |   InFrontDirect(B) = no: alt2 (3.0)
|   |   InFrontDirect(B) = yes: hor2 (2.0)
|   TimeBefore(A) > 17.9524 :
|   |   GroundSpeed(A) <= 5.49952 :
|   |   |   TimeBefore(B) > 38.598 : alt3 (4.0)
|   |   |   TimeBefore(B) <= 38.598 :
|   |   |   |   Turning(A) <= -5.99759 : alt3 (3.0/1.0)
|   |   |   |   Turning(A) > -5.99759 : alt1 (17.0/2.0)
|   |   GroundSpeed(A) > 5.49952 :
|   |   |   HorConflConf = facing: hor3 (0.0)
|   |   |   HorConflConf = catching: hor3 (4.0)
|   |   |   HorConflConf = crossing: hor1 (5.0/2.0)

Tree saved
Evaluation on training data (51 items):
        Before Pruning           After Pruning
       ----------------   -------------------------
       Size      Errors   Size      Errors   Estimate
         16    7(13.7%)     16    7(13.7%)    (33.4%)
```

If the same experiment is repeated using only symbolic parameters, the output is:

```
C4.5 [release 5] decision tree generator
----------------------------------------
    Options:
      File stem <onlySym>

Read 51 cases (26 attributes) from SymSbonz.data

Decision Tree:
Cruising(A) = no:
|   Similar = no:
|   |   HorConflConf = facing: alt1 (0.0)
|   |   HorConflConf = crossing: alt1 (12.0/1.0)
|   |   HorConflConf = catching:
|   |   |   AltProfile(A) = stable: hor3 (0.0)
|   |   |   AltProfile(A) = descend: alt1 (3.0/1.0)
|   |   |   AltProfile(A) = climb: hor3 (3.0)
|   Similar = yes:
```

```
|   |   SpeedDec(A) = Big: alt1 (3.0/1.0)
|   |   SpeedDec(A) = VerySmall: alt3 (0.0)
|   |   SpeedDec(A) = Small:
|   |   |   AltProfile(B) = stable: hor1 (1.0)
|   |   |   AltProfile(B) = descend: alt3 (3.0)
|   |   |   AltProfile(B) = climb: alt3 (3.0)
Cruising(A) = yes:
|   InFrontSpace(A) = no: spe3 (13.0/2.0)
|   InFrontSpace(A) = yes:
|   |   SpeedDec(B) = Small: hor1 (6.0/3.0)
|   |   SpeedDec(B) = VerySmall: hor1 (0.0)
|   |   SpeedDec(B) = Big:
|   |   |   InFrontDirect(B) = no: alt2 (2.0)
|   |   |   InFrontDirect(B) = yes: hor2 (2.0)

Simplified Decision Tree:
Cruising(A) = no:
|   Similar = no:
|   |   HorConflConf = facing: alt1 (0.0)
|   |   HorConflConf = crossing: alt1 (12.0/2.5)
|   |   HorConflConf = catching:
|   |   |   AltProfile(A) = stable: hor3 (0.0)
|   |   |   AltProfile(A) = descend: alt1 (3.0/2.1)
|   |   |   AltProfile(A) = climb: hor3 (3.0/1.1)
|   Similar = yes:
|   |   SpeedDec(A) = Big: alt1 (3.0/2.1)
|   |   SpeedDec(A) = Small: alt3 (7.0/2.4)
|   |   SpeedDec(A) = VerySmall: alt3 (0.0)
Cruising(A) = yes:
|   InFrontSpace(A) = no: spe3 (13.0/3.6)
|   InFrontSpace(A) = yes:
|   |   SpeedDec(B) = Small: hor1 (6.0/4.3)
|   |   SpeedDec(B) = VerySmall: hor1 (0.0)
|   |   SpeedDec(B) = Big:
|   |   |   InFrontDirect(B) = no: alt2 (2.0/1.0)
|   |   |   InFrontDirect(B) = yes: hor2 (2.0/1.0)

Tree saved

Evaluation on training data (51 items):
        Before Pruning           After Pruning
       ----------------   --------------------------
       Size      Errors   Size      Errors   Estimate
         24    8(15.7%)     21    9(17.6%)    (39.1%)
```

The error obtained by using only symbolic parameters (15%) is slightly bigger than the error by using both numeric and symbolic parameters (13%). The pruning option has not been used for the comparison with C4.5 because pruning means generalising and the case-base used here is too small to have the results affected by generalisation.

The simplified decision tree generated with the latest version of case-base is below. The full tree has not been reported because it is too long. It can be clearly seen that the new decision tree is much more complex than the previous one not only because there are more parameters and more cases (1408 instead of 51), but because the case-base comes from real conflicts with real solutions and has not been generated with a simple set of rules as done in the first step of the knowledge engineering process.

```
C4.5 [release 5] decision tree generator     Wed Nov 26 11:37:03 1997
----------------------------------------

    Options:
      File stem <forC45>

Read 1408 cases (21 attributes) from forC45.data

Simplified Decision Tree:

AltConfiguration(A) = stable:
|   CloseToTOD(A) <= 0 :
|   |   CloseToTOD(B) <= 10 :
|   |   |   AltConfiguration(B) = stable:
|   |   |   |   HorConflConf = diverging:
|   |   |   |   |   Manoeuvrability(B) <= 0.87 : hor1 (4.5/4.0)
|   |   |   |   |   Manoeuvrability(B) > 0.87 : hor3 (8.0/1.3)
|   |   |   |   HorConflConf = headon:
|   |   |   |   |   Manoeuvrability(B) <= 0.81 :
|   |   |   |   |   |   Manoeuvrability(B) > 0.77 : spe2 (3.3/2.2)
|   |   |   |   |   |   Manoeuvrability(B) <= 0.77 :
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.69 : hor1 (9.6/4.7)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.69 :
|   |   |   |   |   |   |   |   Manoeuvrability(B) > 0.69 : hor2 (11.2/1.5)
|   |   |   |   |   |   |   |   Manoeuvrability(B) <= 0.69 :
|   |   |   |   |   |   |   |   |   Manoeuvrability(B) > 0.65 : hor3 (3.8/2.9)
|   |   |   |   |   |   |   |   |   Manoeuvrability(B) <= 0.65 :
|   |   |   |   |   |   |   |   |   |   CloseToBoundaries(A) <= 1 :
spe1(4.3/3.3)
|   |   |   |   |   |   |   |   |   |   CloseToBoundaries(A) > 1 : hor2
(4.0/1.2)
|   |   |   |   |   |   Manoeuvrability(B) > 0.81 :
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.69 : hor3 (4.3/3.4)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.69 : hor1 (11.2/1.5)
|   |   |   |   HorConflConf = converging:
|   |   |   |   |   Manoeuvrability(A) <= 0.7 : hor2 (5.8/5.2)
|   |   |   |   |   Manoeuvrability(A) > 0.7 :
|   |   |   |   |   |   Manoeuvrability(B) <= 0.76 : spe3 (4.8/2.1)
|   |   |   |   |   |   Manoeuvrability(B) > 0.76 : hor3 (2.0/1.0)
|   |   |   |   HorConflConf = crossing:
|   |   |   |   |   Speed = same: hor3 (17.7/8.2)
|   |   |   |   |   Speed = slower: upp1 (3.9/3.2)
|   |   |   |   |   Speed = faster: upp2 (4.3/3.4)
|   |   |   AltConfiguration(B) = descending:
|   |   |   |   Manoeuvrability(B) > 0.77 : upp2 (22.4/2.7)
|   |   |   |   Manoeuvrability(B) <= 0.77 :
|   |   |   |   |   Manoeuvrability(A) <= 0.78 : upp2 (8.9/3.8)
|   |   |   |   |   Manoeuvrability(A) > 0.78 : upp1 (10.1/2.3)
|   |   |   AltConfiguration(B) = climbing:
|   |   |   |   Manoeuvrability(B) <= 0.67 :
|   |   |   |   |   Manoeuvrability(B) <= 0.65 :
|   |   |   |   |   |   HorConflConf = diverging: hor1 (1.4/1.3)
|   |   |   |   |   |   HorConflConf = headon: upp1 (0.0)
|   |   |   |   |   |   HorConflConf = converging: upp1 (10.8/7.8)
|   |   |   |   |   |   HorConflConf = crossing: upp2 (2.5/1.4)
|   |   |   |   |   Manoeuvrability(B) > 0.65 :
|   |   |   |   |   |   Manoeuvrability(A) <= 0.84 : dow1 (8.0/2.4)
|   |   |   |   |   |   Manoeuvrability(A) > 0.84 : hor1 (2.4/1.9)
|   |   |   |   Manoeuvrability(B) > 0.67 :
|   |   |   |   |   Manoeuvrability(B) > 0.85 : dow2 (19.3/4.8)
|   |   |   |   |   Manoeuvrability(B) <= 0.85 :
|   |   |   |   |   |   Faster(A) = difficult: hor3 (2.8/1.8)
|   |   |   |   |   |   Faster(A) = possible: dow2 (5.1/1.4)
|   |   |   |   |   |   Faster(A) = easy: hor3 (28.7/4.6)
|   |   CloseToTOD(B) > 10 :
|   |   |   CloseToTOD(B) <= 300 : dow2 (34.0/2.6)
|   |   |   CloseToTOD(B) > 300 : upp2 (4.0/2.2)
|   CloseToTOD(A) > 0 :
|   |   CloseToTOD(B) <= 0 : dow1 (33.3/2.6)
|   |   CloseToTOD(B) > 0 :
|   |   |   CloseToTOD(A) <= 300 :
|   |   |   |   CloseToBoundaries(B) <= 1.8 :
|   |   |   |   |   CloseToTOD(A) <= 234 :
|   |   |   |   |   |   Manoeuvrability(A) <= 0.84 :
```

```
|   |   |   |   |   |   |   Manoeuvrability(B) <= 0.75 :
|   |   |   |   |   |   |   |   LevelsAvailable(A) = withSpaces: dow2 (0.0)
|   |   |   |   |   |   |   |   LevelsAvailable(A) = below: dow1 (3.0/2.5)
|   |   |   |   |   |   |   |   LevelsAvailable(A) = above: dow2 (0.0)
|   |   |   |   |   |   |   |   LevelsAvailable(A) = yes: dow3 (2.5/2.1)
|   |   |   |   |   |   |   |   LevelsAvailable(A) = none:
|   |   |   |   |   |   |   |   |   CloseToTOD(A) <= 63 : spe3 (2.3/1.9)
|   |   |   |   |   |   |   |   |   CloseToTOD(A) > 63 : dow2 (3.1/1.9)
|   |   |   |   |   |   |   Manoeuvrability(B) > 0.75 :[S1]
Subtree 1
|   |   |   |   |   |   Manoeuvrability(A) > 0.84 :
|   |   |   |   |   |   |   LevelsAvailable(A) = withSpaces: dow3 (0.0)
|   |   |   |   |   |   |   LevelsAvailable(A) = below: dow3 (10.0/2.4)
|   |   |   |   |   |   |   LevelsAvailable(A) = above: dow3 (0.0)
|   |   |   |   |   |   |   LevelsAvailable(A) = none: dow2 (9.0/1.3)
|   |   |   |   |   |   |   LevelsAvailable(A) = yes:
|   |   |   |   |   |   |   |   AltitudeNow = same: dow1 (2.6/1.6)
|   |   |   |   |   |   |   |   AltitudeNow = different: dow3 (19.4/1.3)
|   |   |   |   |   CloseToTOD(A) > 234 :
|   |   |   |   |   |   Manoeuvrability(B) > 0 : hor3 (4.0/1.2)
|   |   |   |   |   |   Manoeuvrability(B) <= 0 :
|   |   |   |   |   |   |   EasyToExitHorizontally(A) = difficult: hor1 (0.0)
|   |   |   |   |   |   |   EasyToExitHorizontally(A) = possible: hor1 (0.0)
|   |   |   |   |   |   |   EasyToExitHorizontally(A) = easy: hor1 (3.3/2.4)
|   |   |   |   |   |   |   EasyToExitHorizontally(A) = veryEasy: hor2
(6.7/4.7)
|   |   |   |   CloseToBoundaries(B) > 1.8 :
|   |   |   |   |   CloseToBoundaries(B) <= 2 :
|   |   |   |   |   |   Slower(B) = possible: hor2 (10.0/2.4)
|   |   |   |   |   |   Slower(B) = difficult:
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.88 : hor1 (16.7/3.2)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.88 : hor2 (4.5/1.2)
|   |   |   |   |   |   Slower(B) = easy:
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.76 : hor2 (4.4/1.2)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.76 : hor1 (4.3/1.6)
|   |   |   |   |   CloseToBoundaries(B) > 2 :
|   |   |   |   |   |   CloseToTOD(A) > 181 : upp1 (2.0/1.8)
|   |   |   |   |   |   CloseToTOD(A) <= 181 :
|   |   |   |   |   |   |   CloseToBoundaries(B) <= 8.3 : dow1 (10.0/1.3)
|   |   |   |   |   |   |   CloseToBoundaries(B) > 8.3 : dow3 (3.0/1.1)
|   |   |   CloseToTOD(A) > 300 :
|   |   |   |   CloseToTOD(B) <= 181 : dow2 (10.3/1.7)
|   |   |   |   CloseToTOD(B) > 181 :
|   |   |   |   |   AltConfiguration(B) = descending: upp2 (2.0/1.0)
|   |   |   |   |   AltConfiguration(B) = climbing: dow2 (2.0/1.0)
|   |   |   |   |   AltConfiguration(B) = stable:
|   |   |   |   |   |   Manoeuvrability(B) <= 0.77 : hor3 (14.0/2.5)
|   |   |   |   |   |   Manoeuvrability(B) > 0.77 : spe3 (3.0/2.1)
AltConfiguration(A) = descending:
|   CloseToTOD(B) <= 0 :
|   |   CloseToTOD(A) <= 10 :
|   |   |   LevelsAvailable(B) = withSpaces:
|   |   |   |   Manoeuvrability(B) <= 0 : upp1 (5.1/4.8)
|   |   |   |   Manoeuvrability(B) > 0 :
|   |   |   |   |   Manoeuvrability(A) <= 0.7 :
|   |   |   |   |   |   Manoeuvrability(B) <= 0.71 : dow1 (7.0/1.3)
|   |   |   |   |   |   Manoeuvrability(B) > 0.71 : upp1 (4.2/1.4)
|   |   |   |   |   Manoeuvrability(A) > 0.7 :
|   |   |   |   |   |   EasyToExitHorizontally(A) = difficult: upp2 (3.0/1.1)
|   |   |   |   |   |   EasyToExitHorizontally(A) = possible: upp2 (2.0/1.5)
|   |   |   |   |   |   EasyToExitHorizontally(A) = easy: upp1 (16.7/2.6)
|   |   |   |   |   |   EasyToExitHorizontally(A) = veryEasy: upp1 (18.2/1.5)
|   |   |   LevelsAvailable(B) = below:
|   |   |   |   Manoeuvrability(A) <= 0.72 : upp1 (24.6/6.8)
|   |   |   |   Manoeuvrability(A) > 0.72 :
|   |   |   |   |   AltConfiguration(B) = stable: upp1 (4.3/2.3)
|   |   |   |   |   AltConfiguration(B) = descending: upp1 (6.2/1.6)
|   |   |   |   |   AltConfiguration(B) = climbing: upp3 (4.2/1.3)
|   |   |   LevelsAvailable(B) = above:
|   |   |   |   AltConfiguration(B) = stable: upp1 (1.3/0.9)
|   |   |   |   AltConfiguration(B) = climbing: dow2 (4.7/2.7)
|   |   |   |   AltConfiguration(B) = descending:
|   |   |   |   |   Manoeuvrability(B) <= 0.77 : upp2 (38.7/12.6)
|   |   |   |   |   Manoeuvrability(B) > 0.77 :
|   |   |   |   |   |   EasyToExitHorizontally(B) = difficult: upp1 (2.5/1.1)
```

