An Analysis of Information Complexity in Air Traffic Control Human Machine Interaction

by

Christos George Tsonis

B.S. Aerospace Engineering
Georgia Institute of Technology, 2003

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2006

Signature of Author: ....
Department of Aeronautics and Astronautics
May 26, 2006

Certified by: .... [illegible] ....
Mary L. Cummings
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: .... [illegible] ....
Jaime Peraire
Professor of Aeronautics and Astronautics
Chairman, Department Committee on Graduate Students

An Analysis of Information Complexity in Air Traffic Control Human Machine Interaction

by

Christos George Tsonis

Submitted to the Department of Aeronautics and Astronautics
on May 26, 2006 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Aeronautics and Astronautics

ABSTRACT

This thesis proposes, develops and validates a methodology to quantify the complexity of air traffic control (ATC) human-machine interaction (HMI). Within this context, complexity is defined as the *minimum amount of information required to describe the human machine interaction process in some fixed description language and chosen level of detail.* The methodology elicits human information processing via cognitive task analysis (CTA) and expresses the HMI process algorithmically as a cognitive interaction algorithm (CIA). The CIA is comprised of multiple functions which formally describe each of the interaction processes required to complete a nominal set of tasks using a certain machine interface. Complexities of competing interface and task configurations are estimated by weighted summations of the compressed information content of the associated CIA functions. This information compression removes descriptive redundancy and approximates the minimum description length (MDL) of the CIA. The methodology is applied to a representative en-route ATC task and interface, and the complexity measures are compared to performance results obtained experimentally by human-in-the-loop simulations. It is found that the proposed complexity analysis methodology and resulting complexity metrics are able to predict trends in operator performance and workload. This methodology would allow designers and evaluators of human supervisory control (HSC) interfaces the ability to conduct complexity analyses and use complexity measures to more objectively select between competing interface and task configurations. Such a method could complement subjective interface evaluations, and reduce the amount of costly experimental testing.

Thesis Supervisor: Mary L. Cummings
Title: Assistant Professor of Aeronautics and Astronautics

Page intentionally left blank

Acknowledgements

I am forever grateful to my mother and father. Their unbounded love, support and sacrifices allow me the freedom to dream and the strength to pursue life's journeys. Thank you to my little sister for always lighting up my heart and putting a smile on my face. I thank my grandparents for their wisdom and love, and for the sacrifices they made. The love of a family has been my life's greatest blessing.

I am deeply thankful to my advisor Professor Cummings for entrusting and supporting me as her student and for the hard work put into reviewing this thesis. I am grateful for her advice and mentorship in preparation for the challenges of the professional world. I admire Professor Cummings for building and fostering the community that has become the Humans and Automation Laboratory (HAL).

Thank you to HAL post-doctoral associate Dr. Enlie Wang for advising this thesis before leaving MIT. This research would have not been possible without the grant from the FAA's Civil Aerospace Medical Institute (CAMI) entitled "Developing Measurements of Information Complexity for ATC Displays." I thank the technical monitor Dr. Jing Xing of CAMI for her ideas on complexity, and feedback on the thesis proposal and draft. Thank you to Dr. Peter Moertl of MITRE for also reviewing my thesis proposal and draft, and providing valuable feedback.

To my fellow HALiens: Jinho Jang, Paul Mitchell, Sylvain Bruni, Jessica Marquez, Liang Sim, Angela Ho, Cristin Smith, Carl Nehme, Amy Brezynski, Mark Duppen, Daniel Cunha, Chris McQuin, Stacey Scott and Jake Crandall. Thank you all for your help and for sharing this experience. Finally last but not least I thank my great office mates in 35-217: Maital Dar, Gregory Zerbib, Jonathan Histon, Stephan Langhans, Eric Jones, Mike Reyes, Celia Geslin, Georg Theis, Robin Riedel and Tony Abad.

Page intentionally left blank

Table of Contents

Page intentionally left blank

List of Abbreviations

AET	Algorithmic Execution Time
AIC	Algorithmic Information Content
ATC	Air Traffic Control
CIA	Cognitive Interaction Algorithm
CID	Computer Identification
CNS	Central Nervous System
CTA	Cognitive Task Analysis
DSR	Display System Replacement
FAA	Federal Aviation Administration
FL	Flight Level
HMI	Human Machine Interaction
HSC	Human Supervisory Control
MDL	Minimum Description Length
UAV	Unmanned Aerial Vehicle

Page intentionally left blank

List of Figures

Page intentionally left blank

List of Tables

Page intentionally left blank

1. Introduction

Computer and display technologies have matured to a stage where the presentation of information is no longer primarily limited by the technology's ability to process information, but by the human's. More simply put, today's technology allows for a practically unbounded number of configurations for presenting information on a computer display to a human. It is known however that humans have inherent neurophysiologic information processing limits (Cowan, 2001; Kandel, Schwartz, & Jessel, 2000; Miller, 1956; Norman & Bobrow, 1975; Rasmussen, 1986; Wickens & Hollands, 2000) which must be accounted for in the technological design. These essentially correspond to complexity limits at various stages of information processing that physically occur within the central nervous system (CNS).

In many areas of complex interface development, such as air traffic control (ATC), designers could benefit greatly from a structured and practical analytical method for quantifying complexity in order to select between competing interface and task options. A typical ATC interface is shown in Figure 1-1. While there are many factors exogenous to the interface that ultimately contribute to the complexity of human information processing, including environmental and organizational factors, the interface itself can add to the complexity if not carefully designed (Cummings & Tsonis, 2005). In the ATC domain, the complexity imposed on the individual controllers ultimately limits the air traffic capacity and can increase the economic cost of operation of the ATC system. While a successful method for quantifying the complexity of an interface could be applied in a number of domains, in ATC it could lead to the implementation of improved interfaces which could result in a combination of improved safety, increased controller productivity and airspace capacity, reduced system costs, and reduction in training time

Figure 1-1. Air traffic control human-machine interface. The station shown is the Display System Replacement (DSR) used for en-route traffic operations in the USA. Photo courtesy of Federal Aviation Administration (FAA).

In order for a technological interface to achieve its full system performance potential, be operated safely, and at the same time require the least amount of operator training and cost, complexity of the human-machine interaction process must be minimized subject to certain constraints. The engineer designing the human-machine interface is confronted with the challenge of deciding how to most effectively present the information and design the task in a way such that the human can accomplish the required goals accurately and efficiently. An engineer faced with competing design options should be able to ensure that the information complexity relative to the human is low enough to allow for the goals to be accomplished efficiently.

While increased interface complexity due to a poor design can be addressed to some degree with additional training, this is oftentimes counterproductive and costly. A preferable solution would be to provide the engineer with a tool for quantifying the complexity of competing interface and task configurations prior to delivering the technology to operators. This measure can then form the basis of more objective design choices. Such a tool would also be of value when deciding upon the acquisition of competing technological interfaces, such as ATC or unmanned aerial vehicle (UAV) workstations.

This thesis therefore proposes a practical, theoretically based methodology for quantifying information complexity in human-machine interaction. The chosen definition of complexity stems from information theory. Although the definition is elaborated upon in Chapter 2, complexity is defined as the *(minimum) amount of information required to describe an object*[1] *in some fixed description language* (Gell-Mann & Lloyd, 1996; Shannon, 1948). This complexity analysis methodology expresses the interaction process between the human and the machine as an algorithm based upon results of a cognitive task analysis (CTA). This algorithmic scripting of information between the human and interface is named the *cognitive interaction algorithm* (CIA). Complexity can then be estimated by algorithmic information content (Cover & Thomas, 1991) or by some minimum information content such as Kolmogorov Complexity (Li & Vitányi, 1997).

The complexity analysis methodology is summarized in Figure 1-2. It first proposes the application of a CTA for breaking down the task and eliciting cognitive processes for a given interface configuration. Based on the knowledge acquired from the CTA, the CIA is written. In this thesis the estimates of complexity are provided by the algorithmic information content (AIC), and the compressed algorithmic length (representative of minimum description length) of the CIA. This yields a numeric value for a specific task and interface configuration. In order to evaluate the validity of the methodology, the resulting complexity measures generated for each task and interface configuration are compared to the performance results obtained from an experiment conducted with United States Navy ATC trainees engaged in a basic en-route ATC vectoring task. Based on this comparison it is determined whether the complexity analysis performed to examine the effect of different display configurations is a successful predictor of operator performance.

[1] In this thesis the relevant *object* is the human-machine interaction process.

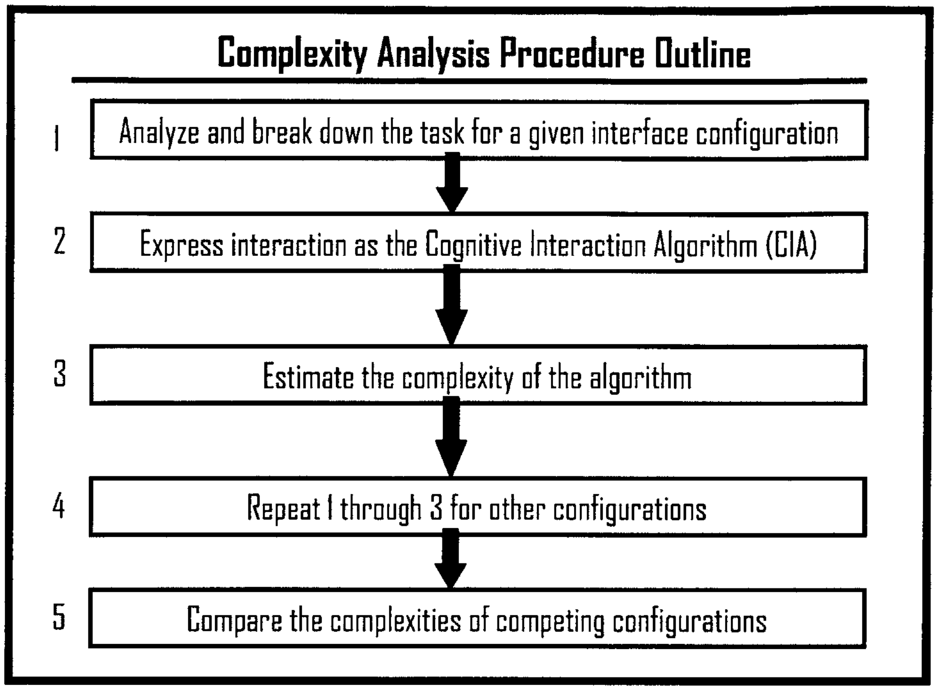

Complexity Analysis Procedure Outline
1 Analyze and break down the task for a given interface configuration
2 Express interaction as the Cognitive Interaction Algorithm (CIA)
3 Estimate the complexity of the algorithm
4 Repeat 1 through 3 for other configurations
5 Compare the complexities of competing configurations

Figure 1-2. Complexity analysis procedure outline

This thesis is organized into five chapters following this introduction. In Chapter 2 the literature in the complexity field is reviewed with a particular emphasis on complexity in information processing and ATC. Chapter 3 describes the methodology in detail. It begins with a discussion of typical ATC tasks and interfaces and then the ATC task which forms the basis of the validation of this methodology is analyzed by a CTA. The CIA functions are devised and presented in Chapter 3, and the method for estimating complexity is illustrated.

The experimental methods used to measure performance on the ATC task are described in Chapter 4 along with the associated experimental results. Chapter 5 compares the performance measures to the complexity estimates obtained from the methods described in Chapter 3. Finally the results are summarized and final conclusions are drawn in Chapter 6. This chapter also includes a discussion on potential limitations and shortcomings, as well as recommendations for future work.

2. Complexity

Complexity is cited extensively throughout the literature in a variety of fields and disciplines and affects many aspects of science, engineering and technology integration (Bar-Yam, 1999). Merriam-Webster dictionary defines complexity as *"composed of many interconnected parts; compound; composite: a complex system"* (2001). In this chapter relevant complexity literature is reviewed. The review is structured to go from a broad to more specific focus and is sectioned into the following components:

- Philosophical discourses on complexity
- Information theory and algorithmic complexity
- Complexity in cognition and human factors
- Complexity in air traffic control

The chapter discusses methods proposed to measure complexity, with a particular emphasis on previous work regarding measures of complexity in ATC and human-machine interaction. This chapter concludes by providing the rationale for the chosen complexity definition.

2.1. Philosophical Discourses on Complexity

The complexity review begins at the epistemologically broadest level with the discussion of complexity from a philosophical perspective. Many efforts have been made to provide comprehensive definitions of complexity. A noteworthy example is the work of Edmonds (1999a; 1999b) who conducted a detailed study on the syntactic origins of complexity. In this work he defines complexity as *"that property of a language expression which makes it difficult to formulate its overall behaviour even when given almost complete information about its atomic components and their inter-relations."* An important consequence of this definition is that it relays the difficulty in formalizing something complex based solely on the understanding of its fundamental parts.

Grassberger (1989) positions complexity somewhere between order and disorder. He considers three diagrams similar to those in Figure 2-1. The human generally judges the middle as the most complex. The given reason is that humans interpret the image on the right as an object with no rules. If describing the images at the level of each pixel however, the amount of information required to objectively and fully describe the rightmost figure is greater. Therefore rather than highlighting that complexity lies somewhere between order and disorder, this example emphasizes that complexity is relative to the language of the observer. More specifically complexity is relative to the language of description used by the observer to decode the object. The language of description is the set of characters, conventions and rules used to decode and convey information. By the language of description used by the human to perceive the figures, the description of the middle figure requires the most information. The human representation of the middle figure has the least pattern redundancy and has a larger description length than the other two.

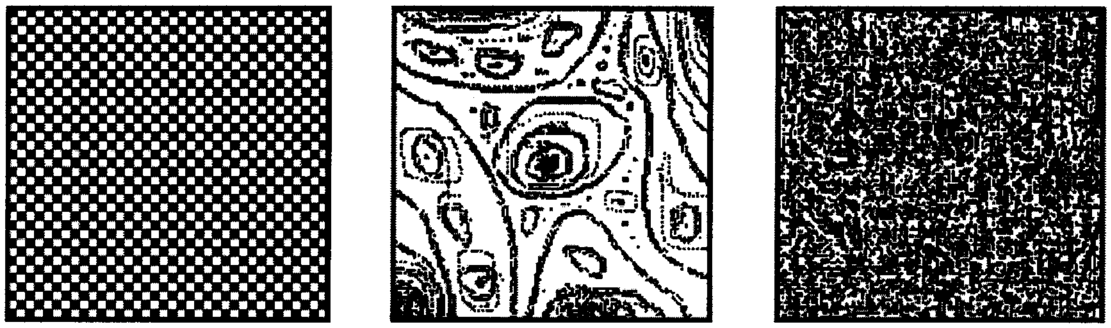


Figure 2-1. Left figure displays an ordered pattern. Middle figure displays chaotic pattern. Right figure displays a disordered pattern (white noise). Grassberger (1989).

In the extensive complexity review conducted by Xing & Manning (2005), they conclude that the concept of complexity is ultimately multi-dimensional and cannot be sufficiently described with a single measure. They identify three dimensions of complexity: *number*, *variety* and *rules*. *Number* refers to the numeric size of the elements in a system and *variety* refers to the diversity of elements within the system. *Rules* govern relationships between system elements. Their work was conducted for the purpose of reviewing techniques which could be applied to the complexity analysis of ATC displays. Further discussion is thus reserved for subsequent sections.

While philosophical discourses of complexity are of interest for conceptual understanding, they are inherently subject to linguistic ambiguity. This limits their direct applicability as a basis for a quantitative metric in engineering. This is less of a problem with the more mathematical based methods reviewed in the following section. However a most important point drawn from the philosophical discourse is that *complexity necessarily depends on the language that is used to model the system* (Edmonds, 1996). Therefore any measure of the complexity of human-machine interaction is dependant upon the language chosen to describe the interaction process. In the proposed methodology the description language is the cognitive interaction algorithm.

2.2. Information Theory and Algorithmic Complexity

Information theory originated in the late 19th Century when mathematical formulae were developed to measure entropy in thermodynamic processes (Gell-Mann & Lloyd, 1996). Thermodynamic entropy effectively measures uncertainty about the microscopic state of matter. Because information serves to reduce uncertainty, the mathematical formula proposed by Claude Shannon (1948) to define information was essentially the same as that which measured entropy[2]. Shannon's formula for the entropy of a discrete message space is shown below.

$$I(M) = -\sum_{m \in M} p(m)\log(p(m)) \qquad \text{(E.2-1)}$$

Equation 2-1 computes the information (I) contained in a set of messages (M), as a function of the probability (p) with which each message (m) in set M is communicated. The term $\log(p(m))$ is the self-information of message m. Prior to Shannon's contribution, information content of a message was oftentimes measured by the number of letters or other symbols needed to convey it.

Stemming from the seminal work of Shannon (1948), information theorists have provided measures of complexity, as they relate to communication and algorithms. Perhaps the most notable of such measures is that developed by Russian mathematician Andrei Kolmogorov and known as *Kolmogorov Complexity.* One definition defines Kolmogorov Complexity as "*the minimum number of bits into which a string can be*

[2] Shannon's equation differs by a multiplicative constant

compressed without losing information" (Black, 1999). In other words the complexity of a string is the length of the string's shortest description in some fixed description language. For a rigorous mathematical definition, and comprehensive resource on Kolmogorov Complexity, the reader is referred to the work by Li & Vitanyi (1997).

The principle of minimum description length (MDL) is related to Kolmogorov Complexity. MDL has its origins in the work of Shannon (1948; 1964) however Rissanen (1978) was the first to formally formulate the idea. The general idea behind MDL is that any regularity in a given set of data can be used to compress the data. The MDL principle is a formalization of Occam's Razor (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1987; Kaye & Martin, 2001) in which the best hypothesis for a given set of data is the one that leads to the largest compression of the data (i.e. the description with the lowest complexity). The vast majority of applications of MDL are for model reduction and statistical inference, and used within the computer science field. As MDL captures the minimum amount information required to describe a system it inherently measures information complexity. A larger MDL corresponds to a larger complexity. For a detailed review of the theory and current applications of MDL, the reader is referred to the recent work compiled by Grünwald, Myung & Pitt (2005). For simplicity in this thesis MDL (and thus complexity) is approximated by the compressed information content of an algorithm (the CIA).

The application of information theoretic formalisms to problems of human machine interaction has thus far had limited success. A notable exception includes Fitt's Law (Fitts, 1954), which applies information theory to provide a quantitative model for analyzing the trade-off between the accuracy and speed of human motor movements as a function of end point target size. The general failure to connect information theoretic formalisms to human-machine interaction is of course partly due to the sheer complexity of human information processing. In addition human information processing is generally not expressed in a formal language by human factors researchers, something which limits the applicability of more systematic and quantitative predictive methods. The application of information and algorithmic theoretic analysis techniques to human-machine interaction could be simplified if human information processing and machine interaction is expressed more formally as an algorithm. The CIA attempts to provide greater formalism to the description of the human information processing involved during human-machine interaction, such that quantitative analysis techniques can be applied.

2.3. Complexity in Cognition and Human Factors

The section begins with the general discussion of complexity in cognition and then continues with a discussion of complexity in human factors. The definition which best connects cognition and complexity is that by Godfrey-Smith (1996). He states that "*the function of cognition is to enable the agent to deal with environmental complexity*" (Godfrey-Smith, 1996). In this thesis the relevant *agent* is the human. The *environment* is formed by the machine interface and the information arriving through the interface.

Cognition is performed within the central nervous system (CNS) which has the brain at its core. On the cellular level the brain is composed of approximately 10^{11} neurons and an even greater number of glial cells. These cells form the basis of the neural circuitry of the central nervous system (CNS). The CNS is described by Pinker as a complex computational device that transforms information in rule-governed, algorithmic ways (Pinker, 1997). The specialized neural circuits contained within the CNS make up systems which are the basis of perception, action, and higher cognitive functions, including consciousness (Kandel, Schwartz, & Jessel, 2000; Pinker, 1997).

Xing (2004) relates neural science to complexity and provides examples of certain areas of the brain and their functions as they relate to a human interacting with an ATC interface. She outlines the functions of areas of the brain involved with information processing stages of *perception*, *cognition* and *action*. The visual cortex performs perceptual tasks that include target searching, text reading, color detection, motion discrimination and many others. Typically part of this information is then transmitted to the associational cortex and combined with elements of long term memory. Among others, the cognitive functions of working memory, text comprehension and planning occur in the associational cortex. According to Xing (2004) the bandwidth of the cognitive stage is much less than that of the perceptual stage. The pre-motor and motor cortical areas encode voluntary body movements, such as those of the eye or hands. The motor cortex, which enables the action phase of the information processing loop, is believed to process information in a serial manner, and consequently has a narrower bandwidth than both the perceptual and cognitive stages (Xing, 2004).

As alluded to in the previous paragraph, the neural networks in the human brain have limited bandwidths, as is the case with any communication medium. In other words, there are limits to how much information can pass through these neural circuits in a given amount of time. If these information limits are exceeded, information is lost and performance on a task likely drops. These limits in information throughput are equivalent to complexity limits. In order to process any incoming information signals, the human CNS must perform neural computations to reduce information complexity. Incoming information to the human, such as that from a display, which has a certain absolute complexity, acts in conjunction with a large number of neural schemas[3]. These neural schemas are essentially adaptive complexity transformers that serve to transform and reduce the incoming absolute state of information complexity to levels which the human is able to process effectively.

The simplified relationship between absolute complexity and complexity relative to the human is shown graphically in Figure 2-2. The equation describing this transformation is shown below.

$$\kappa_{human} = \mathrm{M}\kappa_{absolute} \tag{2-1}$$

In Equation 2-1 κ_{human} is the complexity relative to the human, M is the transformation and $\kappa_{absolute}$ is the absolute complexity. The matrix M is purely a property of the CNS and is constructed through learning and genetic preprogramming. The term κ_{human} is the complexity of the computations that occur in the CNS and is that component of complexity which this proposed methodology attempts to capture. The quantity κ_{human} must not be confused with perceived complexity. Perceived complexity is the conscious (subjective) component of κ_{human}. The term $\kappa_{absolute}$ contains all the information required to fully describe the environmental input signal. Equation 2-1 is shown mostly for

[3] It is important to note that the term *neural schema* is chosen because it is physically more accurate than the similarly used definition of *mental model*, originated by Craik (1943). For more information on neural schemas refer to Arbib (1992; 2003).

conceptual reasons, and it is not intended at this stage for the mathematical properties of the transformation to be defined.

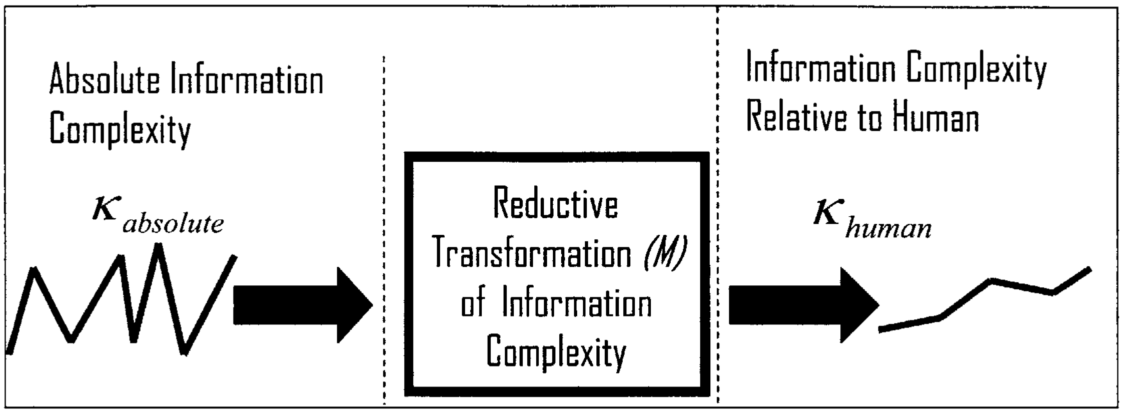

Absolute Information Complexity
$K_{absolute}$
Reductive Transformation *(M)* of Information Complexity
Information Complexity Relative to Human
K_{human}

Figure 2-2. Graphical representation of the transformation between absolute complexity and complexity relative to the human. The reduction is the result of a large number of neural schemas that have been programmed in the human central nervous system and correspond to a transformation represented by *M*.