```
|   |   |   |   |   |   EasyToExitHorizontally(B) = possible: upp1 (12.6/1.5)
|   |   |   |   |   |   EasyToExitHorizontally(B) = easy: upp2 (4.0/2.2)
|   |   |   |   |   |   EasyToExitHorizontally(B) = veryEasy: upp2 (6.6/3.9)
|   |   |   LevelsAvailable(B) = yes:
|   |   |   |   Manoeuvrability(A) <= 0.66 :
|   |   |   |   |   Manoeuvrability(A) <= 0 :
|   |   |   |   |   |   AltConfiguration(B) = stable: upp1 (2.0/1.3)
|   |   |   |   |   |   AltConfiguration(B) = descending: upp2 (20.9/11.6)
|   |   |   |   |   |   AltConfiguration(B) = climbing: upp1 (4.3/3.4)
|   |   |   |   |   Manoeuvrability(A) > 0 :
|   |   |   |   |   |   HorConflConf = diverging: upp2 (11.1/1.3)
|   |   |   |   |   |   HorConflConf = headon: dow2 (10.5/3.9)
|   |   |   |   |   |   HorConflConf = converging: dow2 (4.0/1.2)
|   |   |   |   |   |   HorConflConf = crossing: upp2 (8.1/1.3)
|   |   |   |   Manoeuvrability(A) > 0.66 :
|   |   |   |   |   Manoeuvrability(A) > 0.88 : upp1 (4.4/1.2)
|   |   |   |   |   Manoeuvrability(A) <= 0.88 :
|   |   |   |   |   |   CloseToBoundaries(A) > 4.4 : upp1 (3.8/1.2)
|   |   |   |   |   |   CloseToBoundaries(A) <= 4.4 :
|   |   |   |   |   |   |   AltConfiguration(B) = stable: upp2 (0.0)
|   |   |   |   |   |   |   AltConfiguration(B) = climbing: upp1 (2.4/1.4)
|   |   |   |   |   |   |   AltConfiguration(B) = descending:
|   |   |   |   |   |   |   |   Faster(A) = easy: upp2 (10.0/2.4)
|   |   |   |   |   |   |   |   Faster(A) = difficult:
|   |   |   |   |   |   |   |   |   Slower(B) = difficult: upp2 (6.6/1.9)
|   |   |   |   |   |   |   |   |   Slower(B) = possible: upp1 (0.6/0.6)
|   |   |   |   |   |   |   |   |   Slower(B) = easy:
|   |   |   |   |   |   |   |   |   |   Manoeuvrability(A) <=
0.77:upp1(10.4/3.4)
|   |   |   |   |   |   |   |   |   |   Manoeuvrability(A) > 0.77 :upp2
(2.7/1.7)
|   |   |   |   |   |   |   |   |   Faster(A) = possible:
|   |   |   |   |   |   |   |   |   |   Manoeuvrability(B) <= 0.67 : upp1 (3.0/2.1)
|   |   |   |   |   |   |   |   |   |   Manoeuvrability(B) > 0.67 : upp2 (29.7/2.4)
|   |   |   LevelsAvailable(B) = none:
|   |   |   |   AltConfiguration(B) = stable: upp1 (4.9/1.4)
|   |   |   |   AltConfiguration(B) = descending: upp1 (26.7/4.3)
|   |   |   |   AltConfiguration(B) = climbing: hor3 (7.6/3.1)
|   |   CloseToTOD(A) > 10 :
|   |   |   Manoeuvrability(B) <= 0 : upp1 (2.0/1.0)
|   |   |   Manoeuvrability(B) > 0 :
|   |   |   |   Manoeuvrability(B) <= 0.88 : dow1 (18.2/1.3)
|   |   |   |   Manoeuvrability(B) > 0.88 : dow2 (6.0/2.3)
|   CloseToTOD(B) > 0 :
|   |   CloseToTOD(B) <= 69 :
|   |   |   Manoeuvrability(B) > 0.78 : dow3 (10.0/1.3)
|   |   |   Manoeuvrability(B) <= 0.78 :
|   |   |   |   Manoeuvrability(A) <= 0 : dow3 (3.8/3.2)
|   |   |   |   Manoeuvrability(A) > 0 : dow2 (2.0/1.0)
|   |   CloseToTOD(B) > 69 :
|   |   |   CloseToTOD(B) <= 88 : dow2 (25.0/2.5)
|   |   |   CloseToTOD(B) > 88 :
|   |   |   |   Manoeuvrability(A) > 0.76 : upp1 (9.0/1.3)
|   |   |   |   Manoeuvrability(A) <= 0.76 :
|   |   |   |   |   CloseToTOD(B) > 155 : upp1 (3.8/2.9)
|   |   |   |   |   CloseToTOD(B) <= 155 :
|   |   |   |   |   |   CloseToTOD(B) <= 97 : upp3 (7.0/2.4)
|   |   |   |   |   |   CloseToTOD(B) > 97 : dow2 (7.2/2.3)
AltConfiguration(A) = climbing:
|   Manoeuvrability(A) <= 0 :
|   |   CloseToTOD(A) <= 234 : dow1 (67.1/51.8)
|   |   CloseToTOD(A) > 234 :
|   |   |   CloseToTOD(A) > 352 : upp1 (3.5/2.5)
|   |   |   CloseToTOD(A) <= 352 :
|   |   |   |   EasyToExitHorizontally(B) = difficult: hor1 (0.0)
|   |   |   |   EasyToExitHorizontally(B) = possible: hor1 (0.0)
|   |   |   |   EasyToExitHorizontally(B) = easy: hor2 (3.3/2.4)
|   |   |   |   EasyToExitHorizontally(B) = veryEasy: hor1 (6.7/4.7)
|   Manoeuvrability(A) > 0 :
|   |   LevelsAvailable(B) = none: dow1 (57.0/9.0)
|   |   LevelsAvailable(B) = withSpaces:
|   |   |   HorConflConf = diverging: dow1 (0.4/0.4)
|   |   |   HorConflConf = headon: hor1 (1.7/1.4)
|   |   |   HorConflConf = converging: hor3 (24.2/3.5)
|   |   |   HorConflConf = crossing: dow1 (19.1/2.4)
```

```
|   |   LevelsAvailable(B) = below:
|   |   |   CloseToTOD(A) <= 80 :
|   |   |   |   Manoeuvrability(A) <= 0.65 : dow2 (13.2/1.5)
|   |   |   |   Manoeuvrability(A) > 0.65 :
|   |   |   |   |   Manoeuvrability(B) <= 0.7 :
|   |   |   |   |   |   Manoeuvrability(B) <= 0.62 : dow2 (2.2/1.2)
|   |   |   |   |   |   Manoeuvrability(B) > 0.62 : dow1 (24.5/4.6)
|   |   |   |   |   Manoeuvrability(B) > 0.7 :
|   |   |   |   |   |   LevelsAvailable(A) = withSpaces: dow2 (3.2/1.2)
|   |   |   |   |   |   LevelsAvailable(A) = above: dow2 (0.0)
|   |   |   |   |   |   LevelsAvailable(A) = none: dow2 (11.7/1.9)
|   |   |   |   |   |   LevelsAvailable(A) = below:
|   |   |   |   |   |   |   AltConfiguration(B) = stable: dow1 (2.1/1.5)
|   |   |   |   |   |   |   AltConfiguration(B) = descending: dow1 (0.3/0.3)
|   |   |   |   |   |   |   AltConfiguration(B) = climbing:[S2]  ← Subtree 2
|   |   |   |   |   |   LevelsAvailable(A) = yes:
|   |   |   |   |   |   |   Manoeuvrability(B) <= 0.77 : dow1 (8.2/2.6)
|   |   |   |   |   |   |   Manoeuvrability(B) > 0.77 :
|   |   |   |   |   |   |   |   HorConflConf = headon: dow1 (7.0/1.3)
|   |   |   |   |   |   |   |   HorConflConf = converging: dow2 (1.7/1.4)
|   |   |   |   |   |   |   |   HorConflConf = crossing: dow2 (9.0/1.3)
|   |   |   |   |   |   |   |   HorConflConf = diverging:
|   |   |   |   |   |   |   |   |   Manoeuvrability(A) <= 0.81 : dow1 (3.0/1.1)
|   |   |   |   |   |   |   |   |   Manoeuvrability(A) > 0.81 : dow2 (6.0/1.2)
|   |   |   CloseToTOD(A) > 80 :
|   |   |   |   Manoeuvrability(A) <= 0.83 : upp3 (3.0/2.1)
|   |   |   |   Manoeuvrability(A) > 0.83 : dow3 (3.0/1.1)
|   |   LevelsAvailable(B) = above:
|   |   |   CloseToTOD(A) > 33 : dow1 (16.3/1.3)
|   |   |   CloseToTOD(A) <= 33 :
|   |   |   |   Manoeuvrability(B) <= 0.64 : hor3 (5.5/1.3)
|   |   |   |   Manoeuvrability(B) > 0.64 :
|   |   |   |   |   Manoeuvrability(B) <= 0.72 : upp2 (3.7/1.9)
|   |   |   |   |   Manoeuvrability(B) > 0.72 : dow1 (9.3/2.4)
|   |   LevelsAvailable(B) = yes:
|   |   |   CloseToTOD(A) <= 88 :
|   |   |   |   AltConfiguration(B) = stable:
|   |   |   |   |   HorConflConf = diverging: dow1 (5.9/2.8)
|   |   |   |   |   HorConflConf = headon: hor2 (2.2/1.6)
|   |   |   |   |   HorConflConf = crossing: dow2 (5.6/1.3)
|   |   |   |   |   HorConflConf = converging:
|   |   |   |   |   |   Faster(B) = difficult: hor3 (0.0)
|   |   |   |   |   |   Faster(B) = possible: dow1 (3.3/1.2)
|   |   |   |   |   |   Faster(B) = easy: hor3 (5.2/1.7)
|   |   |   |   AltConfiguration(B) = descending:
|   |   |   |   |   Manoeuvrability(A) <= 0.65 : dow2 (6.9/1.3)
|   |   |   |   |   Manoeuvrability(A) > 0.65 :
|   |   |   |   |   |   LevelsAvailable(A) = withSpaces: dow1 (0.0)
|   |   |   |   |   |   LevelsAvailable(A) = above: dow1 (2.3/1.9)
|   |   |   |   |   |   LevelsAvailable(A) = none: hor3 (6.1/1.4)
|   |   |   |   |   |   LevelsAvailable(A) = below:
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.7 : upp3 (4.5/1.6)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.7 : upp2 (3.1/1.2)
|   |   |   |   |   |   LevelsAvailable(A) = yes:
|   |   |   |   |   |   |   Manoeuvrability(A) <= 0.85 : upp2 (4.6/2.8)
|   |   |   |   |   |   |   Manoeuvrability(A) > 0.85 : dow1 (10.2/1.3)
|   |   |   |   AltConfiguration(B) = climbing:
|   |   |   |   |   EasyToExitHorizontally(B) = difficult: dow1 (3.3/1.4)
|   |   |   |   |   EasyToExitHorizontally(B) = veryEasy: dow2 (67.6/12.7)
|   |   |   |   |   EasyToExitHorizontally(B) = possible:
|   |   |   |   |   |   EasyToExitHorizontally(A) = difficult: dow1 (0.0)
|   |   |   |   |   |   EasyToExitHorizontally(A) = possible: dow1 (0.0)
|   |   |   |   |   |   EasyToExitHorizontally(A) = easy: dow2 (4.4/2.4)
|   |   |   |   |   |   EasyToExitHorizontally(A) = veryEasy: dow1 (7.7/1.8)
|   |   |   |   |   EasyToExitHorizontally(B) = easy:
|   |   |   |   |   |   Manoeuvrability(A) <= 0.63 : dow1 (4.0/1.2)
|   |   |   |   |   |   Manoeuvrability(A) > 0.63 :
|   |   |   |   |   |   |   Manoeuvrability(B) <= 0.7 : dow1 (6.0/3.4)
|   |   |   |   |   |   |   Manoeuvrability(B) > 0.7 :
|   |   |   |   |   |   |   |   Manoeuvrability(B) <= 0.78 : dow2 (11.6/1.4)
|   |   |   |   |   |   |   |   Manoeuvrability(B) > 0.78 :
|   |   |   |   |   |   |   |   |   Manoeuvrability(B) <= 0.85 : dow1 (2.1/1.1)
|   |   |   |   |   |   |   |   |   Manoeuvrability(B) > 0.85 : dow2 (5.1/1.2)
|   |   |   CloseToTOD(A) > 88 :
|   |   |   |   AltConfiguration(B) = stable: dow2 (2.0/1.8)
```

```
|   |   |   |   AltConfiguration(B) = descending: upp3 (5.0/2.3)
|   |   |   |   AltConfiguration(B) = climbing: dow3 (6.0/1.2)



Subtree [S1]
EasyToExitHorizontally(B) = difficult: upp2 (0.0)
EasyToExitHorizontally(B) = possible: upp2 (0.0)
EasyToExitHorizontally(B) = easy: upp2 (9.0/2.4)
EasyToExitHorizontally(B) = veryEasy: dow1 (3.0/1.1)


Subtree [S2]
EasyToExitHorizontally(A) = difficult: dow2 (0.0)
EasyToExitHorizontally(A) = possible: dow1 (3.0/1.1)
EasyToExitHorizontally(A) = easy: dow2 (4.0/1.2)
EasyToExitHorizontally(A) = veryEasy:
|   Priority = same: dow1 (0.5/0.5)
|   Priority = lower: dow1 (2.1/1.1)
|   Priority = higher: dow2 (3.1/1.2)

Evaluation on training data (1408 items):

        Before Pruning           After Pruning
       ----------------   --------------------------
       Size      Errors   Size      Errors   Estimate

        523  141(10.0%)    297  196(13.9%)    (30.7%)   <<
```

Where the subtrees S1 and S2 can be found in the tree and have been reported separately for simplicity.

B.2 The Discriminatory Power in ISAC and C4.5

As said in Chapter 5, the algorithm used by ISAC for the calculation of the discriminatory power is slightly different from the one used in C4.5. The discriminatory power for the parameters in the latest case-base, as calculated by ISAC, is shown in Table B.1. The smaller the value of remainder, the more discriminatory is the parameter.

By comparing the decision tree above with Table B.1, it can be seen that the parameter “CloseToTOD(B)” is the most discriminatory for ISAC, whereas in C4.5 the most discriminatory parameter, which is the root of the decision tree, is “AltConfiguration(A)”. The reason of this discrepancy is because ISAC and C4.5 use slightly different algorithms for the calculation of the information, as explained in Chapter 5.

As said in Chapter 4, the parameter “Similar” has not been used in the latest steps of the knowledge engineering process because derived from other parameters already present in the case description. This decision is supported by the fact that the discriminatory power of this parameter is the lowest, see Table B.1.

B.3 Conclusion

In this appendix the utility of the discriminatory power of the weights is shown from two points of view: on one hand, the discriminatory power is used to build a decision tree in which the information contained in the case-base is stored. On the other hand, the discriminatory power is simply used to build a list of the most important parameters. Some discrepancies between the results of the two methods are shown here and are explained in Chapter 5.

Table B.1: *Discriminatory power from ISAC.*

```
Building Decision Tree.....