In terms of complexity in human factors, Miller (2000) makes a distinction between *perceived* and *actual* complexity. He defines perceived complexity as the "*phenomenon of being difficult to understand or deal with.*" He further decomposes perceived complexity into three dimensions: *component complexity, relational complexity* and *behavioral complexity*. Component complexity is the number and diversity of components that the human perceives a system to have. Relational complexity refers to the number and variety of the links between components. Finally behavioral complexity refers to the number and variety of perceived states or behaviors that the system can exhibit. An increase along any of the dimensions results in an increase in complexity. According to Miller, increased complexity results in an increase in workload and/or unpredictability of the system. This motivates the need to quantify it in order to be able to compare the complexity of competing human-machine interfaces.

Miller's complexity breakdown is similar to the categorical breakdown of complexity developed by Xing & Manning (2005). As already mentioned they arrive at the conclusion that the key elements of complexity are *number*, *variety* and *rules* (Xing, 2004; Xing & Manning, 2005). The added point relevant to this subsection is that complexity is the result of crossing the elements of complexity with the three stages of human information processing identified by Xing (2004). These stages consist of perception, cognition and action.

Another reference that relates to this discussion is the review of visual complexity by Donderi (2006). As part of this extensive review Donderi first covers structuralism and Gestalt psychology and continues by reviewing the topics of visual complexity theory, perceptual learning theory and neural circuit theory. A goal of his review is to help establish the theoretical basis for measures of complexity in human factors. In his earlier work Donderi (2003) uses compressed computer image sizes for measuring the visual complexity of a display. While he does not directly propose a general measure of complexity in HMI, he concludes his review by stating that *"minimizing visual complexity, while respecting the requirements for the task, should maximize task performance."*

In summary, the human brain is conceptually a computational device which employs neural schemas to reduce complexity of the incoming information. This reduction of complexity is required so that the human can successfully interact with the surrounding environment. Minimizing complexity should maximize human performance on a task. In this thesis, all stages of CNS information processing will be categorized as *cognition*.

2.4. Complexity in Air Traffic Control

Air traffic control complexity is a critical issue, particularly in light of expected forecasts of increased air traffic volume. Complexity has been attributed as the primary cause for the deployment failure of the FAA's Advanced Automation System after an expenditure of $US 3-6 billion over a period of twelve years (Bar-Yam, 1999). This complexity can arise from a variety of sources, categorized by Cummings & Tsonis (2005) as environmental, organizational and display (or interface). Figure 2-3 shows the cascading complexity chain illustrating the decomposition of complexity as it applies to complex systems such as ATC.

In ATC, capacity is in great part limited by the cognitive complexity imposed on the individual controller. Extensive work has been conducted to quantify complexity in ATC. For example Mogford et al. (1995) present a review of air traffic control complexity where they identify methods for extracting complexity factors from controllers. In general the majority of ATC complexity has focused on quantifying the complexity imposed on the controller by the air traffic itself, and not the complexity associated with the interface or the entire interaction process.

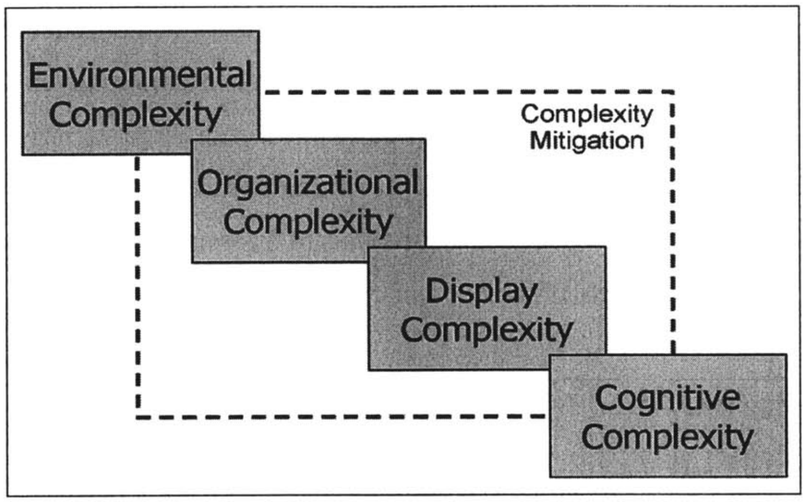


Figure 2-3. Air traffic control complexity chain (Cummings & Tsonis, 2005).

A significant portion of ATC complexity work has been the set of ATC complexity metrics known as *dynamic density* (Kopardekar, 2003; Laudeman, Shelden, Branstrom, & Brasil, 1998; Masalonis, Callaham, & Wanke, 2003; Sirdhar, Seth, & Grabbe, 1998). These measures attempt to capture, weigh and sum numerous driving variables of air traffic complexity to arrive at a numeric real-time metric of ATC complexity. These variables include such entities as number of aircraft, spacing distances, number of speed changes (Majumdar & Ochieng, 2002), and were selected because they were statistically deemed predictive of controller workload.

A smaller amount of work has focused on estimating the complexity imposed on the human controller due to the decision support interfaces intended to aid with the control and supervision of air traffic. As introduced previously, Xing & Manning (2005) cross the components of complexity (*number, variety,* and *relation)* with the three stages of human information processing (*perception*, *cognition*, and *action*). This generates a 3x3 matrix which forms the basic framework of their proposed ATC display complexity metric. The idea is that if general human limitations for each cell in the matrix could be established then these results could be used as a point of reference for evaluating the complexity of ATC interfaces. To date the question of how to establish the general limits and how to fill the cells of this matrix has not been addressed in the literature.

While decomposing complexity of an ATC interface into *number, variety and relation* is reasonable, there are more concise and precise ways to define and capture complexity. For example MDL[4] is able to precisely capture the consequences of all of the above three elements with greater parsimony. For example given two system descriptions with equal *number,* the system description with large *variety* is inherently less compressible than the element with less *variety*, and hence has a larger MDL. This is due to the smaller amount of repetitious and redundant information that is present. This is shown with the simple example of the two binary strings in Figure 2-4.

String (A)	101010101010101010101010
String (B)	110100011010011101010010

Figure 2-4. Two binary strings.

The top string A has a minimum description length in the English language of "twelve repetitions of 10". String B cannot be compressed to that degree, and would required one to express each digit. Hence by the MDL definition of complexity, String A is less complex than string B even though they have the same length. If strings A and B were abstract representations of controller, interface and traffic situations A and B respectively, it would be concluded that situation A is less complex than B. Any relations between elements would also necessitate additional descriptive information but could serve to either reduce or increase the MDL. Given two measures which capture complexity equally, the simplest is the most preferable.

The ability to capture complexity is important in ATC because ATC administrators require objective and quantitative methods to predict and assess human limitations and performance on new technology. A complexity metric would provide a more objective aid for deciding between options in the acquisition of new technology. Furthermore it would provide an objective "check" to subjective ratings that controllers

[4] Or information compression in general, which is an empirical surrogate

use to rate technological configurations. Objective complexity metrics would allow designers of technology to predict how competing interface configurations could affect the end user, while reducing the amount of more costly usability and evaluation trials.

2.5. Conclusions

After reviewing numerous definitions of complexity, the revised definition developed for the context of measuring complexity in human machine interaction is:

Complexity is defined as the (minimum) amount of information required to describe the human-machine interaction process in some fixed description language and chosen level of detail.

In this work the proposed *description language* is the CIA. The CIA represents human information processing and includes the actions involved in the execution of a task using a given machine interface. The above definition is consistent with complexity in information theory and similar to *effective complexity* defined by Gell-Mann & Lloyd (1996). The chosen definition also parallels the general definition of Bar-Yam who defines complexity as the amount of information necessary to describe a system[5] (Bar-Yam, 1997). The selected definition also parsimoniously captures the consequences of each of the complexity elements of *number*, *variety* and *rules*, or any similar such categorical breakdown[6]. A final advantage of the selected definition of complexity is its simplicity, which makes it easier for it to form the basis of a quantitative metric. Complexity (as defined here), is estimated by the information content of an algorithm (CIA) for different task and interface variants. As is discussed in greater detail in the subsequent sections, in this thesis information is quantified by two measures: the algorithmic information content (AIC) and the compressed algorithm size which is representative of MDL.

[5] He also notes that this is dependent upon the level of detail of the system description

[6] The premise is that the variety and rules require information to be described.

Page intentionally left blank

3. Methodology

The purpose of this chapter is the presentation and detailed discussion of the complexity analysis methodology first outlined in Chapter 1 (Figure 3-1). The analysis is applied to an experimental ATC interface and task that was developed for the purposes of examining performance consequences of different interface variants.

The methodological steps in the proposed complexity estimation method are shown in Figure 3-1. Once competing interface configurations have been identified, the first step in the procedure consists of a CTA of the representative ATC task. The motivation for conducting the CTA is that it provides knowledge regarding operator information processing. This knowledge forms the basis for the CIA functions which are formulated relative to an assumed representative operator.

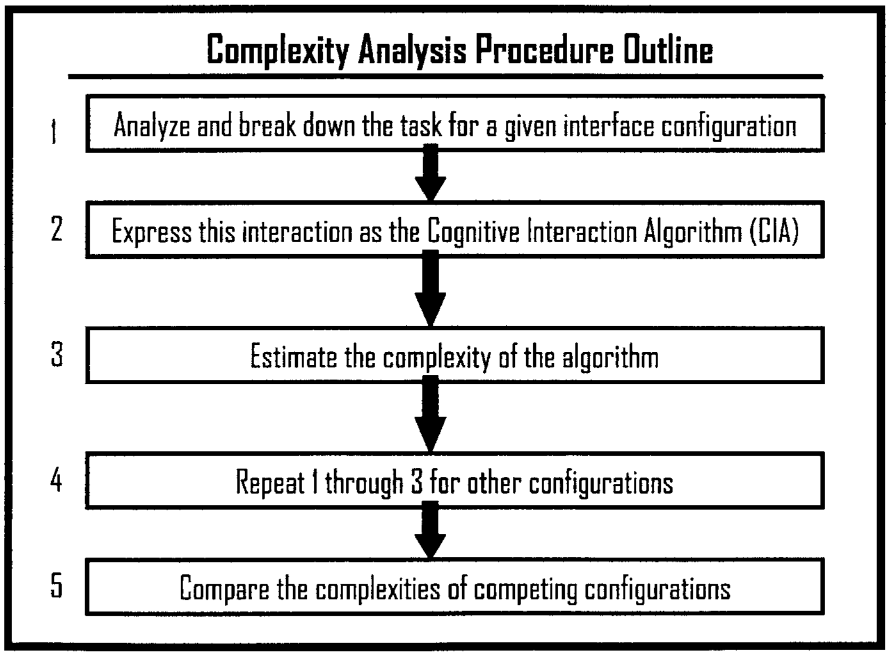

Complexity Analysis Procedure Outline
1 Analyze and break down the task for a given interface configuration
2 Express this interaction as the Cognitive Interaction Algorithm (CIA)
3 Estimate the complexity of the algorithm
4 Repeat 1 through 3 for other configurations
5 Compare the complexities of competing configurations

Figure 3-1. Outline of complexity analysis procedure

The second step consists of writing the CIA functions for the various interaction processes. The ultimate purpose for expressing the interaction process as an algorithm (the CIA) is such that the information complexity can be quantified. In order to fulfill this process, a core CIA is created for each interface configuration and the nominal user interaction. The core CIA is the highest level script of what typical occurs in each assumed scenario and are the algorithms from which all CIA functions are called. The CIA functions represent cognitive and task subprocesses that the operator performs (eg: moving the cursor). In Step 3, the method and equation for estimating overall complexity from the algorithms, is applied to example interface variants for an assumed nominal task. While the purpose of this chapter is to outline the methodology, Chapters 4 and 5 present additional task factors and validate the predictive capacity of the complexity measures (for the given interfaces and tasks).

3.1. Interface Configurations

This complexity estimation methodology is applied in order to quantify and compare complexity between different interface configurations. It is important to emphasize that interface configurations depend upon the context under which they are used and thus cannot be compared independent of a task. For example the complexity of a computer keyboard depends upon the message being typed and not only on the keyboard itself. It is therefore necessary to establish a nominal scenario which describes how the human interacts with the interface. Only within the context of an assumed scenario can the complexity of the different interface configurations be correctly assessed.

As an example illustration of the complexity estimation methodology, four variants of en-route ATC aircraft data-blocks are compared. Data-blocks are interface elements on the display which contain alphanumeric and graphical information pertinent to each target aircraft. They are located beside the actual aircraft position on the display, and controllers extract information from these in order to perform their tasks. This information typically includes such information as altitude, speed, heading, and flight and identification numbers. A screenshot of a typical data-block from an actual en-route ATC display is shown in Figure 3-2.

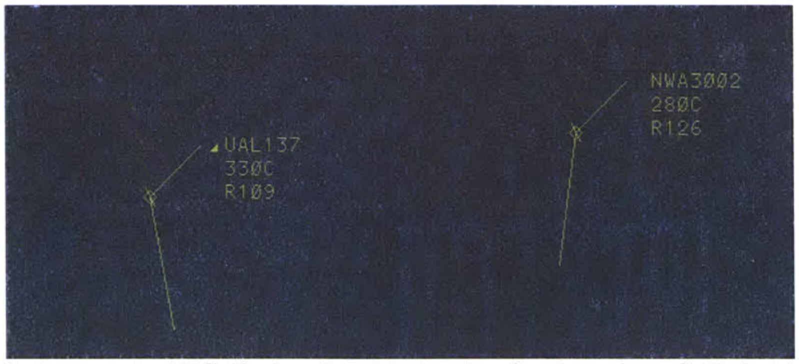


Figure 3-2. Typical en-route ATC display data-blocks showing two aircraft. Data-blocks have three lines of information. The first line of text represents the flight number, the second is the altitude and the third is the computer identification number. Image courtesy of the FAA.

The four example data-block variants, to which the complexity estimation methodology is applied, are shown in Figure 3-3 and vary by the amount of information that each has on the base layer. The data-blocks with fewer than five lines on the base layer are expandable to display all five lines. In the cases with fewer than five lines visible, the rest are made visible by clicking on a double arrowhead. Each click displays one extra line.

From an application perspective, an ATC organization such as the FAA would want to identify any potential consequences of changing the number of lines on the performance of controllers. This is especially relevant as more information becomes available with an increased number of sensors and higher communication bandwidths and decisions must be made on how and where to present that additional information.

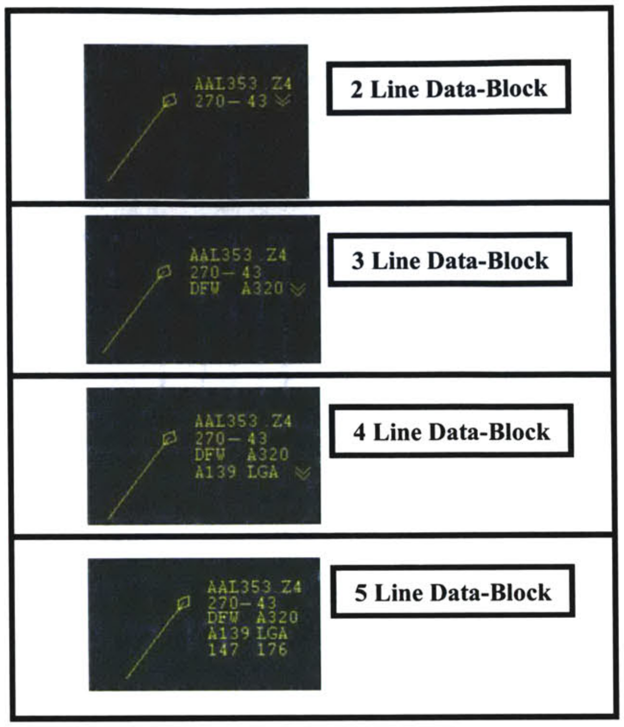


Figure 3-3. Four competing interface variants. These consist of four aircraft data-block types with varying number of lines on the base layer. The first three data-blocks can be expanded to include all five lines of information. The bottommost data-block always displays five lines.

The simulated interface and associated representative tasks are discussed in depth within the following section as part of the CTA.

3.2. Step One: Cognitive Task Analysis

The purpose of the CTA in this complexity estimation methodology is to elicit and present a high level approximation of the information processing that occurs within the human CNS as an ATC task is carried out using a given interface. Since this methodology is intended for applications to problems of human-machine interaction in ATC, a task-analytic overview of actual ATC operations is provided and framed within a historical context. Existing CTA techniques are then briefly reviewed in order to establish an understanding of available methods and the techniques applied. Following this, a representative experimental ATC task and interface is described. The final CTA step is the presentation of the resulting cognitive process charts and a description of the steps and procedures involved in carrying out the ATC task.

3.2.1. Air traffic control background, tasks and interfaces

The safe and expeditious movement of air traffic requires the coordination of an enormous number of human, technological and economic resources. The infrastructure and capability to efficiently and safely move people and goods by air is one most valuable economic assets of a nation. The Air Traffic Action Group reported in 2005 that the air transportation industry contributes 880 billion U.S. dollars annually to the cumulative global gross domestic product (The economic and social benefits of air transport, 2005). Central to ATC operations are air traffic controllers who are responsible for separating and directing aircraft to their destinations. The tasks that controllers perform have changed over the past eighty years, driven by several technopolitical[7] milestones and a steady surge in commercial air traffic (Burkhardt, 1967; Heppenheimer, 1995).

In the early years of ATC, controllers manually tracked aircraft on maps using small paperweights (called "shrimp boats"), as shown in Figure 3-4. These tools provided one of the first ATC interfaces, and subsequent technological evolution and implementation was aimed at eliminating the limitations imposed by these tools. Although at that time controllers had no direct radio link with aircraft, they used telephones to stay in touch with airline dispatchers, airway radio operators, and airport traffic controllers. These individuals fed information to the en-route controllers and also relayed their instructions to pilots.

[7] Technopolitical milestones include such things as key congressional acts (e.g.: Air Commerce Act) and enabling technologies (e.g.: radar), which drove the establishment of the current ATC system

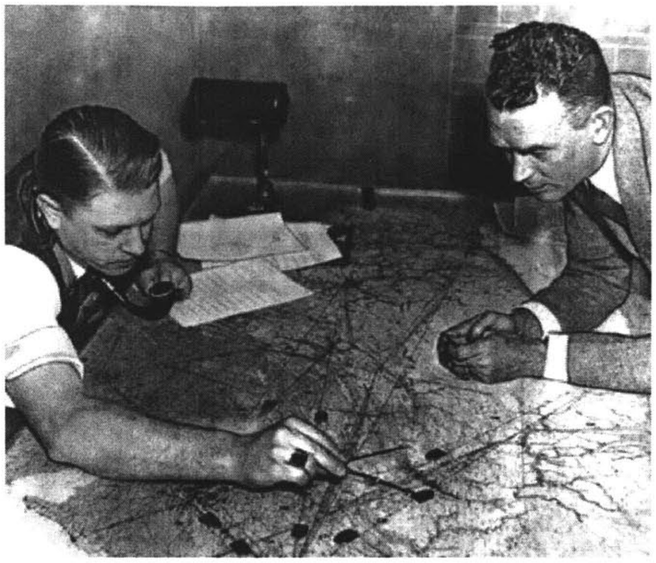


Figure 3-4.Air traffic controllers circa 1935 tracking aircraft using small paperweights on a table top aviation chart. Image courtesy of the FAA.

The radar[8] which was first introduced in 1952 in the United States was the technology that enabled significant increases in controlled traffic capacity (Nolan, 2004; Skolnik, 2001), but also added a new layer of complexity. It was the first technology that presented controllers with a relatively accurate approximation of the aircraft's position at a given time. Although greatly improved over the past 50 years, this technology still forms the foundation of today's air traffic management systems. Information from the radar is presented to the human via the scope, which is the primary tool used by controllers to support their tasks. Radar scopes have evolved from their early days (Figure 3-5), and now include automation aids and tools which the controller can leverage. Current ATC interfaces in the United States resemble that shown in the introduction (Figure 1-1). Many relics of previous systems are carried over to new systems however. One such example is the display of radar sweeps on screen, something not necessary with modern displays. Resistance to change is partly because of the potentially high risk of radical changes in such a safety critical domain. It is also due in part to controller resistance to technological implementations which could replace their own skills.

[8] The word radar originated as an acronym (RADAR) which stands for Radio Detection and Ranging

Figure 3-5. Air traffic controller interacting with a radar scope circa 1960. Image courtesy of the FAA.

Current ATC operations are typically broken down into several phases. The first phase begins with the aircraft on a taxiway. A subset of controllers, typically located in a control tower, are responsible for surface traffic, ensure that aircraft taxi to their runways in an orderly fashion and do not progress onto runways unless provided with a clearance. These controllers issue clearances for take-off. Once the aircraft is airborne, a terminal control center soon assumes responsibility for that flight, vectoring the aircraft away from the airport safely. Once the aircraft has climbed and is at some distance from the terminal area, responsibility for the flight is handed off to en-route controllers who monitor the high altitude traffic flows and issue commands to pilots as required. As a flight nears its destination airport and descends, responsibility is handed back to the terminal controllers who ensure a clear approach. Finally control is yielded back to the tower who clears the runway and return taxi. A profile of the flight phases is shown in Figure 3-6.

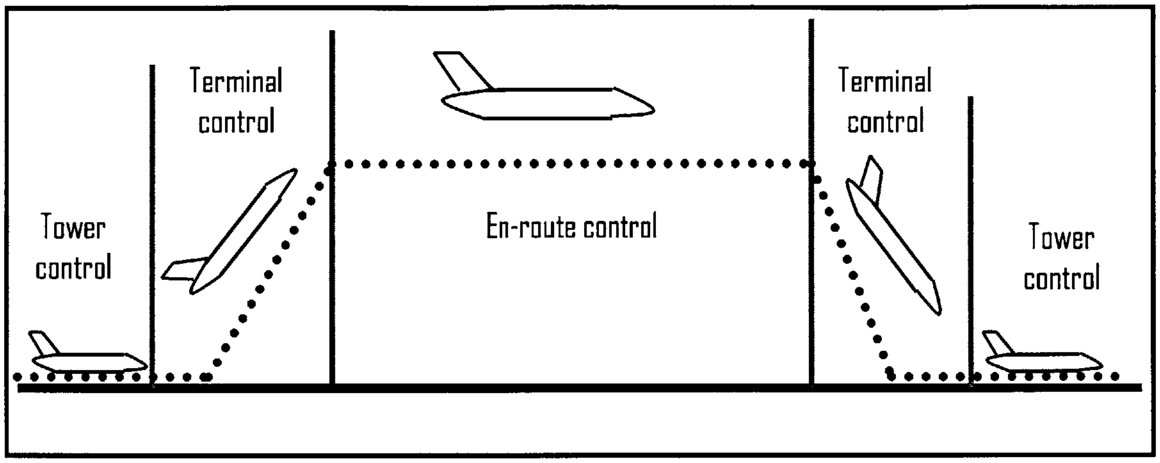


Figure 3-6. Air traffic control flight phases.