Parameter                       Remainder          Type of parameter
^^^^^^^^^                       ^^^^^^^^^          ^^^^^^^^^^^^^^^^^
CloseToTOD(B)                         1.9786             Numeric
CloseToTOD(A)                         1.9786             Numeric
CloseToBoundaries(B)            2.0209             Numeric
CloseToBoundaries(A)            2.0209             Numeric
Manoeuvrability(B)              2.0341             Numeric
Manoeuvrability(A)              2.0341             Numeric
AltConfiguration(B)             2.2236             Symbolic
AltConfiguration(A)             2.2236             Symbolic
HorConflConf                    2.7725             Symbolic
AltitudeNow                     2.7905             Symbolic
EasyToExitHorizontally(B)       3.1536             Symbolic
EasyToExitHorizontally(A)       3.1536             Symbolic
LevelsAvailable(B)              3.6363             Symbolic
LevelsAvailable(A)              3.6363             Symbolic
Priority                        3.6513             Symbolic
Slower(B)                       4.0482             Symbolic
Slower(A)                       4.0482             Symbolic
Faster(B)                       4.0510             Symbolic
Faster(A)                       4.0510             Symbolic
Speed                           4.0624             Symbolic
Similar                         4.1721             Symbolic
```

Appendix C

Classes and Functions in ISAC

The file `header1.h` contains the definition of the classes and functions that constitute the core of ISAC, i.e. the functions that are used at run-time, such as the retrieval function. On the other hand, the file `header2.h` contains the definition of all the functions that have been used during the knowledge engineering process to refine the case description (e.g., the function that calculates the discriminatory power) and that are not necessary at run-time. The use of global variables and the hard coding of some file names is not considered "clean" programming but has been accepted in this prototypical version due to time restrictions.

C.1 The File `header1.h`

```
#define FileCaseBase "/dd/csc/abonzano/ISAC/CaseBase"
#define FileCaseStruct "/dd/csc/abonzano/ISAC/CaseStruct"
#define FileTarget "/dd/csc/abonzano/ISAC/target"
#define SolFile "/dd/csc/abonzano/ISAC/Solutions"
#define ResultsFile "/dd/csc/abonzano/ISAC/results"

//CASE STRUCT (bodies are in ReadCaseStruct.c)
//===========
class MiniCell
 {
  char name[64];
 public:
  MiniCell* next;
  MiniCell(MiniCell*,char*);
  char* GiveName() {return name;}
 };

class TypeNode
 {
  char name[32];
  int number,constraint,NumOfValues,sel;
  double remainder,min,max;
  double globalWeight,globalNewWeight,globalHiPerfWeight;
  MiniCell* PossValues;
 public:
  TypeNode* next;
  TypeNode (TypeNode*,int,int,int,char*,MiniCell*);
  char* GiveName() {return name;}
  MiniCell* GivePossValues() {return PossValues;}
  int GiveNumOfValues() {return NumOfValues;}
  int GiveConstraint() {return constraint;}
  int GiveNumber() {return number;}
  double GiveRemainder() {return remainder;}
  double GiveMin() {return min;}
  double GiveMax() {return max;}
  void PutRemainder(double num) {remainder=num;}
  void StoreMin(double val) {min=val;}
  void StoreMax(double val) {max=val;}
  void PrintForC45(char*);
  void StoreSel(int val) {sel=val;}
```

```
  int GiveSel() {return sel;}
  double GiveWeight() {return globalWeight;}
  double GiveNewWeight() {return globalNewWeight;}
  void ChangeWeight(double);
  void ChangeHighest(double);
  void SwapWeights() {globalWeight=globalNewWeight;}
  void SwapHighest() {globalWeight=globalHiPerfWeight;}
 };

//CASE-BASE (bodies in ReadCaseBase.c)
//=========
class branch;
class OneCase;

class OneFeat
 {
  char FeatName[32];
  double NumValue;
  double weight,NewWeight,HighestPerfWeight;
  char SymValue[32];
 public:
  OneFeat* next;
  OneFeat(char*,OneFeat*,int);
  char* GiveName() {return FeatName;}
  char* GiveSymValue() {return SymValue;}
  double GiveNumValue() {return NumValue;}
  void PutFeatValue(char* ReadValue) {strcpy(SymValue,ReadValue);}
  void PutFeatValue(double val) {NumValue=val;}
  double GiveWeight() {return weight;}
  double GiveNewWeight() {return NewWeight;}
  void ChangeWeight(double);
  void ChangeHighest(double);
  void SwapWeights() {weight=NewWeight;}
  void SwapHighest() {weight=HighestPerfWeight;}
 };

class OneCase
 {
  char CaseName[32];
  char Solution[80];
  int NumNIL;
  double Activation;
  OneFeat* FeatList;
 public:
  double Kc,Fc;
  int ThisCaseIsUsed;
  OneCase* next;
  OneCase(char*,OneCase*,TypeNode*);
  void StoreFeatValue(char*,char*,double);
  void StoreFeatValue(char*,double,double);
  int GiveNumNIL() {return NumNIL;}
  double GiveNumValue(char*);
  char* GiveSymValue(char*);
  char* GiveName() {return CaseName;}
  void StoreSol(char*);
  char* GiveFirstSol();
  char* GiveSol() {return Solution;}
  void ResetAct() {Activation=0.0;}
  void AddAct(double);
  double GiveAct() {return Activation;}
  OneFeat* GiveFeats() {return FeatList;}
  double GiveWeight(char* FeatName);
  double GiveNewWeight(char* FeatName);
  void ChangeWeight(char*,double);
 };

//TREE FOR BASE FILTERING (Bodies in Tree.c)
//=======================
class SimCase
 {
  OneCase *ACase;
 public:
  SimCase *next;
  SimCase(OneCase*,SimCase*);
  OneCase* GiveCase() {return ACase;}
```

```
  double GiveAct() {return ACase->GiveAct();}
  int GiveNumNIL() {return ACase->GiveNumNIL();}
  char* GiveName() {return ACase->GiveName();}
  char* GiveSol() {return ACase->GiveSol();}
  char* GiveFirstSol() {return ACase->GiveFirstSol();}
 };

class branch
 {
  char FeatValue[64];
  SimCase* ListOfCases;
 public:
  branch* next;
  branch(branch*,char*,char*);
  char* GiveName() {return FeatValue;}
  void AddACase(OneCase* OCase)
                    {ListOfCases= new SimCase(OCase,ListOfCases);}
  SimCase* GiveList() {return ListOfCases;}
 };

//FOR THE FINAL SOLUTION
//======================
class SAN
 {
  char name[32];
  char sol[32];
  double act;
 public:
  SAN *next;
  SAN(char*,char*,double,SAN*);
  char* GiveName() {return name;}
  char* GiveSol() {return sol;}
  double GiveAct() {return act;}
 };

class solsType
 {
 public:
  solsType(char*,int,solsType*);
  char name[12];
  int val;
  solsType *next;
 };

//GLOBAL VARIABLES
//================
extern int GUM;
extern int GDU;
extern int BUU;
extern int BDM;
extern int randomWeight;
extern int representation;
extern int MaxIterations;
extern int NumForTrainingSet;
extern int options;
extern int shift;
extern int multipl;
extern int average;
extern int global;
extern int DoAlsoGlobal;
extern int updateWeights;
extern int DoGraphic;
extern double MaxActivation;
extern char TypeOfSimulation[12];

//FUNCTIONS
//=========
int ThereAreNoConstraints(TypeNode*);
void Shuffle(char*,TypeNode*);

void wait();
void ATCBR(void);
char* DelEOL(char*);
char* Read(FILE*);
char** ReadSol(int*);
```

```
void CreateCopy(char*);

TypeNode* ReadCaseStruct(char*);
void CheckMinMax(char*,double,TypeNode*);
MiniCell* ReadValues(int,FILE*);
int ItIsANumber(TypeNode*,char*);
int ItIsAConstraint(char*,TypeNode*);
void AddCaseToBranch(char*,char*,OneCase*,branch*);
branch* BuildWebOfPointers(TypeNode*,OneCase*);

OneFeat* BuildEmptyFeatList(TypeNode*);
OneCase* ReadCaseBase(char*,TypeNode*);
OneCase* ReadOneCase(char*,OneCase*,TypeNode*,FILE*);
OneCase* ReadAllTargets(char*,TypeNode*);
OneCase* ReadOneTarget(char*,OneCase*,TypeNode*,FILE*);
void FindCases(OneCase*,OneCase*,branch*,TypeNode*,char*);
double FindMaxAct(SimCase*);

SimCase* BaseFiltering(OneCase*,branch*,TypeNode*);
SimCase* CutSubList(SimCase*,SimCase*);
int ItIsIn(SimCase*,SimCase*);

void ResetActivation(OneCase*);
void ResetCaseBase(OneCase*);
void SpreadingActivation(SimCase*,TypeNode*,OneCase*,branch*);
void GlobalSpreadingActivation(SimCase*,TypeNode*,OneCase*,branch*);
void CalcSymAct(TypeNode*,OneCase*,branch*);
void CalcNumAct(TypeNode*,OneCase*,SimCase*);

SAN* Analyse(SimCase*,char*,double,SAN*);
char* ChooseFinal(SAN*);
char* FilterSol(char*,char*);

//SHOW
//=====
void ShowCaseStruct(TypeNode*);
void ShowCaseBase(OneCase*,TypeNode*);
void ShowBranches(branch*);
void ShowTarget(OneCase*,TypeNode*);
void WriteCaseBase(OneCase*,TypeNode*);

//DELETE
//=======
MiniCell* Delete(MiniCell*);
TypeNode* Delete(TypeNode*);
OneCase* Delete(OneCase*);
SimCase* Delete(SimCase*);
OneFeat* Delete(OneFeat*);
branch* Delete(branch*);
SAN* Delete(SAN*);
solsType* Delete(solsType*);
```

C.2 The File `header2.h`

```
#define BigFile "/dd/csc/abonzano/ISAC/BigCB"
#define InputFile "/dd/csc/abonzano/ISAC/simul"
#define InputFile2 "/dd/csc/abonzano/ISAC/simul2"
#define ForC45 "/dd/csc/abonzano/ISAC/forC45"
#define forGraph "/dd/csc/abonzano/ISAC/.numbers"
#define FileRealSet "/dd/csc/abonzano/ISAC/RealSet"

//NUMERIC REMAINDER (in NumericRemainder.c)
//======================
class cell
 {
  double val;
 public:
  cell *next;
  cell(double,cell*);
  void NewNext(cell* NEW) {next=NEW;}
  double GiveVal() {return val;}
 };
//LOCAL WEIGHTS
//=============
```

```
extern int NumCasesRetrieved;
double Evaluate();
void ToyCaseBase();
void EliminateNIL(OneCase*,TypeNode*);
int CaseIsCorrect(char*,char*);
int StringsAreCompatible(char*,char*);
void PrepFilesForPiv(char*,TypeNode*);
void Training_Test(char*,TypeNode*);
void IntrospectiveTest(TypeNode*);
void TestForPivotals(TypeNode*);
void Histogram(TypeNode*,OneCase*);
void GlobalHistogram(TypeNode*);
double Testing(OneCase*,OneCase*,branch*,TypeNode*);
void NormalizeMaxActivation(OneCase*,TypeNode*);
void NormalizeGlobalMaxActivation(TypeNode*);
void LocalWeightsSum(SimCase*,OneCase*,TypeNode*);
void GlobalWeights(SimCase*,OneCase*,TypeNode*);
void LocalWeightsMul(SimCase*,OneCase*,TypeNode*);
void UpdateWeights(OneCase*,int);
void UpdateGlobalWeights(TypeNode*,int);
void UpdateHighest(OneCase*);
void UpdateGlobalHighest(TypeNode*);
void CalcAverageFromLocal(TypeNode*,OneCase*);

//SIMULATIONS
//============
void ToFile(OneCase*,FILE*,TypeNode*);
void LeaveOneOut(TypeNode*,OneCase*,branch*);
void LeaveOneIn(TypeNode*,char*);
void MakeSymmetric();
int NumOfCases(OneCase*);

//OLD METHOD (FLAT SEARCH)
//==========
void LeaveOneInO(TypeNode*,char*);
branch* BuildTreeO(TypeNode*,OneCase*);
void FindCasesO(OneCase*,OneCase*,branch*,TypeNode*,char*);
SimCase* SpreadingActivationO(SimCase*,TypeNode*,OneCase*);
void CalcSymActO(char*,char*,SimCase*);
void CalcNumActO(double,double,SimCase*,double,double);

//C4.5
//====
void C45_IN(TypeNode*);
void C45_OUT(TypeNode*);
void C45DataNames(TypeNode*,OneCase*);
void C45TestSol(TypeNode*,OneCase*,char*);
void ReadResults(char);
char* CheckNIL(char*);

//DECISION TREE
//=============
TypeNode* BuildDecTree(TypeNode*,OneCase*);
double Remainder(int,int,int*,int**);
double NumRem(OneCase*,TypeNode*,char**,int*,int);
double Info(int,int,int**);
double Weight(int,int,int*,int**);
TypeNode* OrderSList(TypeNode*);
cell* ReadAllNumbers(OneCase*,char*);
double FromOne(TypeNode*,OneCase*,double,char**,int*,int);

//BIG CASE-BASE
//==============
void BigCaseBase(TypeNode*);

//COVERAGE
//========
void FindNumbers(TypeNode*);
void SMA(TypeNode*);
void AVE(TypeNode*);
double NumCases(TypeNode*,int);
```

Appendix D

The Data Files

This appendix shows the files that contain the knowledge base data which is read by ISAC at the start up. The files `CaseBase`, `Solutions` and `CaseStruct` contain respectively the case-base, the possible solutions for a case and the structure for the case. In the data files, lines that begin with the symbol "`//`" are comment lines and are automatically skipped by ISAC.

D.1 The file `CaseStruct`

Each parameter has the following fields that must be arranged in order on the same line:

- the name of the parameter,
- an integer that indicates whether the parameter has numeric (1) or symbolic values (0),
- an integer that indicates whether the parameter is a constraint (1) or not (0),
- a real number that indicates the weight of the parameter,
- if the parameter has symbolic values, the number of possible values.

Then, if the parameter has symbolic values, these values are listed one after the other, each on a new line. The field reserved for the weight is used if the human expert has an idea of the importance of each parameter in relation to all the others. Usually this is difficult to decide upon and consequently the weight of the parameters is determined with introspective learning techniques. If this happen, the weight values read from the file are discarded.

The case structure reported below is that for the "TwoInOne.canonical" and for the "TwoInOne.nonCanonical" case representations.

```
// NO TABS ALLOWED, COMMENTS MUST BE AT THE BEGINNING OF THE LINE
// ALL POSSIBLE VALUES MUST BE ON A <<NEW>> LINE
// Information about the case structure
// Name-Of-The-Parameter Is-It-A-Number? Is-It-A-Constraint?
// Weight Num-Of-Possible-Values Values

                                        HorConflConf 0 0 1 4
                                        //----------
                                        crossing
                                        converging
```

```
headon
diverging

Priority 0 0 1 3
//------
higher
lower
same

Similar 0 0 1 2
//-----
yes
no

AltitudeNow 0 0 1 2
//---------
different
same

Speed 0 0 1 3
//---
faster
slower
same

AltConfiguration(A) 0 0 1 3
//-------------
climbing
descending
stable

CloseToTOD(A) 1 0 1
//-----------

CloseToBoundaries(A) 1 0 1
//------------------

Manoeuvrability(A) 1 0 1
//----------------

EasyToExitHorizontally(A) 0 0 1 4
//----------------
veryEasy
easy
possible
difficult

LevelsAvailable(A) 0 0 1 5
//----------------
yes
none
above
below
withSpaces

Faster(A) 0 0 1 3
//-------
easy
possible
difficult

Slower(A) 0 0 1 3
//-------
easy
possible
difficult

AltConfiguration(B) 0 0 1 3
//-----------------
climbing
descending
stable

CloseToTOD(B) 1 0 1
//-----------

CloseToBoundaries(B) 1 0 1
//------------------

Manoeuvrability(B) 1 0 1
//----------------

EasyToExitHorizontally(B) 0 0 1 4
//----------------
veryEasy
easy
possible
difficult

LevelsAvailable(B) 0 0 1 5
//----------------
none
yes
above
below
withSpaces

Faster(B) 0 0 1 3
//-------
easy
possible
difficult

Slower(B) 0 0 1 3
//-------
easy
possible
difficult
```

The case structure reported below is that for the “OneInOne” case representation.

```
// NO TABS ALLOWED, COMMENTS MUST BE AT THE BEGINNING OF THE LINE
// ALL POSSIBLE VALUES MUST BE ON A <<NEW>> LINE
// Information about the case structure
// Name-Of-The-Parameter Is-It-A-Number? Is-It-A-Constraint?
// Weight Num-Of-Possible-Values Values

AltConfiguration 0 0 1 3
//--------------
climbing
descending
```

```
stable

CloseToTOD 1 0 1
//--------

CloseToBoundaries 1 0 1
//---------------

AltitudeNow 0 0 1 2
//---------
different
same

Speed 0 0 1 3
//---
faster
slower
same

Manoeuvrability 1 0 1
//-------------

Priority 0 0 1 3
//------
higher
lower
same




HorConflConf 0 0 1 4
//----------
crossing
converging
headon
diverging

EasyToExitHorizontally 0 0 1 4
//--------------------
veryEasy
easy
possible
difficult

LevelsAvailable 0 0 1 5
//-------------
yes
none
above
below
withSpaces


Faster 0 0 1 3
//------------
easy
possible
difficult
Slower 0 0 1 3
//------------
easy
possible
difficult
```

D.2 The file `Solutions`

The first non-commented line of this file must contain the number of possible solutions. These are then listed, as before, each one on a new line. The solutions should not be longer than 32 characters. This file is the same for all the case representations.

```
// There must be the number of possible solutions
// and the names (not longer than 32 char)
12
upp1
dow1
upp2
dow2
upp3
dow3
spe1
spe2
spe3
hor1
hor2
hor3
```

D.3 The file `CaseBase`

Though the full case-base in the “TwoInOne.nonCanonical” case representation contains around 1400 cases, only 20 are reported in this thesis. Because of the case representation, all cases are repeated twice with the order of the aircraft swapped as shown below. For example, `Case697_1` and `Case697_2` describe the same conflict. The case-base that

uses the “TwoInOne.canonical” case representation contains only half of the cases, i.e. around 700, one for each conflict, expressed in the canonical form.

```
@n Case697_1
HorConflConf diverging
Priority same
AltitudeNow same
Speed slower
AltConfiguration(A) stable
CloseToTOD(A) 147
CloseToBoundaries(A) 4.7
Manoeuvrability(A) .71
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) yes
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 3112
CloseToBoundaries(B) 2.7
Manoeuvrability(B) .83
EasyToExitHorizontally(B) possible
LevelsAvailable(B) below
Faster(B) difficult
Slower(B) difficult
@s dow1
```

```
@n Case697_2
HorConflConf diverging
       Priority same
AltitudeNow same
Speed faster
AltConfiguration(A) stable
CloseToTOD(A) 3112
CloseToBoundaries(A) 2.7
Manoeuvrability(A) .83
EasyToExitHorizontally(A) possible
LevelsAvailable(A) below
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 147
CloseToBoundaries(B) 4.7
Manoeuvrability(B) .71
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) yes
Faster(B) difficult
Slower(B) difficult
@s dow2
```

```
@n Case698_1
HorConflConf crossing
Priority same
AltitudeNow same
Speed slower
AltConfiguration(A) stable
CloseToTOD(A) 80
CloseToBoundaries(A) 2.7
Manoeuvrability(A) .7
EasyToExitHorizontally(A) possible
LevelsAvailable(A) yes
Faster(A) easy
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 2973
CloseToBoundaries(B) 4.3
Manoeuvrability(B) .83
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) yes
Faster(B) difficult
Slower(B) difficult
@s dow1
```

```
@n Case698_2
HorConflConf crossing
       Priority same
AltitudeNow same
Speed faster
AltConfiguration(A) stable
CloseToTOD(A) 2973
CloseToBoundaries(A) 4.3
Manoeuvrability(A) .83
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) yes
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 80
CloseToBoundaries(B) 2.7
Manoeuvrability(B) .7
EasyToExitHorizontally(B) possible
LevelsAvailable(B) yes
Faster(B) easy
Slower(B) difficult
@s dow2
```

```
@n Case699_1
HorConflConf converging
Priority same
AltitudeNow different
Speed same
AltConfiguration(A) stable
CloseToTOD(A) 317
CloseToBoundaries(A) 2
Manoeuvrability(A) .85
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) above
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) climbing
CloseToTOD(B) 417
CloseToBoundaries(B) 2.8
Manoeuvrability(B) .85
EasyToExitHorizontally(B) easy
LevelsAvailable(B) yes
Faster(B) difficult
Slower(B) possible
@s dow2
```

```
@n Case699_2
HorConflConf converging
       Priority same
AltitudeNow different
Speed same
AltConfiguration(A) climbing
CloseToTOD(A) 417
CloseToBoundaries(A) 2.8
Manoeuvrability(A) .85
EasyToExitHorizontally(A) easy
LevelsAvailable(A) yes
Faster(A) difficult
Slower(A) possible
AltConfiguration(B) stable
CloseToTOD(B) 317
CloseToBoundaries(B) 2
Manoeuvrability(B) .85
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) above
Faster(B) difficult
Slower(B) difficult
@s dow1
```

```
@n Case700_1
HorConflConf converging
Priority same
AltitudeNow different
Speed faster
AltConfiguration(A) stable
CloseToTOD(A) 417
CloseToBoundaries(A) 3.9
Manoeuvrability(A) .83
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) above
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) climbing
CloseToTOD(B) 1323
CloseToBoundaries(B) 3.3
Manoeuvrability(B) .71
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) yes
Faster(B) difficult
Slower(B) difficult
@s dow2&hor3
```

```
@n Case700_2
HorConflConf converging
      Priority same
AltitudeNow different
Speed slower
AltConfiguration(A) climbing
CloseToTOD(A) 1323
CloseToBoundaries(A) 3.3
Manoeuvrability(A) .71
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) yes
Faster(A) difficult
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 417
CloseToBoundaries(B) 3.9
Manoeuvrability(B) .83
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) above
Faster(B) difficult
Slower(B) difficult
      @s dow1&hor3
```

```
@n Case701_1
HorConflConf crossing
Priority same
AltitudeNow same
Speed faster
AltConfiguration(A) stable
CloseToTOD(A) 112
CloseToBoundaries(A) 6.1
Manoeuvrability(A) .8
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) above
Faster(A) difficult
Slower(A) possible
AltConfiguration(B) stable
CloseToTOD(B) 210
CloseToBoundaries(B) 2.9
Manoeuvrability(B) .75
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) yes
Faster(B) easy
Slower(B) difficult
@s dow1&hor3&spe3
```

```
@n Case701_2
HorConflConf crossing
      Priority same
AltitudeNow same
Speed slower
AltConfiguration(A) stable
CloseToTOD(A) 210
CloseToBoundaries(A) 2.9
Manoeuvrability(A) .75
EasyToExitHorizontally(A) veryEasy
LevelsAvailable(A) yes
Faster(A) easy
Slower(A) difficult
AltConfiguration(B) stable
CloseToTOD(B) 112
CloseToBoundaries(B) 6.1
Manoeuvrability(B) .8
EasyToExitHorizontally(B) veryEasy
LevelsAvailable(B) yes
Faster(B) difficult
Slower(B) possible
@s dow2&hor3&spe3
```

The conflict named “Case697”, reported above in the “TwoInOne” case representation, is reported here in the “OneInOne” case representation. The conflict has been split into two separate cases, one for each aircraft involved in the conflict.

```
@n Case697(A)
HorConflConf diverging
AltitudeNow same
AltConfiguration stable
Speed slower
CloseToTOD 147
CloseToBoundaries 4.7
Manoeuvrability .71
Priority same
EasyToExitHorizontally veryEasy
LevelsAvailable yes
Faster difficult
Slower difficult
@s dow1
```

```
      @n Case697(B)
HorConflConf diverging
AltitudeNow same
AltConfiguration stable
Speed faster
CloseToTOD 3112
CloseToBoundaries 2.7
Manoeuvrability .83
      Priority same
EasyToExitHorizontally possible
LevelsAvailable below
Faster difficult
Slower difficult
@s dow2
```

Appendix E

The Code

For space restrictions it is not possible to show all the C files that compose ISAC and the interface with HIPS and ISAC. For this reason, the names of all the programs are listed but only the code of the most significant files is reported.

E.1 From ISAC

Files in the directory `ISAC`