The representative task and interface to which this complexity analysis methodology is applied replicates en-route ATC characteristics. Typical en-route ATC operations consist of directing and ensuring separation of aircraft flying at high altitudes. In the USA, the national airspace is sectioned into twenty-two air route traffic control centers (ARTCC), each broken down into several dozen sectors. One controller is generally responsible for a single sector, although in many cases is assisted by an additional person. Controllers accept aircraft from neighboring en-route sectors and then must hand them off once they have traversed their sector. As aircraft fly through the sector, controllers must ensure adherence to a set of separation standards that define the minimum distance allowed between aircraft. Furthermore controllers attempt to expedite the flow of traffic such that airlines can minimize flight time and fuel consumption.

In the USA the most recent en-route control interface implemented is the Display System Replacement (DSR), manufactured by Lockheed Martin. The DSR station is shown in Chapter 1 (Figure 1-1) and includes

- A primary 20"x 20" color display (radar) with trackball and keyboard
- A 15" secondary color display (data) with flight strip bays and keyboard input
- An auxiliary 15" display with flight strip bays, printer and keyboard input

A screenshot of the primary display is shown in Figure 3-7. The experimental interface developed for this research was modeled on this system.

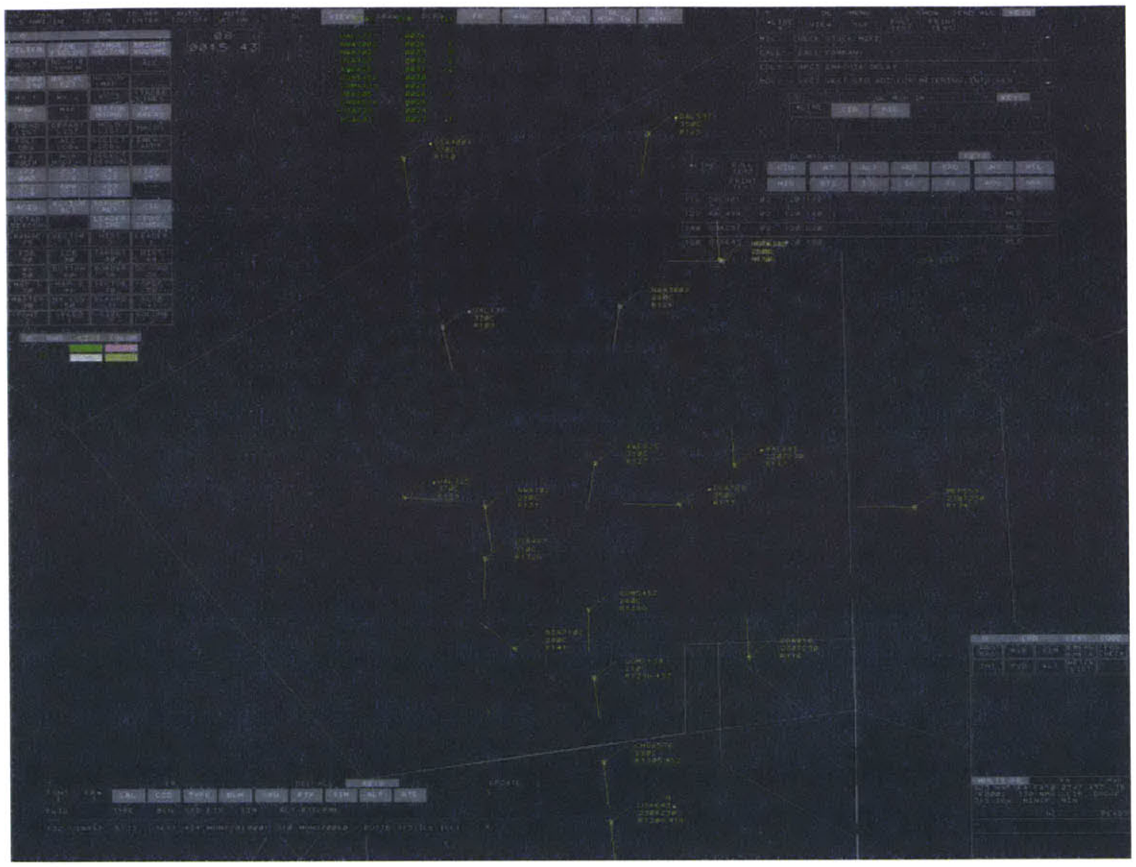


Figure 3-7. En-route DSR primary display interface screenshot. Image courtesy of the FAA.

Air traffic controllers have skills and knowledge that are the result of extensive training and practice. Their preparation includes a four year FAA approved education program, twelve weeks of operational training, followed by two to four years of practical training within an ATC facility (Bureau of Labor Statistics, 2006). Reducing the complexity of ATC interfaces and tasks has the added benefit of reducing this long preparation time.

For ATC tasks controllers generally command inputs via a trackball and keyboard, observe visual information from the primary (radar scope) and secondary displays and communicate with aircraft via a headset which they must key in order for their communication to be aired. Controllers must make both spatial and temporal mental projections based on multiple aircraft and follow numerous standardized procedures and regulations. They track targets and comprehend and project the location and behavior of aircraft in the three spatial dimensions in addition to the dimension of time. In most circumstances they also communicate with controllers who support them, as well as supervisors and controllers of other sectors. In performing these tasks and interacting with these technologies and other humans, controllers process information using neural schemas and mental computations that result from years of training. The information processing of controllers has been the subject of a large body of research (e.g., Majumdar & Ochieng, 2002; Mogford, 1997; Seamster, Redding, Cannon, Ryder, & Purcell, 1993). The following section discusses the CTA methods which were applied to the representative ATC task.

3.2.2. Cognitive task analysis methods

Cognitive task analysis (CTA) is an umbrella term encompassing a variety of methods and techniques that serve to break down and extract the cognitive processes and physical actions which a human performs in the accomplishment of a set of goals and tasks. Cognitive task analyses are practical tools that have the general purpose of methodically breaking down a human-machine interaction process into numerous interconnected subtasks.

A variety of CTA techniques have been developed, applied and reported in the human factors research and development community (Annett & Stanton, 2000; Diaper & Stanton, 2004; Hackos, 1998; Schraagen, Chipman, & Shalin, 2000). CTA techniques include numerous methods for extracting information, including observation, interviews and surveys as well as examination of verbal communications between team members, think aloud methods, diagramming methods and psychological scaling methods (Seamster, Redding, & Kaempf, 1997).

Several methods have been applied to problems of human machine interaction in the aviation community, including ATC (Seamster et al., 1997). For example Seamster et al. (1993) conducted an extensive CTA study to specify the instructional content and sequencing for the FAA's en-route ATC curriculum redesign. This analysis attempted to capture knowledge structures, mental models, skills and strategies of en-route controllers in order to provide an understanding of the key components of the controller's job. As a result, Seamster and his colleagues identified thirteen primary tasks, presented a mental model of the controllers' knowledge organization, categorized controller strategies, and expressed controller goals hierarchically.

In this research the ultimate purpose of the CTA is to extract information regarding the cognitive processes of controllers in order to develop a more accurate CIA. The CTA portion conducted for this complexity analysis consisted of the following components:

- observation of operators
- recording of operators verbalizing their thought process
- informal interviews with selected operators
- detailed description of the task, interface and interaction processes
- hierarchical task decomposition and listing of information requirements
- generation of cognitive process flow charts
- description of cognitive process flow charts

The following subsections discuss the methods applied.

3.2.2.1. Observation

Observation is a fundamental component of a CTA, although it is perhaps the most difficult to formally express. In the CTA applied in this research, observations consisted of two main components. The first portion consisted of observations and discussions during scenario validation, carried out with pre-trial participants. This included a presentation of the proposed interface to a group of research colleagues, who provided feedback on the interface. The next portion was the observation of two French ATC trainees who were used to primarily elicit the high level cognitive processes involved with the direction of aircraft, avoidance of conflicts, and the prioritization of tasks. The two participant controllers were males in their mid-twenties, had completed most of their ATC training, and were gaining operational experience. They first followed a tutorial which explained the representative task and interface usage. This lasted approximately fifteen minutes and was followed by a six minute practice scenario that familiarized the controllers with the interface and task. The controllers alternated in carrying out this practice scenario as well as the subsequent two scenarios each. This was done in order to have the added benefit of having one controller observe the other and provide the additional perspective, thus maximizing the session's output. This also stimulated greater post-scenario discussion between the two participants and the experimenter, regarding the task and interface used to accomplish the task.

3.2.2.2. Think aloud method

A think aloud method was used with the French ATC trainees in order to gather information on which subtasks were being attended to, the approximate duration of the attention, and the prioritization of tasks. As a core component of the CTA, the French ATC trainees were asked to verbalize their thought and action processes as they interacted with test bed scenarios. These participants were instructed to speak out loud while executing their tasks and then to comment both freely and in response to specific questions after each scenario. Each controller's computer display was fed to a secondary flat panel display which was recorded using a Sony Hi-8 VHS camcorder, which also captured the audio discussions during and after each of the scenarios. After each scenario these participants discussed their usage of the interface, the task, and their cognitive processes. Post-trial unstructured interviews also took place in order to elaborate on certain elements of their information processing, interface use and task performance. For example, the controllers were asked to elaborate on what determined which aircraft they decided to command.

3.2.2.3. Task decomposition and information requirements

The task breakdown decomposes tasks in a hierarchical fashion and identifies information requirements. One of the most well established CTA approaches is the hierarchical task analysis (HTA) which has its origins in the 1960's (Annett, 2004). HTA defines complex tasks as a hierarchy of goals and sub-goals. The breakdown of the tasks conducted for this research and presented in tabular format is analogous to an HTA. This is important for the eventual development of the CIA as it provides a basis for the variables used within the various CIA functions (e.g.: cursor position, target flight number, etc.).

3.2.2.4. Cognitive process charting

Cognitive process charting has the purpose of expressing the various subtasks and the connections between subtasks graphically in addition to capturing operator decisions. This step in the CTA procedure is a simplified parallel to Goals, Operators, Methods, Selection Rules (GOMS), which is among the most complete CTA tools in the Human Factors field. From its origin (Card, Moran, & Newell, 1983), different variants of GOMS have been developed over the years, with varying degrees of complexity (John & Kieras, 1996a; 1996b). GOMS consists of descriptions of the methods needed to accomplish specified goals (Kieras, 2004). Goals represent a hierarchy of what a user is trying to achieve. This is analogous to the task breakdown conducted in this research. Operators are the set of operations which a user composes in order to achieve a given solution. Methods represent sequences of Operators. The Methods are grouped together in order to accomplish a single Goal. Finally Selection Rules are used to select a solution method amongst several possibilities. In this research the simple analog to Operators, Methods and Selection Rules are the cognitive process charts which are developed. These charts capture the steps, decisions and processes which an operator typically carries out when performing tasks using the given interface.

3.2.2.5. Subtask timing

As a final part of the CTA, approximations of the time spent on several of the different subtasks identified by the task breakdown were recorded. The desire was that each of the resulting CIA functions had an associated estimated (human) execution time. For example, the algorithm of the function describing a human finding a specific aircraft on screen would have the associated time approximation for performing that subtask. This was initially motivated in order to form a basis for calculating total CIA execution time for each task and interface configuration[9]. This essentially corresponds to the time required for a CIA to execute when "run" on a typical human controller.

In order to capture subtask times, a simple stop watch graphical user interface was programmed in Matlab® and run alongside the ATC simulation. The estimations were

[9] This information could have been used to generate an additional metric of algorithmic execution time in addition to one consisting of CIA complexity divided by CIA execution time. The validity of these additional two measures could have then also been determined.

carried out with a single participant over multiple trials. The participant started the timer, carried out specific sections of the tasks as instructed, and then stopped the timer. This allowed several important subtasks and mental sub-processes to have an associated time dimension. Table 3-1 summarizes the timed subtasks. The first measure is the time required to access a piece of information within the data-block. This was measured for each data-block and each data-block line. The time captured the span from when the participant fixated upon the given data-block, until the time when the information was read and the timer stopped. The second metric captures the time taken to locate a given flight on screen. The timer was started at the time of the request and halted when the aircraft was fixated upon. These measurements are categorized based on data-block type and the number of aircraft on-screen (one of the experimental variables discussed in Chapter 4). The time to issue an aircraft command type was also timed for each command type. This interval captured the time from which the aircraft was clicked until the time the command was issued. Finally the average time to type and submit a response to a question was also timed. The results obtained are included in Section 3.2.3.