```
BaseFiltering.C
BigCaseBase.C
C45.C
CaseBase
CaseStruct
DecisionTree.C
Filter.C
FindCases.C
FindSol.C
GlobalWeights.C
header2.h
ISAC
IntrospectiveLearning.C
Main.C
NumericRemainder.C
Old.C
Old2.C
Old3.C
Pivotal.C
ReadCaseBase.C
ReadCaseStruct.C
ReadTarget.C
RealSet
Show.C
Shuffle.C
Simulations.C
SolForMAC
Solutions
SpreadingActivation.C
Suggestion
TreeEqualWebOfPointer.C
WebOfPointers.C
Weights.C
discrim
header1.h
makefile
target
ISAC/R5/Src: the source files for the
C4.5 system
ISAC/cb1/cb2c/cb2n:
CaseBase
CaseStruct
RealSet
Solutions
target
ISAC/utilities:
AddEnd
AddOne
DisplayDir
EliminTab
MakeLoops
```

Main.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <ctype.h>
#include <time.h>
#include "header1.h"
#include "header2.h"

int shift,representation,multipl,options,updateWeights;
int global,average,GUM,GDU,BUU,BDM;
int randomWeight,DoAlsoGlobal,DoGraphic;
char TypeOfSimulation[12];

char** ReadSol(int *NumSol)
{
 int i,NumFeat=0;
 char line[80],*token,String[32],**Sol;
 FILE *fileptr;
 //I read the possible solutions
 if(!(fileptr=fopen(SolFile,"r")))
  {
   cout << "\nError: can't open the file with the solutions: "
        << SolFile << "\n";
   exit(0);
  }
 strcpy(line,Read(fileptr));
 *NumSol=atoi(line);
 Sol= new char*[*NumSol];
 for(i=0;i<*NumSol;i++)
  {
   Sol[i]= new char[32];
   strcpy(line,Read(fileptr));
   token=strtok(line," ");
   strcpy(Sol[i],DelEOL(token));
  }
 fclose(fileptr);
 return Sol;
}

char* Read(FILE* fileptr)
{
 char line[80];
 while(fgets(line,80,fileptr))
  if(!(((line[0]=='/')&&(line[1]=='/'))||(line[0]=='\n')||
        (line[0]==' ')))
   return line;
 return NULL;
}

char* DelEOL(char* tok)
{
 int i=0;
 char ausil[80];
 strcpy(ausil,tok);
 while((tok[i]!=' ')&&(tok[i] != '\n')&&tok[i])
  i++;
 ausil[i] = '\0';
 return ausil;
}

void CreateCopy(char* filename)
// it copies the case-base into ".CBCopy" and ".CBCopyBis"
{
 char line[80];
 FILE *fileptr,*dest1,*dest2;
 if(!(fileptr=fopen(filename,"r")))
  {
   cout << "\nError: I cannot open file *" << filename <<
           "* for the Case Base\n";
   exit(0);
  }

 if(!(dest1=fopen(".CBCopy","w")))
  {
   cout << "\nError: can't open .CBCopy \n";
   exit(0);
  }

 if(!(dest2=fopen(".CBCopyBis","w")))
  {
   cout << "\nError: can't open .CBCopyBis \n";
```

```
   exit(0);
  }

 while(fgets(line,80,fileptr))
  {
   fprintf(dest1,"%s",line);
   fprintf(dest2,"%s",line);
  }
 fclose(fileptr);
 fclose(dest1);
 fclose(dest2);
}

main(int argc,char** argv)
{
 char choice,TimeName[32];
 int repetition=2;
 OneCase *CaseList=NULL,*TargetList=NULL;
 branch *Branches=NULL;
 TypeNode *StructList=NULL;

 options=0;
 shift=7;
 multipl=0;
 NumForTrainingSet=40;
 MaxIterations=20;
 randomWeight=0;
 representation=2;
 DoAlsoGlobal=0;
 average=0;
 DoGraphic=0;

 if(argc==1)
  {
   cout << "\nParameters for ISAC:";
   cout << "\n -o show options";
   cout << "\n -r1 or -r2 (for OneInOne or TwoInOne)";
   cout << "\n -c numCases (num of cases in training Set)";
   cout << "\n -m0 if adding increment -m1 if multiplying";
   cout << "\n -s simulation (the config of GUM etc. we
want)";
   cout << "\n -a0 if weights=1 -a1 if random weights";
   cout << "\n -h value of shift";
   cout << "\n -u0 don't do experiment with global -u1 do
experiment";
   cout << "\n -x0 if I.L. on global feat, -x1 if global is
average di local";
   cout << "\n -y1 if I want graphic with perf on training set
and on test set";
   cout << "\n -i iterations (num of iterations)" << endl;
   exit(0);
  }

 for(int i=1;i<argc;i++)
  {
   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='O'))
    options=1;

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='R'))
    representation=argv[i][2]-48; // 1 or 2

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='A'))
    {
     randomWeight=argv[i][2]-48; // 0 or 1
     cout << "\nRandom Weights: " << randomWeight;
    }

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='M'))
    {
     multipl=argv[i][2]-48; // 0 or 1
     cout << "\nMultipl: " << multipl;
    }

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='C'))
    {
     NumForTrainingSet=atoi(argv[i+1]);
     cout << "\nNumForTrainingSet: " << NumForTrainingSet;
    }

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='X'))
    {
     average=argv[i][2]-48; // 0 or 1
     cout << "\nAverage for global: " << average;
    }

   if((argv[i][0]=='-')&&(toupper(argv[i][1])=='Y'))
    {
     DoGraphic=argv[i][2]-48; // 0 or 1
     cout << "\nDoing graphic" << endl;
    }
```

```
if((argv[i][0]=='-')&&(toupper(argv[i][1])=='H'))
 {
  shift=atoi(argv[i+1]);
  cout << "\nShift: " << shift;
 }

if((argv[i][0]=='-')&&(toupper(argv[i][1])=='U'))
 {
  DoAlsoGlobal=argv[i][2]-48; // 0 or 1
  cout << "\nDoAlsoGLobal: " << DoAlsoGlobal;
 }

if((argv[i][0]=='-')&&(toupper(argv[i][1])=='I'))
 {
  MaxIterations=atoi(argv[i+1]);
  cout << "\nMaxIterations: " << MaxIterations;
 }

if((argv[i][0]=='-')&&(toupper(argv[i][1])=='S'))
 {
  strcpy(TypeOfSimulation,argv[i+1]);

  if(strcmp(argv[i+1],"onlyBad")==0)
   {
    GUM=0; GDU=0; BUU=1; BDM=1;
    cout << "\nType of simulation: onlyBad";
   }
  else if(strcmp(argv[i+1],"onlyGood")==0)
   {
    GUM=1; GDU=1; BUU=0; BDM=0;
    cout << "\nType of simulation: onlyGood";
   }
  else if(strcmp(argv[i+1],"allFour")==0)
   {
    GUM=1; GDU=1; BUU=1; BDM=1;
    cout << "\nType of simulation: allFour";
   }
  else if(strcmp(argv[i+1],"onlyGUM")==0)
   {
    GUM=1; GDU=0; BUU=0; BDM=0;
    cout << "\nType of simulation: onlyGUM";
   }
  else if(strcmp(argv[i+1],"onlyGDU")==0)
   {
    GUM=0; GDU=1; BUU=0; BDM=0;
    cout << "\nType of simulation: onlyGDU";
   }
  else if(strcmp(argv[i+1],"onlyBUU")==0)
   {
    GUM=0; GDU=0; BUU=1; BDM=0;
    cout << "\nType of simulation: onlyBUU";
   }
  else if(strcmp(argv[i+1],"onlyBDM")==0)
   {
    GUM=0; GDU=0; BUU=0; BDM=1;
    cout << "\nType of simulation: onlyBDM";
   }
  else if(strcmp(argv[i+1],"withoutGDU")==0)
   {
    GUM=1; GDU=0; BUU=1; BDM=1;
    cout << "\nType of simulation: withoutGDU";
   }
  else if(strcmp(argv[i+1],"withoutGUM")==0)
   {
    GUM=0; GDU=1; BUU=1; BDM=1;
    cout << "\nType of simulation: withoutGUM";
   }
  else if(strcmp(argv[i+1],"withoutBUU")==0)
   {
    GUM=1; GDU=1; BUU=0; BDM=1;
    cout << "\nType of simulation: withoutBUU";
   }
  else if(strcmp(argv[i+1],"withoutBDM")==0)
   {
    GUM=1; GDU=1; BUU=1; BDM=0;
    cout << "\nType of simulation: withoutBDM";
   }
  else
   {
    cout << "\nNOT found the configuration *" << argv[i+1]
<<"*"<< endl;
    exit(0);
   }
  }
 }

 StructList=ReadCaseStruct(FileCaseStruct);
 //ShowCaseStruct(StructList);
```

```
 //cout << "\nAlt: Shuffle works only for the TwoInOne case
representation!";
 //Shuffle(FileCaseBase,StructList);
 //Shuffle(FileRealSet,StructList);

 if(options)
  {
   cout << "\nOptions:";
   cout << "\n n\tnormal retrieval";
   cout << "\n w\tintrospective learning";
   cout << "\n p\ttest for pivotals";
   cout << "\n i\twith Target in CaseBase";
   cout << "\n l\twith Target NOT in CaseBase";
   cout << "\n b\tBig random Case Base";
   cout << "\n e\tEliminate NIL values";
   cout << "\n y\tbuild a toy CaseBase for pivotal";
   cout << "\n d\tdiscriminatory power";
   cout << "\n t\ttime calculation";
   cout << "\n o\ttime with old algorithm(FlatSearch)";
   cout << "\n 4\tC45(L.O.IN)";
   cout << "\n 5\tC45(L.O.OUT)";
   cout << "\n a\tcalculate AVE and SMA\n q\tquit" << endl;

   cout << "\nChoice: ";
   cin >> choice;

   switch(choice)
    {
     case 'n' : // ******real system**
             TargetList=ReadAllTargets(FileTarget,StructList);
             CaseList=ReadCaseBase(FileCaseBase,StructList);
             Branches=BuildWebOfPointers(StructList,CaseList);

FindCases(CaseList,TargetList,Branches,StructList,"times");
             TargetList=Delete(TargetList);
             break;

     case 'i' : // ******Leave One In**
             CreateCopy(FileCaseBase);
             LeaveOneIn(StructList,"times");
             break;

     case 'l' : // ******Leave One Out**
             CaseList=ReadCaseBase(FileCaseBase,StructList);
             Branches=BuildWebOfPointers(StructList,CaseList);
             LeaveOneOut(StructList,CaseList,Branches);
            break;

     case 'w' : // ******Introspective learning**
             IntrospectiveTest(StructList);
             break;

     case 'p' : // ******test for pivotals**
             TestForPivotals(StructList);
             break;

     case 'd' : // ******discrimination power**
             cout << "\nMust add \"None\" in file Solutions
and"
                  << "all cases must have a solution"
                  << "\nNow give a number";
             CaseList=ReadCaseBase(FileCaseBase,StructList);
             StructList=BuildDecTree(StructList,CaseList);
             break;

     case 'b' : // ******random case-base (leave one IN)**
             BigCaseBase(StructList);
             CreateCopy(BigFile);
             LeaveOneIn(StructList,"times");
             break;

     case 'e' : // ******Eliminate NIL values**
             CaseList=ReadCaseBase(FileCaseBase,StructList);
             EliminateNIL(CaseList,StructList);
             break;

     case 'y' : // *****Build a toy CaseBase for the pivotal
tests*
             ToyCaseBase();
             break;

     case 't' : // ******time with random case base**
            cout << "\nName of the output file: ";
             cin >> TimeName;
             BigCaseBase(StructList);
             CreateCopy(BigFile);
             cout << "\nrepetition=2" << endl;
            repetition=2;
             break;
```

```
    case 'o' : // ******time with old algorithm*******
           cout << "\nName of the output file: ";
             cin >> TimeName;
             BigCaseBase(StructList);
             CreateCopy(BigFile);
             cout << "\nrepetition=2" << endl;
           repetition=2;
           break;

    case '4' : // ****C45(LeaveOneIN)*******
             CreateCopy(FileCaseBase);
             C45_IN(StructList);
           exit(0);
             break;

    case '5' : // ****C45(LeaveOneOUT)*******
             CreateCopy(FileCaseBase);
             C45_OUT(StructList);
           exit(0);
             break;

    case 'a' : // ***calculate AVE and SMA*****
             AVE(StructList);
             break;

    case 'q' :exit(0);
```

FindCases.C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include "header1.h"
#include "header2.h"
#include <time.h>

double FindMaxAct(SimCase *SubList)
{
 double MaxAct=-999;
 SimCase *ptr=SubList;
 while(ptr!=NULL)
  {
```

```
             break;

     default  :cout << "\n\nNo options with this key!!" <<
endl;
             exit(0);
    }
  }
 else // only retrieval
  {
   TargetList=ReadAllTargets(FileTarget,StructList);
   CaseList=ReadCaseBase(FileCaseBase,StructList);
   Branches=BuildWebOfPointers(StructList,CaseList);
   FindCases(CaseList,TargetList,Branches,StructList,"times");
   TargetList=Delete(TargetList);
  }

 system("rm -f .CB*");
 system("rm -f simul");
 system("rm -f simul2");
 system("rm -f BigCB");
 system("rm -f auxil");
 system("rm -f .CBCopy");
 system("rm -f .CBCopyBis");
 system("rm -f WrittenCaseBase");
 cout << "\n";
}
```

```
   if(MaxAct<ptr->GiveAct())
    MaxAct=ptr->GiveAct();
   ptr=ptr->next;
  }
 return MaxAct;
}

void ResetActivation(OneCase *CaseList)
{
 OneCase *PCase=CaseList;
 while(PCase!=NULL)
  {
   PCase->ResetAct();
   PCase=PCase->next;
  }
```

```
}

void ResetCaseBase(OneCase *CaseList)
{
 OneCase *PCase=CaseList;
 while(PCase!=NULL)
  {
   OneFeat *PFeat=PCase->GiveFeats();
   while(PFeat!=NULL)
    {
     PFeat->ChangeHighest(1.0);
     PFeat->ChangeWeight(1.0);
     PFeat->SwapWeights();
     PFeat=PFeat->next;
    }
   PCase->Kc=1.0;
   PCase->Fc=1.0;
   PCase->ThisCaseIsUsed=0;
   PCase->ResetAct();
   PCase=PCase->next;
  }
}

void FindCases(OneCase* CaseList,OneCase* TargetList,
               branch* Branches, TypeNode* StructList,
               char *TimeName)
{
 FILE *results;
 char FinalSol[160];
 SimCase *SubList=NULL,*FinalList=NULL,*ptr=NULL;
 double MaxAct=0;
 OneCase *PTarget=TargetList,*PCase=NULL;
 SAN *ListOfSol=NULL;
 long t1,t2;

 results=fopen(ResultsFile,"a");

 while(PTarget!=NULL)
  { //big loop
   fprintf(results,"\nTarget %s %s",PTarget-
>GiveName(),PTarget->GiveSol());
   if(ThereAreNoConstraints(StructList)) //No need of Base
filtering
    {
     PCase=CaseList;
     while(PCase!=NULL)
      {
       // The case goes in SubList only if it is not a target
       // for testing purposes
       if(PCase->ThisCaseIsUsed==0)
        SubList=new SimCase(PCase,SubList);
       PCase=PCase->next;
      }
    }
   else // There is at least one constraint
    SubList=BaseFiltering(PTarget,Branches,StructList);

   if(global==0)
    SpreadingActivation(SubList,StructList,PTarget,Branches);
   else

GlobalSpreadingActivation(SubList,StructList,PTarget,Branches)
;

   // I find the maximum activation
   MaxAct=FindMaxAct(SubList);
   //cout << "\nMaxAct: " << MaxAct;
   if(MaxAct>0) // if the max act is > 0 I give solution
    {
     ptr=SubList;
     while(ptr!=NULL)
      {
       if(ptr->GiveAct()==MaxAct)
        {
         fprintf(results,"\nRetrieved: %s %s %f",ptr-
>GiveName(),
                           ptr->GiveSol(),ptr->GiveAct());

         FinalList= new SimCase(ptr->GiveCase(),FinalList);
         NumCasesRetrieved++;
        }
       ptr=ptr->next;
      }

     if((updateWeights==1)&&(multipl==0)&&(global==0))
      LocalWeightsSum(FinalList,PTarget,StructList);
     if((updateWeights==1)&&(multipl==1)&&(global==0))
      LocalWeightsMul(FinalList,PTarget,StructList);
     if((updateWeights==1)&&(multipl==0)&&(global==1))
      GlobalWeights(FinalList,PTarget,StructList);
```

```
     ListOfSol=Analyse(FinalList,PTarget-
>GiveName(),MaxAct,ListOfSol);
    }
   else // if max act is <=0 I don't give sol
    ListOfSol= new SAN("None","None",0,ListOfSol);

   SubList=Delete(SubList);
   FinalList=Delete(FinalList);

   ResetActivation(CaseList);
   PTarget=PTarget->next;
  } //big loop

 fclose(results);

 strcpy(FinalSol,ChooseFinal(ListOfSol));
 ListOfSol=Delete(ListOfSol);

 results=fopen(ResultsFile,"a");
 fprintf(results,"\nSol: %s",FinalSol);
 fprintf(results,"\n---------\n");
 fclose(results);

 char CommandLine[80],position[32],StringaDaStampare[320];
 FILE *TipoDiStringa;

 /*LBONZ*/

TipoDiStringa=fopen("/dd/csc/abonzano/GHMI/tipoDiStringa","r")
;
 fscanf(TipoDiStringa,"%s\n%s",position,StringaDaStampare);
 fclose(TipoDiStringa);