Table 3-1. Subtask time measurements and categories

	Measurement Categories		
Time to extract information from data-block	Data-block type	Data-block line	
Time to locate a flight number on screen	Data-block type	Number of a/c	
Time to issue a specific command type	Altitude	Speed	Heading
Time to type and submit question response	N/A		

3.2.3. Cognitive task analysis results

This section presents the results from the CTA methods applied. The results of the CTA form a resource from which to draw information to construct the CIA. The first part of the CTA results is a detailed description of the experimental task and interface. This is followed by the table summarizing the subtasks and mental sub-processes involved with the pursuit of this task, including information requirements. Flow charts of the subtasks and sub-processes are then presented. These allow the reader to visualize the hypothesized information processing involved in the interaction.

3.2.3.1. Description of task and interface

The representative task developed, which includes the interface configurations shown in Figure 3-3 replicated a typical en-route ATC task and interface similar to that described earlier in the chapter. The software was programmed in Matlab®. The interface layout is shown and labeled in Figure 3-8. On the bottom left hand side of the display shown in Figure 3-8 is the data-link interface where secondary task questions (from a hypothetical traffic supervisor) appear. Operators submit their answers by typing them in the associated edit box and then clicking the submit button using the mouse. The alert confirmation button, located on the bottom center, should be clicked when the operators first notice an alert appear besides an aircraft data-block. The aircraft command panel, in the lower right portion of the screen, appears when an aircraft is selected and is used to issue altitude, speed or heading commands. It is important to note that each of these three command types has to be issued independently, requiring the controller to reselect an aircraft each time. Selected aircraft are displayed as green on the display (RGB[10] vector of [0.5 0.7 0.4]) and the unselected aircraft are yellow (RGB vector of [0.8 0.8 0.2]).

The interface and task are simplified so participants who were not expert controllers could be trained to use the experimental interface in a matter of minutes. Among several important differences is that communication between aircraft and controllers is via data-link and not radio (voice). Controllers click on an aircraft and then issue commands using the mouse and keyboard through a displayed panel. Procedures, rules and regulations are simplified and there are no secondary automation aids, such as planning or conflict alerting systems. Furthermore there is no use of flight strips.

[10] RGB stands for *red green blue*. The components of each RGB vector are the fractions of each color.

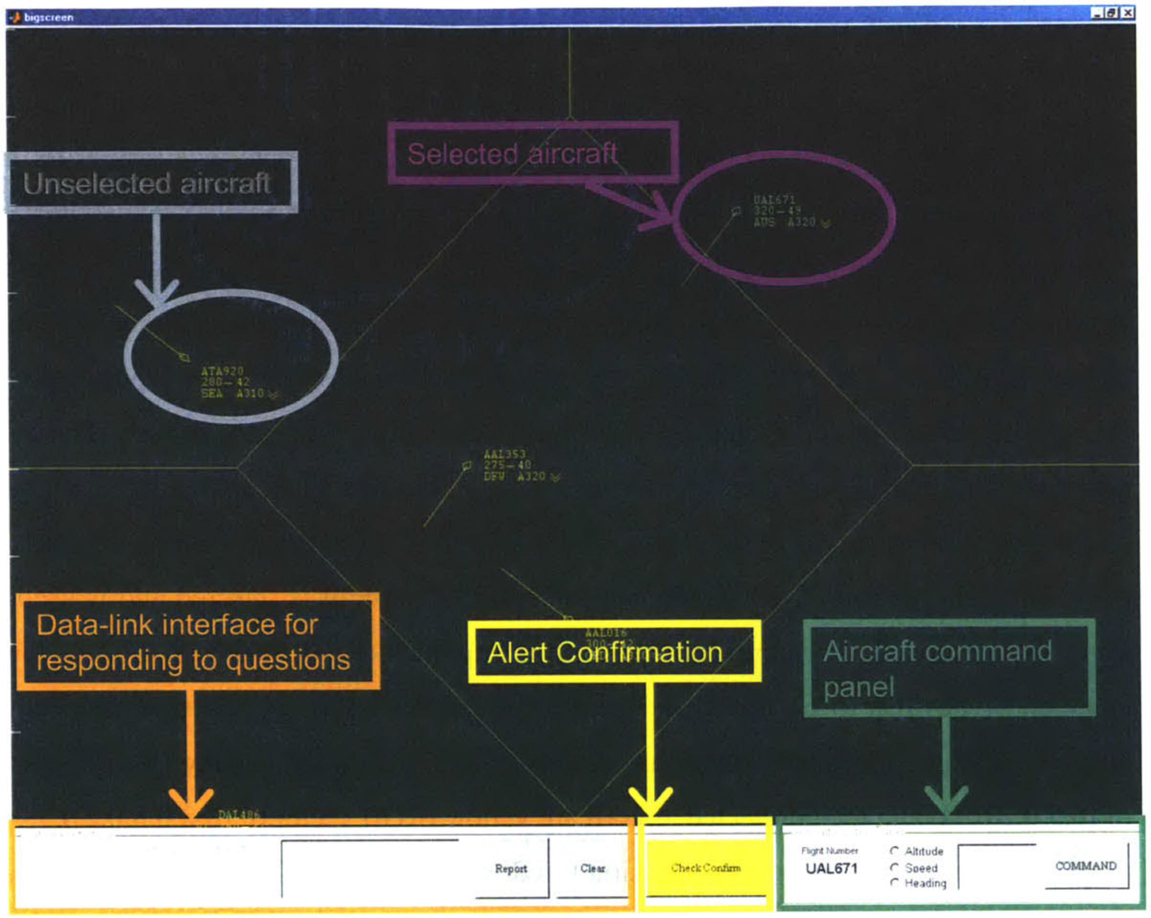


Figure 3-8. The Reprogrammable Air Traffic Experimental Testbed (RATE) interface.

Throughout each scenario there is a steady flow of aircraft into and out of the airspace. Participant controllers are tasked with vectoring aircraft, and have to do so at a specific egress altitudes and velocities. Operators are provided with a chart containing a map of the sector with the exit specifications. Each entering aircraft has four possible departure locations from which to egress. Furthermore participants are tasked with maintaining aircraft separation of 1,000 feet vertically and three nautical miles laterally. This is not a severe challenge as crossing aircraft are generally staggered. A secondary task consists of responding to numerous requests for flight information through the data-link. These requests are prompted by an aural alert and a green text message. The information required to answer these requests is available within the aircraft data-blocks. Another secondary task consists of detecting and confirming visual alerts that periodically appear beside an aircraft data-block.

The four aircraft data-block variants, shown previously in Figure 3-3, include a total of five lines of information pertinent to each flight, as labeled in Figure 3-9. While the data-block on the right displays only two lines, it too is expandable to display all five. It is from these five lines that participants perceive and extract the information required to perform their tasks. Within these lines, eleven individual pieces of information are contained. The top line of the data-block contains the flight number and the exit assigned to each aircraft (Z1 through Z4). The second line contains the flight level, a climb indicator arrow, and the aircraft speed. The third line contains the aircraft's final destination and the aircraft type. The fourth line contains the computer identification (CID) number and the flight's origin. Finally, the last line contains counts of the passengers and baggage aboard.

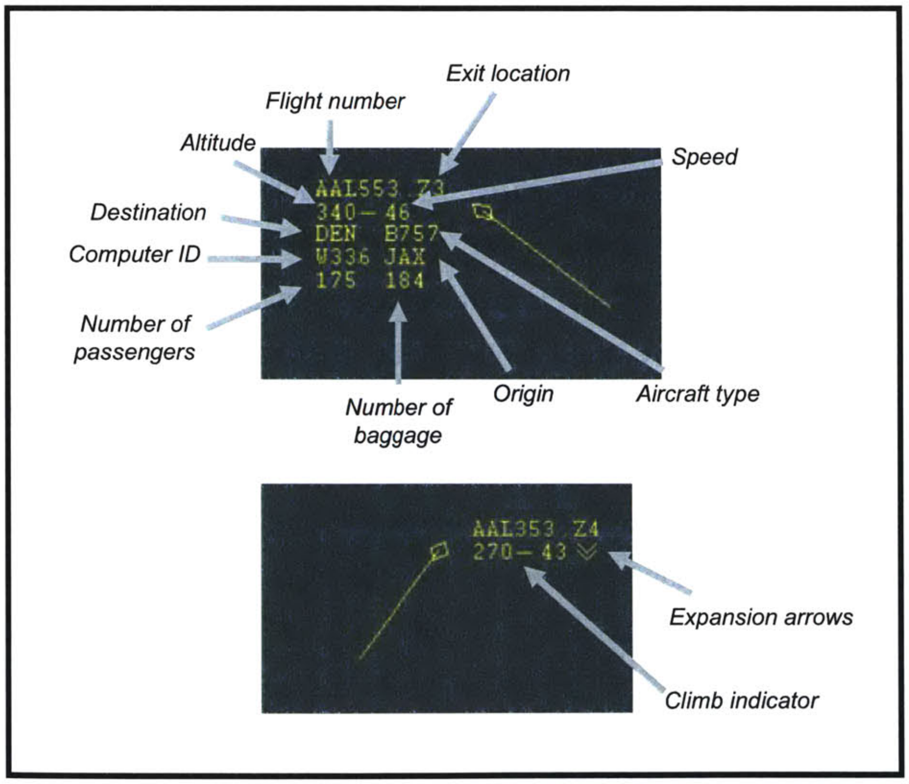


Figure 3-9. Five line (top) and two line (bottom) aircraft data-blocks. The double arrowhead used to expand the data-block is shown on the right. The information included in each data-block is labeled.

During each scenario, when aircraft become visible on screen, operators begin comprehending the situation in order to formulate a set of priorities and a given course of initial action. Information processing resources required at this step include the visual perceptual processing of the displayed information and the associated eye movements and fixations required to scan the display. Furthermore this portion of the task requires parallel access to long-term memory in order to use information to decode the meaning of the perceived elements. Information regarding the purpose of the overarching goals and tasks is also extracted from long term memory. Relevant information is processed in short-term memory, and from it a certain level of comprehension arises, along with a plan of initial action. This initial process is a function of the number of aircraft and information processing will take longer with more aircraft on the screen. Once operators identify an aircraft in need of vectoring, they click on it. The control panel appears and they move their cursor towards it and select one of the three radio buttons (for altitude, speed or heading). When this is clicked, the current value of the property for that particular flight appears within the text box. They then have to click within the text box, erase all or part of the original text and type the value to be commanded. If operators do not know the exit specifications they check the exit chart. They then have to use the mouse to submit the command, at which point the panel disappears and the aircraft begins its maneuver. If the aircraft needs to change altitude or speed, operators have to click on the aircraft and repeat the process. Operators then generally scan the display and switch focus to another aircraft and issue the necessary commands.

At some point they are interrupted by an aural alert signaling that a new information request has appeared. At this point they either switch focus almost immediately (dropping other tasks) in order to read the question or they finish whatever vectoring subtask they were attending before proceeding to respond. They then read the question, search for the aircraft for which the information is requested, and once found, search within the data-block of that aircraft for the specific piece of information. Operators would generally notice and report the appearance of an alert beside a data-block while scanning the display and not actively engaged in a specific ATC subtask. Figure 3-10 shows a high level flow chart of the basic ATC tasks discussed in this section. The following section provides a tabulated hierarchical breakdown of the tasks and the associated information requirements.

3.2.3.2. Decomposition of task and information requirements

The second result of the CTA is a tabulation of subtasks including information requirements. Decomposing the tasks helps understand important processes that should be expressed as CIA functions. Knowledge of the information requirements provides information regarding what variables the CIA functions need and call upon. Appendix A includes the entire tabulated breakdown of all tasks. A small portion of the table is shown in Table 3-2. The portion shown includes part of the breakdown of aircraft direction task. Each subtask is described and the information requirements for the given subtask are listed. The tasks in the table are each numbered. The representative task can essentially be broken down into four components. Tasks beginning with the digit 1 are tasks concerned with the direction of aircraft. Tasks beginning with the digit 2 are concerned with the separation of aircraft. Tasks beginning with the digit 3 are concerned with the response to questions. Finally tasks beginning with the digit 4 are concerned with the alert responses. Sub-tasks are broken down with additional digits into more specific parts.

Apart from forming a basis for CIA functions and variables, the decomposition of the task provides useful preliminary information for producing the cognitive process charts presented in the next section. The cognitive process charts add value to the tabular decomposition by showing how the subtasks are connected, and what decisions are made throughout the HMI process.

Table 3-2. Portion of table containing a hierarchical task breakdown and information requirements. Entire six page table is included in Appendix A.

No.	Subtask	Description	Information requirements
1	Direct aircraft	Direct aircraft that appear on screen to the required egress and ensure exits at the correct velocity and altitude for the given egress.	Flight number, actual altitude, actual velocity, actual heading, desired altitude, desired velocity, desired heading, selected aircraft, egress specifications, control panel state, cursor position, aircraft/subtask priority, overall objectives.
1.1*	Scan display	Scan eyes about the display in order to perceive information required for comprehension of the situation. Isolate an aircraft that must be directed.	Positions of aircraft, status of aircraft, flight numbers, egress locations and information
1.1.1	Choose aircraft	Mentally isolate an aircraft to direct based on a certain priority (eg: which is closest to the egress)	Position of selected aircraft, flight number of selected aircraft
1.1.1.1	Compute distance to exit line	Capture the distance of the aircraft to exit and hold for comparison	Position of selected aircraft, position of nearest exit line
1.1.2	Perceive destination	Perceive the egress destination of a certain aircraft	Egress information (Z1 - Z4), aircraft position
1.1.2.1	Check specific exit requirements	Check the exit requirements for a given exit from the chart	
1.2	Issue Command	Issue a command to a given aircraft	Aircraft, desired aircraft state, current aircraft state
1.2.1	Heading command	Change the heading of a given aircraft	Aircraft position, desired aircraft direction, actual aircraft direction
1.2.1.1	Click on aircraft	Select aircraft to which to command will be issued by clicking on the graphical symbol	Aircraft, cursor position, click/selection feedback

3.2.3.3. Cognitive process flow charts

The cognitive process charts capture the interaction process between the human and machine interface, show how tasks are connected, and what decisions are made. The execution of the task is broken down into a number of smaller charts (or modules). The overall and highest level module for the interaction process is shown in Figure 3-10. A description of this follows.

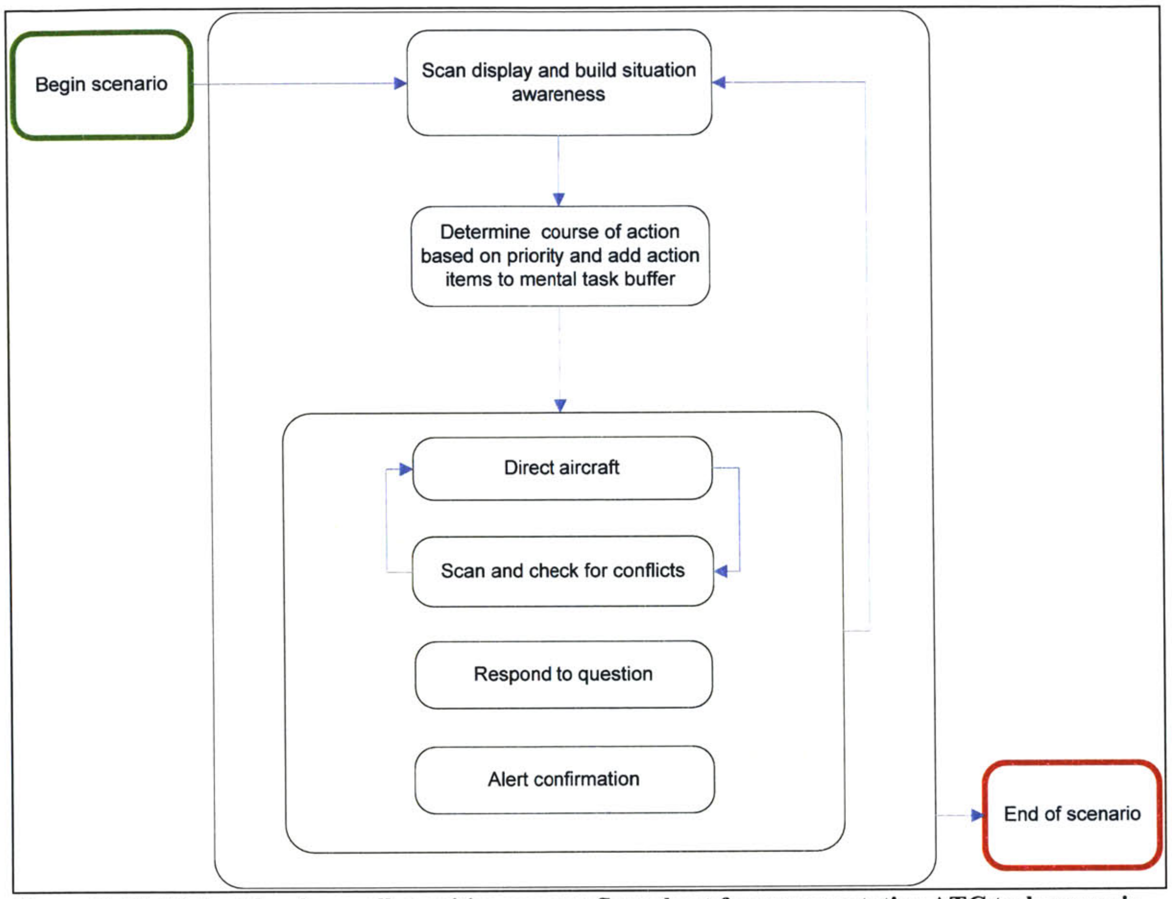

Begin scenario
Scan display and build situation awareness
Determine course of action based on priority and add action items to mental task buffer
Direct aircraft
Scan and check for conflicts
Respond to question
Alert confirmation
End of scenario

Figure 3-10. Highest level overall cognitive process flow chart for representative ATC task scenario

Once a scenario begins, the controllers scan the display and build an awareness of the situation and formulate a course of action. Potential action items are prioritized and placed into a working memory store (called *task buffer* in the figure). For example the task buffer may include a planned sequence of actions such as the following:

- Change altitude of *DAL341* to FL300 (30,000 feet)
- Change speed of *DAL341* to 35 (350 knots)
- Change heading of *AAL021* to the northeast quadrant

The limit of how many information items can be stored in such a buffer sequence is dependent upon the complexity of the information and the storage duration. Much work has been devoted to capturing these limits in short term memory, in addition to understanding the memory mechanisms themselves (Baddeley, 1986; Braver et al., 1997; Cowan, 2001; Just & Carpenter, 1992; Miller, 1956; Squire, 1992). It is important to note that in this research the ultimate complexity metric is not absolute (i.e. measured with respect to some absolute human memory limit) but relative (i.e. measured between different task and interface variants).

In Figure 3-10 the shaded region contains the four basic tasks which operators had to carry out. These include the direction of aircraft (primary task #1), conflict detection (primary task #2), responding to questions and confirming alerts (secondary tasks). Following the start of a scenario, there are no questions or alerts to attend to for fifteen seconds so an operator generally begins by directing aircraft. Once the operator decides upon the aircraft of greatest priority and determines the desired command, the potential for a resulting conflict is checked before finally issuing the command. The operator then follows the next item in the task buffer, or reassesses the situation. At some point the requirement to respond to a question arises. This is added to the task buffer, and once its position in the queue is reached, the operator will search for the information and answer the question. At some point an alert will appear and when it is detected by the operator, the requirement to confirm the alert is added to the buffer. Since this is such a rapid task it is usually carried out almost immediately after an alert is detected. These activities continue throughout, with a total of sixteen questions appearing and four alerts. The duration of a single scenario is 540 seconds. In Figure 3-10 the leftmost box (green outline) indicates the start of the process and the rightmost box (red outline) indicates the end. The following subsections break down and discuss interaction process components in greater detail.

3.2.3.3.1 Primary task #1: Aircraft selection and direction

Directing aircraft to the sector exits while avoiding conflicts is the priority of the operators. The first step in this process is developing an understanding of the situation. This consists of capturing the aircraft positions and states with respect to the exit requirements. As a result of this, a certain aircraft is selected which has the highest priority. As noted in the CTA observations, the aircraft that was nearest to the sector exit line was generally selected first. This simple strategy for aircraft prioritization only imposes a perceptual load, and requires no higher level cognitive processes such as the comparison of two or more numbers. The process flow chart is shown in Figure 3-11. Once an aircraft is selected, it is determined whether a new aircraft state needs to be commanded (illustrated by the *Change required?* box). The process of determining whether a change is required is broken down further as shown in Figure 3-12.

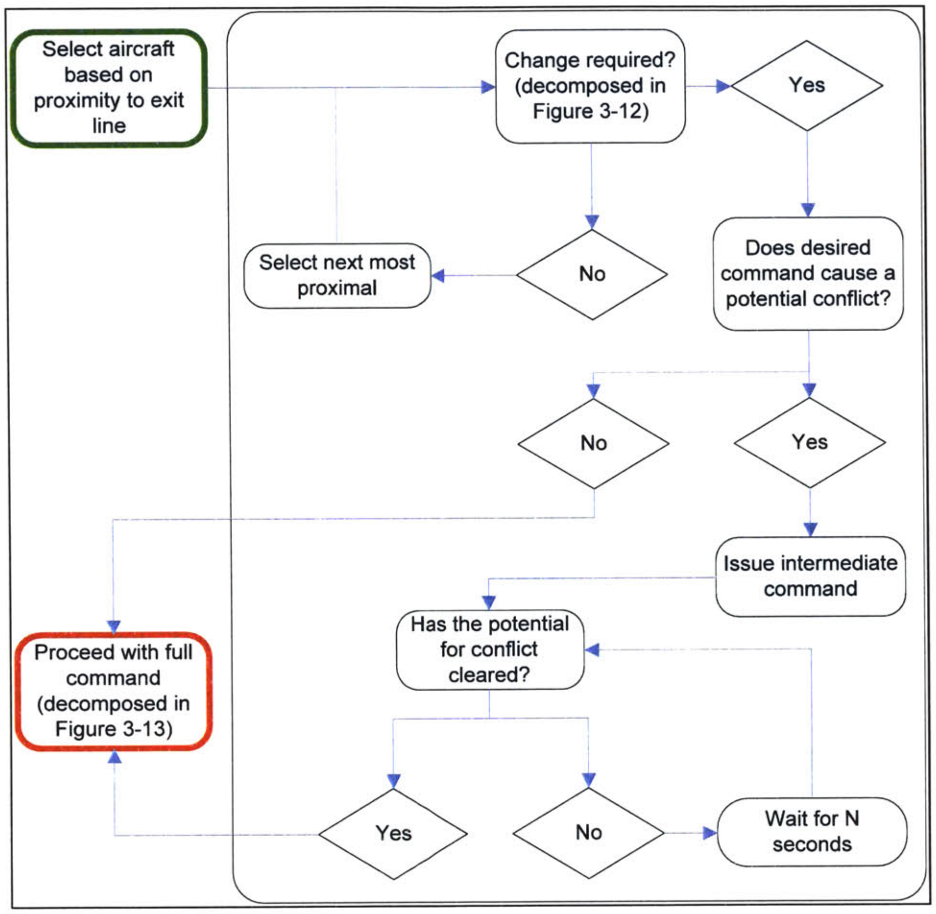

Select aircraft based on proximity to exit line
Change required? (decomposed in Figure 3-12)
Yes
Select next most proximal
No
Does desired command cause a potential conflict?
No
Yes
Issue intermediate command
Has the potential for conflict cleared?
Proceed with full command (decomposed in Figure 3-13)
Yes
No
Wait for N seconds

Figure 3-11. Aircraft selection and direction portion of the cognitive process

If no change is required, the next most proximal aircraft is selected. If a change is required, the operators check whether that required command has the potential of causing a conflict with other aircraft. If it does not, the full change of state can be commanded. If there is a potential for conflict an intermediate command is issued, which is followed up with the complete command after the potential for conflict has cleared. It is important to note that even though in Figure 3-11, the response to whether there is the potential for conflict is *yes* or *no*, in actuality the operator generates a response with an associated probability.