 // strcpy(FinalSol,FilterSol(FinalSol,StringaDaStampare));
 // FilterSol is for giving the wrong solution
 // and to make it readable to controllers

 strcat(StringaDaStampare,FinalSol);

 /*LBONZ*/
 strcpy(CommandLine,"/dd/csc/abonzano/ISAC/Suggestion -
geometry ");
 strcat(CommandLine,position);
 strcat(CommandLine," \"");
 strcat(CommandLine,StringaDaStampare);
 strcat(CommandLine,"\" &");
 system(CommandLine);
}
```

IntrospectiveLearning.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include "header1.h"
#include "header2.h"

int MaxIterations,NumCasesRetrieved;

double Evaluate()
{
 int ItIsCorrect;
 double
NumConflicts=0,NumCorrect=0,NumRetrieved,NumCorrectlyRetrieved
;
 double highestAct=0.0;
 char line[80],*token,Suggestion[80],TargetSol[80];

 FILE *fileptr=fopen("./results","r");

 while(fgets(line,80,fileptr))
  {
   if(line[0]=='T')
    {
     token=strtok(line," ");
     token=strtok(NULL," ");
     token=strtok(NULL," ");
     strcpy(TargetSol,DelEOL(token));
     NumRetrieved=0;
    }

   if(line[0]=='R')
```

```
    {
     token=strtok(line," ");
     token=strtok(NULL," ");
     token=strtok(NULL," ");
     strcpy(Suggestion,DelEOL(token));
     strcat(Suggestion,"&end&");
     token=strtok(NULL," ");
     double activation=atof(token);
     if(highestAct<activation)
      {
       highestAct=activation;
       NumRetrieved=1;
       NumCorrectlyRetrieved=0;
      }
     else
      NumRetrieved++;

     if(CaseIsCorrect(Suggestion,TargetSol))
      NumCorrectlyRetrieved++;
    }

   if(line[0]=='S')
    {
     NumConflicts++;
     if(NumRetrieved!=0)
      NumCorrect+=NumCorrectlyRetrieved/NumRetrieved;
     highestAct=0.0;
    }
  }

 fclose(fileptr);
 double ToBeReturned=NumCorrect/NumConflicts*100;
 if(ToBeReturned>100)
  {
   cout << "\nexiting because the performance is bigger than
100: "
        << ToBeReturned << ". Check the file results" << endl;
   exit(0);
  }
 cout << "\nANALYSING RESULTS: " << 100-ToBeReturned  << " "
<< endl;
 return ToBeReturned;
}

void IntrospectiveTest(TypeNode *StructList)
{
 FILE *target,*globalTest;
 int i,MaxLoop;
 double Etr[20],Ets[20],MaxPerform;
 char FileName[32];
 OneCase
*CaseList=NULL,*TargetList=NULL,*testSet=NULL,*trainingSet=NUL
L;
 branch *Branches=NULL;

 for(i=0;i<MaxIterations;i++)
  {
   Etr[i]=0.0;
   Ets[i]=0.0;
  }

 Training_Test(FileRealSet,StructList);

 global=0;
 cout << "\n *****LOCAL WEIGHTS*****" << endl;
 testSet=ReadCaseBase("TestSet.bis",StructList);
 trainingSet=ReadCaseBase("TrainingSet.bis",StructList);
 CaseList=ReadCaseBase(FileCaseBase,StructList);
 Branches=BuildWebOfPointers(StructList,CaseList);

 updateWeights=0;
 MaxPerform=Testing(testSet,CaseList,Branches,StructList);
 cout << "\n*Error BEFORE IL on test set: " << 100-MaxPerform
<< endl;

 strcpy(FileName,"zzz.");
 strcat(FileName,TypeOfSimulation);

 globalTest=fopen(FileName,"a");
 fprintf(globalTest,"\n%f ",100-MaxPerform);
 fclose(globalTest);

 cout << "\n*Iterating on training set:";
 MaxPerform=0.0;
 for(i=0;i<MaxIterations;i++)
  {
   int VarChangeHighest=0;
   double ThisPerform;

   if(DoGraphic)
```

```
    {
     updateWeights=0;
     Ets[i]=100-Testing(testSet,CaseList,Branches,StructList);
    }

   updateWeights=1;

ThisPerform=Testing(trainingSet,CaseList,Branches,StructList);
   if(ThisPerform>MaxPerform)
    {
     VarChangeHighest=1;
     MaxPerform=ThisPerform;
    }
   UpdateWeights(CaseList,VarChangeHighest);
   Etr[i]=100-ThisPerform;
  }
 UpdateHighest(CaseList);

 if(DoGraphic)
  {
   strcpy(FileName,"ppp.");
   strcat(FileName,TypeOfSimulation);
   strcat(FileName,".etr");
   FILE *fptr1=fopen(FileName,"a");
   strcpy(FileName,"ppp.");
   strcat(FileName,TypeOfSimulation);
   strcat(FileName,".ets");
   FILE *fptr2=fopen(FileName,"a");
   for(i=0;i<MaxIterations;i++)
    {
     fprintf(fptr1,"%2f\t",Etr[i]);
     fprintf(fptr2,"%2f\t",Ets[i]);
    }
   fprintf(fptr1,"\n");
   fprintf(fptr2,"\n");
   fclose(fptr1);
   fclose(fptr2);
   //Histogram(StructList,CaseList);
  }

 updateWeights=0;
 MaxPerform=Testing(testSet,CaseList,Branches,StructList);
 cout << "\n*Error AFTER IL on test set: " << 100-MaxPerform;

 strcpy(FileName,"zzz.");
 strcat(FileName,TypeOfSimulation);
 globalTest=fopen(FileName,"a");
 fprintf(globalTest,"\t%f",100-MaxPerform);
 fclose(globalTest);

 if(DoAlsoGlobal)
  {
   global=1;
   cout << "\n\n\n *****GLOBAL WEIGHTS*****" << endl;
   for(i=0;i<MaxIterations;i++)
    {
     Etr[i]=0.0;
     Ets[i]=0.0;
    }

   updateWeights=0;
   MaxPerform=Testing(testSet,CaseList,Branches,StructList);
   cout << "\n*Error BEFORE IL on test set: " << 100-
MaxPerform;

   strcpy(FileName,"zzz.");
   strcat(FileName,TypeOfSimulation);

   globalTest=fopen(FileName,"a");
   fprintf(globalTest,"\t%f",100-MaxPerform);
   fclose(globalTest);

   if(average==0)
    {
     cout << "\n*Iterating on training set:";
     MaxPerform=0.0;
     for(i=0;i<MaxIterations;i++)
      {
       int VarChangeHighest=0;
       double ThisPerform;
       if(DoGraphic)
        {
         updateWeights=0;
         Ets[i]=100-
Testing(testSet,CaseList,Branches,StructList);
        }

       updateWeights=1;

ThisPerform=Testing(trainingSet,CaseList,Branches,StructList);
```

```
      if(ThisPerform>MaxPerform)
       {
        VarChangeHighest=1;
        MaxPerform=ThisPerform;
       }
      UpdateGlobalWeights(StructList,VarChangeHighest);
      Etr[i]=100-ThisPerform;
     } // end for
    if(DoGraphic)
     {
      strcpy(FileName,"ggg.");
      strcat(FileName,TypeOfSimulation);
      strcat(FileName,".etr");
      FILE *fptr1=fopen(FileName,"a");
      strcpy(FileName,"ggg.");
      strcat(FileName,TypeOfSimulation);
      strcat(FileName,".ets");
      FILE *fptr2=fopen(FileName,"a");
      for(i=0;i<MaxIterations;i++)
       {
        fprintf(fptr1,"%2f\t",Etr[i]);
        fprintf(fptr2,"%2f\t",Ets[i]);
       }
      fprintf(fptr1,"\n");
      fprintf(fptr2,"\n");
      fclose(fptr1);
      fclose(fptr2);
      //Histogram(StructList,CaseList);
     }
   }
  else // average==1
   {
    cout << "\nCalculating the average from the local..." <<
endl;
    CalcAverageFromLocal(StructList,CaseList);
   }
  UpdateGlobalHighest(StructList);

  updateWeights=0;
  MaxPerform=Testing(testSet,CaseList,Branches,StructList);
  cout << "\n*Error AFTER IL on test set: " << 100-
MaxPerform;

  strcpy(FileName,"zzz.");
  strcat(FileName,TypeOfSimulation);
  globalTest=fopen(FileName,"a");
  fprintf(globalTest,"\t%f",100-MaxPerform);
  fclose(globalTest);
 }
testSet=Delete(testSet);
trainingSet=Delete(trainingSet);
CaseList=Delete(CaseList);
Branches=Delete(Branches);
StructList=Delete(StructList);
}

double Testing(OneCase *Set,OneCase *CaseList,branch
*Branches,TypeNode *StructList)
{
 FILE *target;
 OneCase *TargetList=NULL,*PTest=Set;

 system("rm ./results");

 while(PTest!=NULL)
  {
   target=fopen(FileTarget,"w");
   if(representation==1)
    {
     ToFile(PTest,target,StructList);
     fprintf(target,"\n");
     ToFile(PTest->next,target,StructList);
     PTest=PTest->next;
    }
   else
    ToFile(PTest,target,StructList);

   fprintf(target,"\n");
   fclose(target);

   TargetList=ReadAllTargets(FileTarget,StructList);
   FindCases(CaseList,TargetList,Branches,StructList,"times");
   TargetList=Delete(TargetList);

   PTest=PTest->next;
  }
 return Evaluate();
}
```

E.2 Files for the Interface Between ISAC and GHMI

These are the files in the directory GHMI.

```
ISAC_Bada.C
ISAC_Bada.H
ISAC_Bada.data
ISAC_Calculate.C
ISAC_Calculate.H
ISAC_Functions.C
ISAC_Interface.C
ISAC_Interface.H
ISAC_MAC.C
ISAC_MAC.H
ISAC_Print.C
ISAC_Print.H
ISAC_twoAC
ISAC_wrongDir

e1 //TACs
evaluation1

k1 //MACs
konflict1

ISAC_wrongDir:
trafficSamples
zz_andy.bar
zz_dia.hun
zz_frank.dow
zz_guy.tod
zz_leif.lun
zz_loui.sil
zz_peter.eri
zz_ray.dowd
zz_rod.mcg

bada:
AT42__.PTF
SYNONYM.LST
```

ISAC_Bada.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>

void FromBada(char *tipo,double *rate,double *MaxAlt)
{
 int TypeFound=0,typeLen;
 FILE *fileptr;
 char *token,line[180],fileName[12],tempFN[12];
 fileptr=fopen("ISAC_Bada.data","r");
 while(fgets(line,180,fileptr))
  {
   token=strtok(line,"_");
   strcpy(tempFN,token);
   token=strtok(NULL," ");
   while(strcmp(token,"*")!=0)
    {
     if(strcmp(token,tipo)==0)
      {
       strcpy(fileName,tempFN);
       TypeFound=1;
      }
     token=strtok(NULL," ");
    }
  }
fclose(fileptr);

if(TypeFound==0)
 {
```

```
  printf("\nFile for the type NOT found!");
  fflush(stdout);
  exit(0);
 }
typeLen=strlen(fileName);
fflush(stdout);

strcpy(line,"./bada/");
strcat(line,fileName);
for(int i=typeLen;i<6;i++)
 strcat(line,"_");
strcat(line,".PTF");

fileptr=fopen(line,"r");
double aveSpeed=0.0,maxAlt=0.0;
while(fgets(line,180,fileptr))
 {
  // extracting the MaxAltitude and ClimabRate
  token=strtok(line," ");
  if(strcmp(token,"climb")==0)
   {
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    aveSpeed=atof(token);
   }
  if(strcmp(token,"cruise")==0)
   {
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    aveSpeed+=atof(token);
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    maxAlt=atof(token);
   }
  if(strcmp(token,"descent")==0)
   {
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    token=strtok(NULL," ");
    aveSpeed+=atof(token);
   }
 }

fclose(fileptr);
*rate=aveSpeed/3;
*MaxAlt=maxAlt;
}
```

ISAC_Calculate.C

```
#include "CoreConstants.H"
#include "HipsCore.H"
#include "Zones.H"
#include "Rules.H"
#include "Atmosphere.H"
#include "Constraints.H"
#include "HipsCoreAPI.h"
#include "AirPosition.H"
#include <stdio.h>
#include <math.h>
#include "ISAC_Calculate.H"

int CalculatePoints(Hips ph,OneAircraft *AC1,OneAircraft *AC2)
//==================
{
 double xbefore,ybefore;
 Hips_FlightPlan *Pfp;
 Hips_ConflictList *Pcl;

 // calculating the point on the boundary of the horizontal
no-go zone
 Pcl=Hips_GetConflicts(ph,AC1->name);
 int i=0;
 while(strcmp(Pcl->Conflict[i].EnvironmentName,AC2->name)!=0)
  {
   i++;
   if(i==Pcl->NumberOfConflicts)
   // it was -1 in two aircraft conflicts
```

```
    {
     printf("\nWarning: these two aircraft are NOT
conflicting.\n\n");
     return 0;
    }
  }
 AC1->xOnConfl=Pcl->Conflict[i].Point[0].X;
 AC1->yOnConfl=Pcl->Conflict[i].Point[0].Y;
 AC1->timeOnConfl=Pcl->Conflict[i].Point[0].Time;

 // Calculating the point before the one on the conflict
 Pfp=Hips_GetFlightPlan(ph,AC1->name);
 i=0;
 while(AC1->timeOnConfl>Pfp->PlanPoint[i].Time)
 {
  xbefore=Pfp->PlanPoint[i].X;
  ybefore=Pfp->PlanPoint[i].Y;
  i++;
 }
 AC1->xbefore=xbefore;
 AC2->ybefore=ybefore;

 // verify for boundaries
 int NumOnBound=0;
 Pfp=Hips_GetFlightPlan(ph,AC1->name);
 i=0;
 while(i<Pfp->NumberOfPlanPoints)
  {
   if(Pfp->TrajData[i].OnBoundary==1)
    NumOnBound++;
   i++;
  }
 if(NumOnBound!=4)
  printf("\n###%s has %d points on boundaries!!!!###",
         Pfp->Name,NumOnBound);
 return 1;
}

void CalculateHorConflConf(Hips ph,OneAircraft
*AC1,OneAircraft *AC2)
//=======================
{
 // four options (analysed in this order):
 // - head-on if angle bigger than 150
 // - converging if 2 or more pts in common and at the same
point in the future
 // - diverging if 2 or more pts in common and at separate
points in the future
 // - crossing if one point in common
 // I first see if they have more than 2 points together.
 // If yes I check whether they are converging or diverging
 // If not I check whether they are haed-on or crossing
 // all this is independent on the sector boundaries
 int i,j,CommonPoints=0,NOPP1,NOPP2;
 double HCCangle;
 char angle[12];

 HCCangle=GetAngle(AC1->xbefore,AC1->ybefore,
                   AC1->xOnConfl,AC1->yOnConfl,
                    AC2->xbefore,AC2->ybefore,
                   AC2->xOnConfl,AC2->yOnConfl);

 if(HCCangle>BiggestAngle)
  strcpy(angle,"headon");
 else
  { //huge if
   Hips_FlightPlan *Pfp;

   // I store in two vectors the flight plans of the two
aircraft
   // if the flight plan has more than 50 points I have an
error!
   double x1[50],y1[50],time1[50];
   double x2[50],y2[50],time2[50];

   Pfp=Hips_GetFlightPlan(ph,AC1->name); // first aircraft
   if(Pfp->NumberOfPlanPoints>50)
    {
     printf("\nWarning: %s Flight Plan with more than 50
points: ",AC1->name);
     printf("I cannot calculate the Horizontal Conflict
Configuration");
    }
   i=0;
   while(i<Pfp->NumberOfPlanPoints)
    {
     x1[i]=Pfp->PlanPoint[i].X;
     y1[i]=Pfp->PlanPoint[i].Y;
     time1[i]=Pfp->PlanPoint[i].Time;
```

```
     i++;
    }
   NOPP1=Pfp->NumberOfPlanPoints-1;

   Pfp=Hips_GetFlightPlan(ph,AC2->name); // second aircraft
   if(Pfp->NumberOfPlanPoints>50)
    {
     printf("\nWarning: Flight Plan with more than 50 points:
");
     printf("I cannot calculate the Horizontal Conflict
Configuration");
    }
   i=0;
   while(i<Pfp->NumberOfPlanPoints)
    {
     x2[i]=Pfp->PlanPoint[i].X;
     y2[i]=Pfp->PlanPoint[i].Y;
     time2[i]=Pfp->PlanPoint[i].Time;
     i++;
    }
   NOPP2=Pfp->NumberOfPlanPoints-1;

   i=NOPP1;
   while(i>=0)
    {
     j=NOPP2;
     while(j>=0)
      {
       double dist;
       dist=((x1[i]-x2[j])*(x1[i]-x2[j])+
             (y1[i]-y2[j])*(y1[i]-y2[j]));
       if(dist<4)
        CommonPoints++;
       j--;
      }
     i--;
    }

   if(CommonPoints>1)
     // two or more points in common:
     // catching if one of the two last points is in common
     // and at the same time (less than 1 minute) and
     // at the same alt
     // otherwise diverging
    {
     // if any of this point is close to the other, then they
are catching
     // USE WIDESCREEN
     if( ((((x1[NOPP1]-x2[NOPP2])*(x1[NOPP1]-x2[NOPP2])+
            (y1[NOPP1]-y2[NOPP2])*(y1[NOPP1]-y2[NOPP2]))<4)&&
        ((time1[NOPP1]-time2[NOPP2])<1)) ||
        ((((x1[NOPP1-1]-x2[NOPP2])*(x1[NOPP1-1]-x2[NOPP2])+
           (y1[NOPP1-1]-y2[NOPP2])*(y1[NOPP1-1]-
y2[NOPP2]))<4)&&
         ((time1[NOPP1-1]-time2[NOPP2])<1)) ||
         ((((x1[NOPP1]-x2[NOPP2-1])*(x1[NOPP1]-x2[NOPP2-1])+
            (y1[NOPP1]-y2[NOPP2-1])*(y1[NOPP1]-y2[NOPP2-
1]))<4)&&
          ((time1[NOPP1]-time2[NOPP2]-1)<1)) ||
          ((((x1[NOPP1-1]-x2[NOPP2-1])*(x1[NOPP1-1]-x2[NOPP2-
1])+
             (y1[NOPP1-1]-y2[NOPP2-1])*(y1[NOPP1-1]-y2[NOPP2-
1]))<4)&&
          ((time1[NOPP1-1]-time2[NOPP2]-1)<1)) )
       strcpy(angle,"converging");
      else
       strcpy(angle,"diverging");
     }
    else // less than two points crossing
     strcpy(angle,"crossing");
   } // end of huge if
  strcpy(AC1->HorConflConf,angle);
  strcpy(AC2->HorConflConf,angle);
}

void CalculateAltitudeConfiguration(Hips ph,OneAircraft* AC1,
                            OneAircraft* AC2,int alt,int alt1,
                            double time)
//================================
{
 // is the a/c climbing, descending, stable? (WHEN THE
CONFLICT BEGINS)
 int NextAlt,NextAlt1,i;
 Hips_FlightPlan *Pfp;
 Hips_TrajPosition *Ptraj;

 strcpy(AC1->AltIntention,"stable");
 strcpy(AC2->AltIntention,"stable");

 Pfp=Hips_GetFlightPlan(ph,AC1->name);
```

```
 Ptraj=Pfp->PlanPoint;

 i=0;
 while(Ptraj[i].Time<AC1->timeOnConfl)
  i++; // this is the trajectory point before the no-go zone
begins
 i--;

 if(i==Pfp->NumberOfPlanPoints-1)
  printf("\nsomething strange in AltIntention");

 if((int)Ptraj[i+1].Altitude-(int)Ptraj[i].Altitude>5)
  strcpy(AC1->AltIntention,"climbing");
 else if((int)Ptraj[i].Altitude-(int)Ptraj[i+1].Altitude>5)
  strcpy(AC1->AltIntention,"descending");

 Pfp=Hips_GetFlightPlan(ph,AC2->name);
 Ptraj=Pfp->PlanPoint;

 i=0;
 while(Ptraj[i].Time<AC1->timeOnConfl)
  i++; // the aircraft is before this trajectory point
 i--;

 if(i==Pfp->NumberOfPlanPoints-1)
  printf("\nsomething strange in AltIntention");

 if((int)Ptraj[i+1].Altitude-(int)Ptraj[i].Altitude>5)
  strcpy(AC2->AltIntention,"climbing");
 else if((int)Ptraj[i].Altitude-(int)Ptraj[i+1].Altitude>5)
  strcpy(AC2->AltIntention,"descending");

 // calculating SomebodyClimbing
 if((strcmp(AC1->AltIntention,"climbing")==0)||
    (strcmp(AC2->AltIntention,"climbing")==0))
  {
   strcpy(AC1->SomebodyClimbing,"yes");
   strcpy(AC2->SomebodyClimbing,"yes");
  }
 else
  {
   strcpy(AC1->SomebodyClimbing,"no");
   strcpy(AC2->SomebodyClimbing,"no");
  }
}
```

```
void CalculateEasyToExit(Hips ph,OneAircraft *AC1,OneAircraft
*AC2)
//======================
{
 // more accurate function: I don't calculate the angles from
where
 // the aircraft is, but from the previous pointS considering
the no-go zone
 // moreover, I don't use any more the centre of the conflict
but
 // the actual trajectory of the aircraft

 // EasyToExitRight/Left values:
 // - veryEasy (if the aircraft is already turning that
direction and the
 //   angle is less than 10 degrees or if the angle is less
than 5 degrees)
 // - easy (if the angle is less than 10 degrees)
 // - possible (if the angle is between 10 and 15 degrees)
 // - difficult (if the angle is bigger than 15 degrees)
 // All these value imply that there must not be other
environmental
 // no-go zones

 int i,j,startingK,first=1,last;
 double angle,globalAngle,lgap,rgap,lnogo,rnogo;
 double xextreme,yextreme;
 double SmallestLnogo=999,SmallestRnogo=999;
 double SmallestLavail=999,SmallestRavail=999;
 Hips_FlightPlan *Pfp;
 Hips_ZoneList *Pzl;
 Hips_Zone *Pz;
 Pzl=Hips_GetZones(ph,Hips_RouteDiagram);
 Pz=Pzl->Zone;
 Pfp=Hips_GetFlightPlan(ph,AC1->name);

 // I check if the aircraft is turning left or right
 while(Pfp->TrajData[first].OnBoundary!=1)
  first++;
 last=Pfp->NumberOfPlanPoints-1;
 while(Pfp->TrajData[last].OnBoundary!=1)
  last--;
 if(first>last)
  {
```

```
   printf("\nError: something wrong in the flight plan of
%s\n\n",
           Pfp->Name);
   exit(0);
  }

 if(last==first+1)
  globalAngle=0;
 else
  globalAngle=GetAngle2(Pfp->PlanPoint[first-1].X,Pfp-
>PlanPoint[first-1].Y,
             Pfp->PlanPoint[last].X,Pfp->PlanPoint[last].Y,
             Pfp->PlanPoint[first].X,Pfp->PlanPoint[first].Y);

 // I need to find the point on the traj which is on the
sector boundary
 // but before the point on the no-go zone boundary

 for(int kk=0;kk<Pfp->NumberOfPlanPoints;kk++)
  if((Pfp->TrajData[kk].OnBoundary==1)&&
     (Pfp->PlanPoint[kk].Time<AC1->timeOnConfl))
   startingK=kk;

 while(AC1->timeOnConfl>Pfp->PlanPoint[startingK].Time)
  {
   double lnogo=0,rnogo=0,lavail=-90,ravail=90;
   for(i=0;i<Pzl->NumberOfZones;i++)
    {
     if(strcmp(Pz[i].EnvironmentName,AC2->name)==0)
      {
       for(j=0;j<Pz[i].NumberOfPoints;j++)
        {
         xextreme=Pz[i].Point[j].U;
         yextreme=Pz[i].Point[j].V;
         angle=GetAngle2(Pfp->PlanPoint[startingK].X,
                    Pfp->PlanPoint[startingK].Y,
                            AC1->xOnConfl,AC1->yOnConfl,
                     xextreme,yextreme);
         if(angle<lnogo)
          lnogo=angle;
         if(angle>rnogo)
          rnogo=angle;
        }
      }
     else
      {
       for(j=0;j<Pz[i].NumberOfPoints;j++)
        {
         xextreme=Pz[i].Point[j].U;
         yextreme=Pz[i].Point[j].V;
         angle=GetAngle2(Pfp->PlanPoint[startingK].X,
                     Pfp->PlanPoint[startingK].Y,
                       AC1->xOnConfl,AC1->yOnConfl,
                       xextreme,yextreme);
         if((angle<0)&&(fabs(angle)<fabs(lavail)))
          lavail=angle;
         if((angle<0)&&(angle<ravail))
          ravail=angle;
        }
      }
    }

   if(SmallestLnogo>fabs(lnogo))
    SmallestLnogo=fabs(lnogo);
   if(SmallestRnogo>fabs(rnogo))
    SmallestRnogo=fabs(rnogo);
   if(SmallestLavail>fabs(lavail))
    SmallestLavail=fabs(lavail);
   if(SmallestRavail>fabs(ravail))
    SmallestRavail=fabs(ravail);
   startingK++;
  }

 lnogo=fabs(SmallestLnogo);
 rnogo=fabs(SmallestRnogo);

 if(SmallestLavail<SmallestLnogo)
  lgap=0;
 else
  lgap=fabs(SmallestLavail-SmallestLnogo);

 if(SmallestRavail<SmallestRnogo)
  rgap=0;
 else
  rgap=fabs(SmallestRavail-SmallestRnogo);

 // here I should be able to determine whether the no-go zone
is on the
 // sector boundaries or not
```

```
 // here I could put as the minimum angle not 10 but the
maximum between
 // 10 and globalAngle

 if(((rgap>5)&&(rnogo<10)&&(globalAngle>0))||(rnogo<5))
  strcpy(AC1->EasyToExitRight,"veryEasy");
 else if((rgap>5)&&(rnogo<10))
  strcpy(AC1->EasyToExitRight,"easy");
 else if((rgap>5)&&(rnogo<15))
  strcpy(AC1->EasyToExitRight,"possible");
 else
  strcpy(AC1->EasyToExitRight,"difficult");

 if(((lgap>5)&&(lnogo<10)&&(globalAngle<0))||(lnogo<5))
  strcpy(AC1->EasyToExitLeft,"veryEasy");
 else if((lgap>5)&&(lnogo<10))
  strcpy(AC1->EasyToExitLeft,"easy");
 else if((lgap>5)&&(lnogo<15))
  strcpy(AC1->EasyToExitLeft,"possible");
 else
  strcpy(AC1->EasyToExitLeft,"difficult");

 if((strcmp(AC1->EasyToExitRight,"veryEasy")==0)||
    (strcmp(AC1->EasyToExitLeft,"veryEasy")==0))
  strcpy(AC1->EasyToExitHorizontally,"veryEasy");
 else if((strcmp(AC1->EasyToExitRight,"easy")==0)||
    (strcmp(AC1->EasyToExitLeft,"easy")==0))
  strcpy(AC1->EasyToExitHorizontally,"easy");
 else if((strcmp(AC1->EasyToExitRight,"possible")==0)||
    (strcmp(AC1->EasyToExitLeft,"possible")==0))
  strcpy(AC1->EasyToExitHorizontally,"possible");
 else
  strcpy(AC1->EasyToExitHorizontally,"difficult");
}

void CalculateBoundaries(Hips ph,OneAircraft *AC)
//======================
{
 // The distance to the boundary is the distance between the
 // first point of the trajectory which is in the no-go zone
 // and the entry or exit point in the sector, i.e the points
 // of the trajectory which are on the boundary
 // the entry point is important, too, because if too close to
 // the entry point the controller should coordinate with the
 // previous sector
 int ind=0;
 double timeBefore,timeAfter;
 Hips_FlightPlan *Pfp;
 Pfp=Hips_GetFlightPlan(ph,AC->name);

 // I already have xOnConfl and yOnConfl, I need the xOnBound,
yOnBound
 while(Pfp->PlanPoint[ind].Time<AC->timeOnConfl)
  {
   if(Pfp->TrajData[ind].OnBoundary==1)
    timeBefore=Pfp->PlanPoint[ind].Time;
   ind++;
  }

 ind=0;
 while(!((Pfp->PlanPoint[ind].Time>AC->timeOnConfl)&&
       (Pfp->TrajData[ind].OnBoundary==1)))
  ind++;
 timeAfter=Pfp->PlanPoint[ind].Time;

 // ask NIGEL for here: is it 4 minutes from the a/c or
 // from the beginning of the conflict?
 // CloseToBound is the smallest time to go to the closest
boundary
 double minimus=timeAfter-AC->timeOnConfl;
 if(minimus>AC->timeOnConfl-timeBefore)
  minimus=AC->timeOnConfl-timeBefore;
 AC->CloseToBound=minimus;
}

void CalculateLevelsAvailable(Hips ph,OneAircraft *AC1,double
InitLevel,
                              double time)
//============================
{
 int i;
 level lev[NumLevels];
 double FinalLevel;
 Hips_FlightPlan *Pfp;

 Pfp=Hips_GetFlightPlan(ph,AC1->name);
 FinalLevel=Pfp->PlanPoint[Pfp->NumberOfPlanPoints-
1].Altitude;
```

```
 // the final level is the altitude of the last point of the
flight plan

 // I look where in the trajectory, the a/c is
 int j=0;
 Hips_DiagramPoint Start;
 while((Pfp->PlanPoint[j+1].Time<time)&&((j+1)<Pfp-
>NumberOfPlanPoints))
  j++;
 Start=Hips_MapPoint(ph,Pfp-
>PlanPoint[j],Hips_AltitudeDiagram);

 ResetLevels(lev);
 CheckLevels(lev,ph,Start.U); // Start.U the X of the aircraft
in the
                          // Hips_AltitudeDiagram

 // I look for the indices of the initial and final levels
 double ClosestFinal=999,ClosestInit=999; // this is the
distance of the
      // actual aircraft level from one of the allowed
altitude levels
 int FinalInd=0,InitialInd=0;
 for(i=0;i<NumLevels;i++)
  {
   if(ClosestInit>(fabs(InitLevel-lev[i].name)))
    {
     ClosestInit=(fabs(InitLevel-lev[i].name));
     InitialInd=i;
    }
   if(ClosestFinal>(fabs(FinalLevel-lev[i].name)))
    {
     ClosestFinal=(fabs(FinalLevel-lev[i].name));
     FinalInd=i;
    }
  }

 //If the aircraft is Stable the possible values are:
 // - NONE (it means that in each of the levels above and
below there is at
 //         least a no go zone generated by another aircraft)
 // - YESABOVE (it means that one of the two levels above is
completely free)
 // - YESBELOW (it means that one of the two levels below is
completely free)
 // If the aircraft is climbing or descending the possible
values are:
 // - NONE (it means that none of the intermediate levels, the
starting level
 //         and the final level are free)
 // - YES (it means that there is at least one level
available: completely free)
 // - YESWITHSPACES (it means that there are no levels free,
but in some there
 //                  are some spaces between the no go zones)

 if(InitialInd==FinalInd)
  { // a/c is Stable
   int above=0,below=0;
   if((lev[InitialInd].free==1)||(lev[InitialInd+1].free==1)||
      (lev[InitialInd+2].free==1))
   above=1;
   if((lev[InitialInd].free==1)||(lev[InitialInd-1].free==1)||
      (lev[InitialInd-2].free==1))
   below=1;
   if((above==1)&&(below==1))
    strcpy(AC1->LevelsAvailable,"yes");
   else if(above==1)
    strcpy(AC1->LevelsAvailable,"above");
   else if(below==1)
    strcpy(AC1->LevelsAvailable,"below");
   else
    strcpy(AC1->LevelsAvailable,"none");
  }
 else
  { // a/c is climbing or descending
   int found=0,spaces=0;
   if(InitialInd>FinalInd) // descending
    for(i=InitialInd;i>=FinalInd;i--)
     {
      if(lev[i].free==1)
       found=1;
      if(lev[i].spaces==1)
       spaces=1;
     }
   else // climbing
    for(i=InitialInd;i<=FinalInd;i++)
     {
      if(lev[i].free==1)
```

```
       found=1;
      if(lev[i].spaces==1)
       spaces=1;
     }
   if(found==1)
    strcpy(AC1->LevelsAvailable,"yes");
   else if(spaces==1)
    strcpy(AC1->LevelsAvailable,"withSpaces");
   else
    strcpy(AC1->LevelsAvailable,"none");
  }
}

void CalculateSpeed(Hips ph,OneAircraft *AC1,OneAircraft
*AC2,double time)
//=================
{
 // I calculate whether it is easy, possible or difficult to
exit the no-go
 // zone by increasing or decreasing the speed.
 // Easy -> change less than 0.01 Mach
 // Possible -> 0.02 M
 // Difficult -> 0.03 M and more
 // the value does not depend on the environment aircraft
(yellow no go zones)
 // but only on the red ones!!!!! CHANGE THIS!!!!!
 // fare come in Hor angle: calcola TUTTE le speed e prendo la
piu'
 // grossa per la red zone e la piu' piccola per la yellow
zone and see
 // if possible etc.
 int j,zindice;
 double mach;
 Hips_ZoneList *Pzl;
 Hips_Zone *Pz;
 Hips_FlightPlan *Pfp;
 Hips_DiagramPoint Start,Max,Min;
 Max.V=-999;
 Min.V=999;

 Pfp=Hips_GetFlightPlan(ph,AC1->name);
 Pzl=Hips_GetZones(ph,Hips_SpeedDiagram);
 Pz=Pzl->Zone;

 zindice=0;
 while(zindice<Pzl->NumberOfZones)
  {
   if(strcmp(Pz[zindice].EnvironmentName,AC2->name)==0)
    for(j=0;j<Pz[zindice].NumberOfPoints;j++)
     {
      if(Max.V<Pz[zindice].Point[j].V)
       {
        Max.V=Pz[zindice].Point[j].V;
        Max.U=Pz[zindice].Point[j].U;
       }
      if(Min.V>Pz[zindice].Point[j].V)
       {
        Min.V=Pz[zindice].Point[j].V;
        Min.U=Pz[zindice].Point[j].U;
       }
     }
   zindice++;
  }

 // I look where in the trajectory, the a/c is
 int i=0;
 while((Pfp->PlanPoint[i+1].Time<time)&&((i+1)<Pfp-
>NumberOfPlanPoints))
  i++;

 // I find the mach speed of the point where the a/c is
 if(Pfp->TrajData[i].CasNotMach==1)
  {
   // I convert from CAS to MACH
   printf("\nflying in CAS");
   mach=Mach_From_TAS(TAS_From_CAS(Pfp->TrajData[i].CasMach,
                                   Pfp->PlanPoint[i].Altitude),
                      Pfp->PlanPoint[i].Altitude);
  }
 else
  mach=Pfp->TrajData[i].CasMach;

 // I look for the actual position of the a/c in the Speed
window
 double t,d,mt,md,MaxGS,MinGS,MaxMach,MinMach;
 FlightPosition fp;

 Start=Hips_MapPoint(ph,Pfp->PlanPoint[i],Hips_SpeedDiagram);
 UnMapPoint(Start,fp,Hips_SpeedDiagram,(HipsCore*)ph);
 t=fp.Time;
```

```
 d=fp.Distance;

 // I look for the Mach speed of the Max & Min points
 // I suppose that the altitude remains the same
 UnMapPoint(Max,fp,Hips_SpeedDiagram,(HipsCore*)ph);
 mt=fp.Time;
 md=fp.Distance;
 MaxGS=(md-d)/(mt-t)*60;
 MaxMach=Mach_From_TAS(MaxGS,Pfp->PlanPoint[i].Altitude);

 UnMapPoint(Min,fp,Hips_SpeedDiagram,(HipsCore*)ph);
 mt=fp.Time;
 md=fp.Distance;
 MinGS=(md-d)/(mt-t)*60;
 MinMach= Mach_From_TAS(MinGS,Pfp->PlanPoint[i].Altitude);

 if(fabs(MaxMach-mach)<=0.01)
  strcpy(AC1->Faster,"easy");
 else if(fabs(MaxMach-mach)>0.02)
  strcpy(AC1->Faster,"difficult");
 else
  strcpy(AC1->Faster,"possible");

 if(fabs(mach-MinMach)<=0.01)
  strcpy(AC1->Slower,"easy");
 else if(fabs(mach-MinMach)>0.02)
  strcpy(AC1->Slower,"difficult");
 else
  strcpy(AC1->Slower,"possible");
}

void CalculateInFront(Hips ph,OneAircraft *AC1,OneAircraft
*AC2,
                      double xstart,double ystart)
//===================
{
 // redo the function! don't use xstart & ystart
 /*
 Hips_ZoneList *Pzl;
 Hips_Centre *Pc;
 int i=0,NOC;
 char PassingInFrontDir[5],PassingInFrontSpace[5];
 double angle;

 strcpy(PassingInFrontDir,"no");
 strcpy(AC1->PIFSpace,"no");

 Pzl=Hips_GetZones(ph,Hips_RouteDiagram);
 NOC=Pzl->NumberOfCentres;
 Pc=Pzl->Centre;

 // Passing in front if going directly?
 while(strcmp(AC2->name,Pc[i].EnvironmentName)!=0)
  i++;
 angle=GetAngle2(xstart,ystart,Pc[i].Point.U,Pc[i].Point.V,
                 Pc[i].Vector.U,Pc[i].Vector.V);

 // Attention!!! FinalAngle is not calculated anywhere!!!!
 // look for it in CalculateAvailExit...
 if(((AC1->FinalAngle>0)&&(angle>0))||((AC1-
>FinalAngle<0)&&(angle<0)))
  strcpy(PassingInFrontDir,"yes");
 strcpy(AC1->PIFDirect,PassingInFrontDir);

 // Passing in front if going where there is more space?
 //ATTENTION! lgap lnogo are not calculated anywhere
 // if you need it go to CalculateEasyToExit
 if((AC1->lgap > AC1->rgap)&&(angle<0)) // more space on the
left and other a/c
                        // is going to the left
  strcpy(AC1->PIFSpace,"yes");

 if((AC1->rgap > AC1->lgap)&&(angle>0)) // more space on the
right and other a/c
                       // is going to the right
  strcpy(AC1->PIFSpace,"yes");*/
}

void CalculateFromTo(Hips ph,OneAircraft* AC)
//==================
{
 FILE *fileptr;
 //int trovati=0;
 int foundD=0,foundA=0;
 double dlat,dlon,alat,alon,blat,blon,doneTraj,toDoTraj;
 char line[80],*token,slat[12],slon[12],beacon[12];

 // for depart and arrival
 fileptr=fopen("aerodromes","r");
 while(fgets(line,80,fileptr))
```

```
 {
  token=strtok(line," ");
  if(strcmp(token,AC->depart)==0)
   {
    foundD=1;
    token=strtok(NULL," ");
    strcpy(slat,token);
    token=strtok(NULL," ");
    strcpy(slon,token);
    dlat=LatToDouble(slat);
    dlon=LatToDouble(slon);
   }
  if(strcmp(token,AC->arrival)==0)
   {
    foundA=1;
    token=strtok(NULL," ");
    strcpy(slat,token);
    token=strtok(NULL," ");
    strcpy(slon,token);
    alat=LatToDouble(slat);
    alon=LatToDouble(slon);
   }
 }
fclose(fileptr);

// looking for the beacon name
int ind=0;
char auxBeacon[12];
Hips_FlightPlan *Pfp;
Pfp=Hips_GetFlightPlan(ph,AC->name);

// I take the closest beacon to where the aircraft is.
// Not geographical point because I don't have lat lon.
while(Pfp->PlanPoint[ind].Time<AC->timeOnConfl)
 {
  //printf("\nworking on %s",Pfp->TrajData[ind].BeaconName);
  if(strcmp(Pfp->TrajData[ind].BeaconName,"#GEO")!=0)
   {
    strcpy(auxBeacon,Pfp->TrajData[ind].BeaconName);
    fileptr=fopen("AllBeacons","r");
    while(fgets(line,80,fileptr))
     {
      token=strtok(line," ");
      if(strcmp(token,auxBeacon)==0)
       {
        //printf("\nfound beacon: %s",auxBeacon);
        strcpy(beacon,auxBeacon);
        token=strtok(NULL," ");
        strcpy(slat,token);
        token=strtok(NULL," ");
        strcpy(slon,token);
        blat=LatToDouble(slat);
        blon=LatToDouble(slon);
       }
     }
    fclose(fileptr);
   }
  ind++;
 }

if((foundD)&&(foundA))
 {
  doneTraj=DistanceFrom(dlat,dlon,blat,blon);
  toDoTraj=DistanceFrom(blat,blon,alat,alon);
  double perc=doneTraj/(doneTraj+toDoTraj);
 }

if(foundA)
 { // CloseToTOD is the distance in miles from the TOD
  AC->CloseToTOD=DistanceFrom(blat,blon,alat,alon);
 }
else
 AC->CloseToTOD=-999;
}
```

ISAC_MAC.C

```
#include <stdio.h>
#include <string.h>

void FindSolForMAC()
{
 int Found=0,NumberOfSol=0,Maxi,j,i=0,NOC[12],NumberOfSame;
 char *token,line[256],sols2[12][32],sols[12][32],man[12][8];
 char acMoved[12][12],ac1[12][12],ac2[12][12],aux[256];
```

```
FILE *fileptr;

/*LBONZ*/
fileptr=fopen("/dd/csc/abonzano/ISAC/SolForMAC","r");
while(fgets(line,256,fileptr))
 {
  if(line[0]=='O')
   NumberOfSol++;
  if(strlen(line)>2)
   {
    strcpy(sols[i],line);
    NOC[i]=NumberOfSol;
    i++;
   }
 }
fclose(fileptr);
Maxi=i;

for(i=0;i<Maxi;i++)
 {
  strcpy(aux,sols[i]);
  token=strtok(aux,"_");
  strcpy(man[i],token);
  token=strtok(NULL,"_");
  strcpy(ac1[i],token);
  token=strtok(NULL,"_");
  strcpy(ac2[i],token);
 }

for(i=0;i<Maxi;i++)
 {
  if(man[i][0]=='h')
   strcpy(sols2[i],"hor");
  if(man[i][0]=='s')
   strcpy(sols2[i],"spe");
  if(man[i][0]=='u')
   strcpy(sols2[i],"upp");
  if(man[i][0]=='d')
   strcpy(sols2[i],"dow");

  if(man[i][3]=='1')
   {
    strcat(sols2[i],ac1[i]);
    strcpy(acMoved[i],ac1[i]);
   }
  if(man[i][3]=='2')
   {
    strcat(sols2[i],ac2[i]);
    strcpy(acMoved[i],ac2[i]);
   }
  if(man[i][3]=='3')
   strcpy(acMoved[i],"both");

 }

 // I check if there are sols in common for ALL the different
conflicts
 i=0;
 NumberOfSame=0;
 while(NOC[i]==1)
  {
   char solToCheck[32];
   strcpy(solToCheck,sols2[i]);

   for(j=0;j<Maxi;j++)
    {
     if((NOC[j]>1)&&(strcmp(solToCheck,sols2[j])))
      NumberOfSame++;
    }
    if(NumberOfSame+1==NumberOfSol)
     {
      Found=1;
      printf("\nFINAL SOL (FIRST STEP): %s",solToCheck);
     }
   i++;
  }

 // I check if there is a same aircraft moved in all the
conflicts
 // (useful to do?)

 // I solve the closest conflict in order of time
 printf("\n");
}
```