Determining whether a change to an aircraft state is required is a sub-process in itself and is shown in Figure 3-12. At first once an aircraft has been chosen based on priority, the exit gate is read from the top line of the data-block (Z1 through Z4). This information must be held in short term memory. Each of these four egresses has an associated set of exit requirements. If these are memorized by the operator, the commands can be issued immediately. If any one of these requirements is not known however, the operator must shift attention to the chart with the exit specifications and extract the requirements from there. Once the information is available, the desired value is compared to the current value. The current value for the heading is perceived from the graphical orientation of the aircraft symbol. The current altitude and speed are captured from the second line of the data-block. If the aircraft has already been selected, the current value can be also seen from the edit box within the command panel. This also requires that a radio button has been selected (see Figure 3-8) for heading, altitude or speed.

It is important to note that while determining if a commanded change is required, the mental decision process may or may not be coupled with the physical action of clicking on the aircraft and issuing a command through the panel. For example, the selection of the aircraft included in Figure 3-12 may not necessarily occur in the sequential position shown.

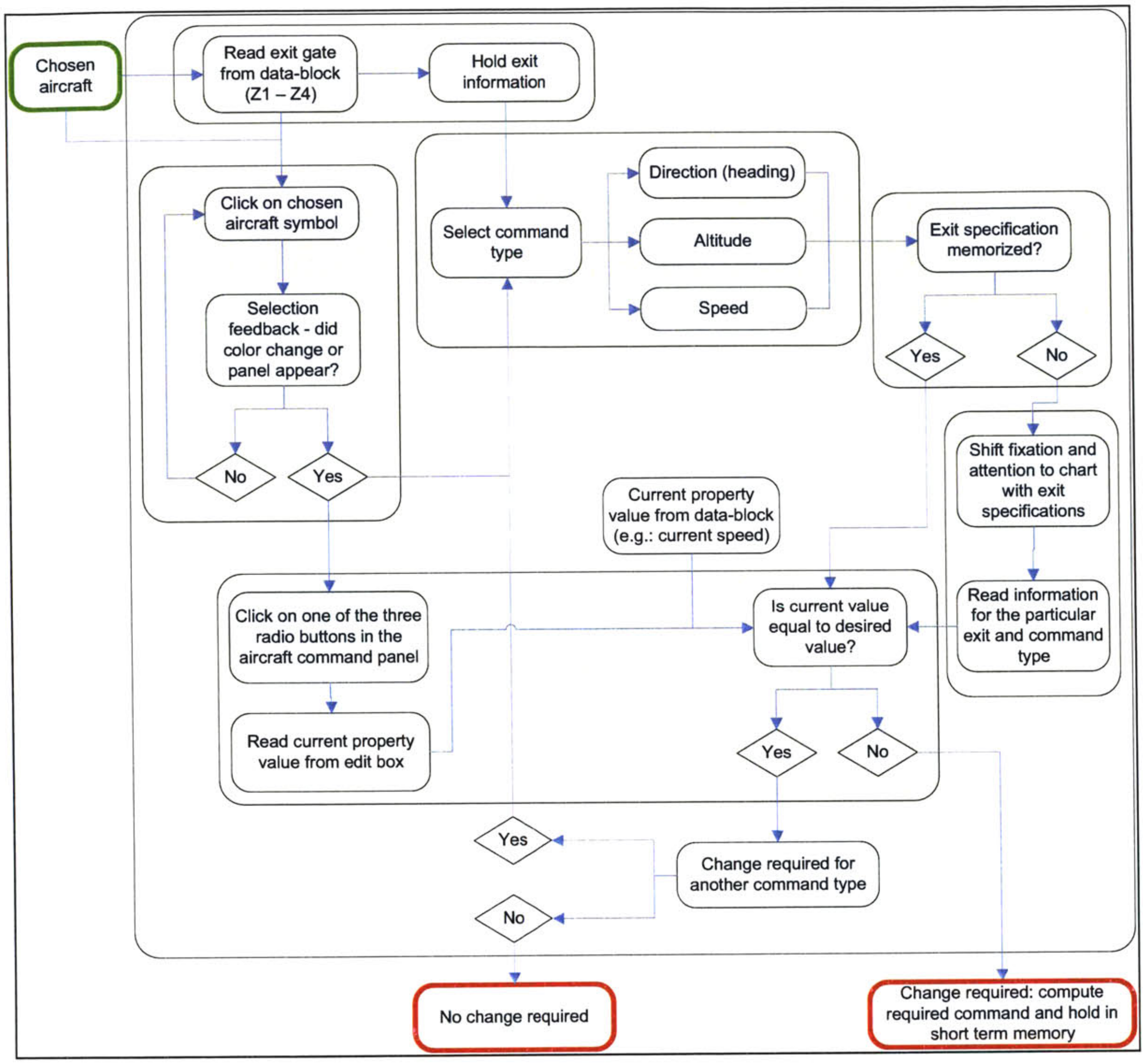

Chosen aircraft
Read exit gate from data-block (Z1 – Z4)
Hold exit information
Click on chosen aircraft symbol
Selection feedback - did color change or panel appear?
No
Yes
Select command type
Direction (heading)
Altitude
Speed
Exit specification memorized?
Yes
No
Shift fixation and attention to chart with exit specifications
Read information for the particular exit and command type
Current property value from data-block (e.g.: current speed)
Click on one of the three radio buttons in the aircraft command panel
Read current property value from edit box
Is current value equal to desired value?
Yes
No
Change required for another command type
Yes
No
No change required
Change required: compute required command and hold in short term memory

Figure 3-12. Cognitive process flow chart for determining whether a change is needed to the current aircraft heading, altitude or speed to meet exit objectives.

The next portion of the chart in Figure 3-11 which is decomposed is the execution of the chosen aircraft command using the interface. The resulting cognitive process is shown in Figure 3-13. Once the desired command value has been determined, the aircraft must be clicked upon if this has not happened already. Once the panel appears, the controller moves the cursor towards it, and clicks one of the three radio buttons. The current value then appears in the edit box. The controller clicks in the edit box and switches between input devices (mouse to keyboard). The text is erased, and the new value is typed. The controller then switches back to the mouse and moves the cursor to the *Command* button, which is clicked.

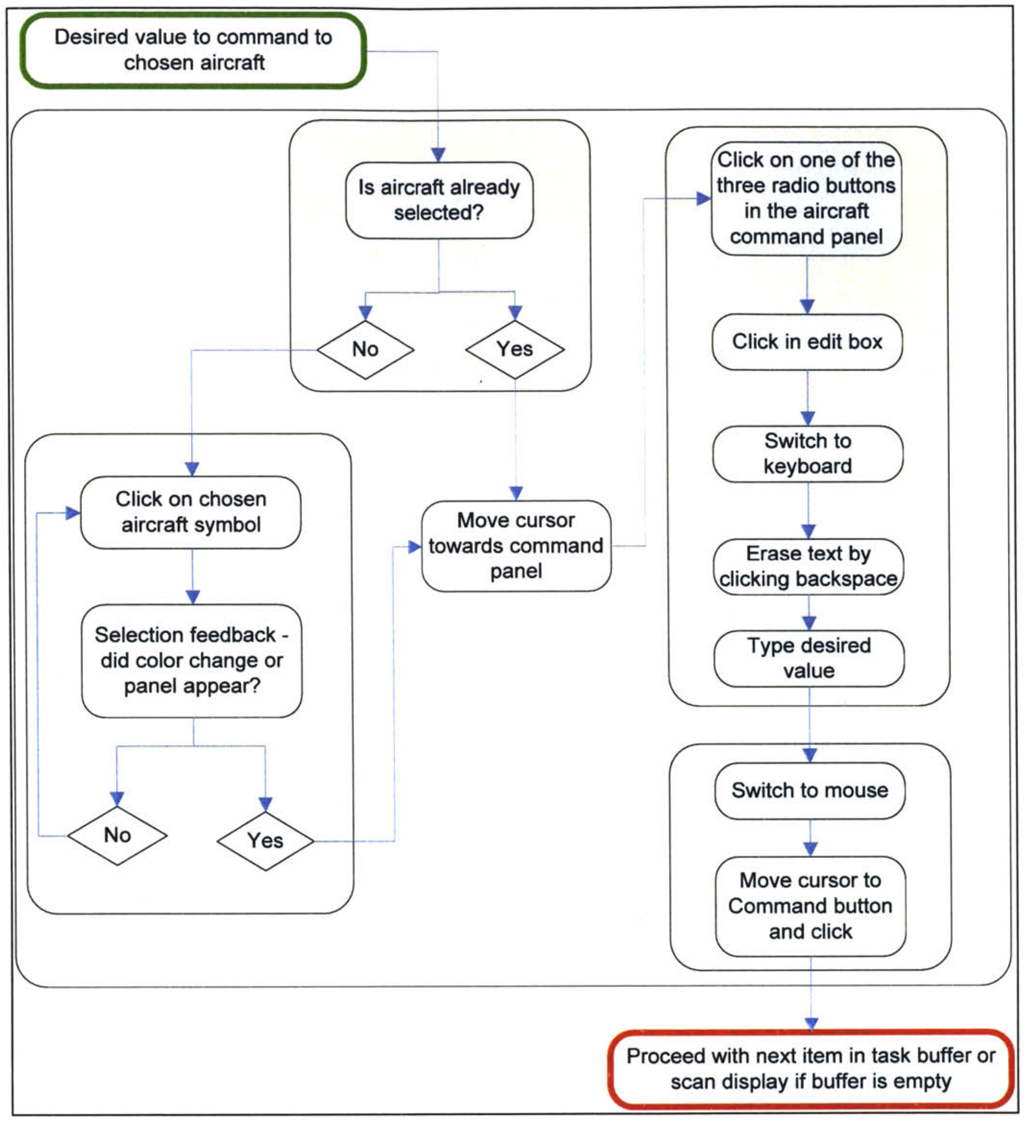

Desired value to command to chosen aircraft
Is aircraft already selected?
No
Yes
Click on chosen aircraft symbol
Selection feedback - did color change or panel appear?
No
Yes
Move cursor towards command panel
Click on one of the three radio buttons in the aircraft command panel
Click in edit box
Switch to keyboard
Erase text by clicking backspace
Type desired value
Switch to mouse
Move cursor to Command button and click
Proceed with next item in task buffer or scan display if buffer is empty

Figure 3-13. Cognitive process flow chart of issuing a known command to an aircraft.

The time to complete each command type was measured and tabulated in Table 3-3. This average time spans from the time the aircraft is selected by clicking on the symbol, and includes the time to move the cursor to the radio button and click, clicking in the text box, switching to the keyboard, erasing the current text, checking the chart for the exit specification, deciding upon the desired command, typing the desired command, and finally transitioning back to the mouse and clicking the *Command* button. This time does not vary substantially between command types, although participants not comfortable with compass headings may have a more difficult time determining a desired heading angle command. In that case the command time would likely be greater for heading commands.

Table 3-3. Command execution time by command type

	Command Time (s)		
	Heading	***Altitude***	***Speed***
Average	7.1	6.7	6.5
St. Deviation	1.0	1.0	0.9

3.2.3.3.2 Primary task #2: Conflict detection and avoidance

Conflict detection and avoidance is intertwined with the direction of aircraft cognitive processes described in the previous section (in fact, if *primary task #1* is conducted correctly no conflict avoidance is required). This subsection attempts to capture cognitive processes involved with the detection of potential conflicts. The flow chart is shown in Figure 3-14. Controllers basically scan the aircraft in the vicinity of a certain aircraft and check for intersecting trajectories. Flight levels are also checked to determine the potential for these to intersect. If a conflict is detected, a command is issued to avert it.

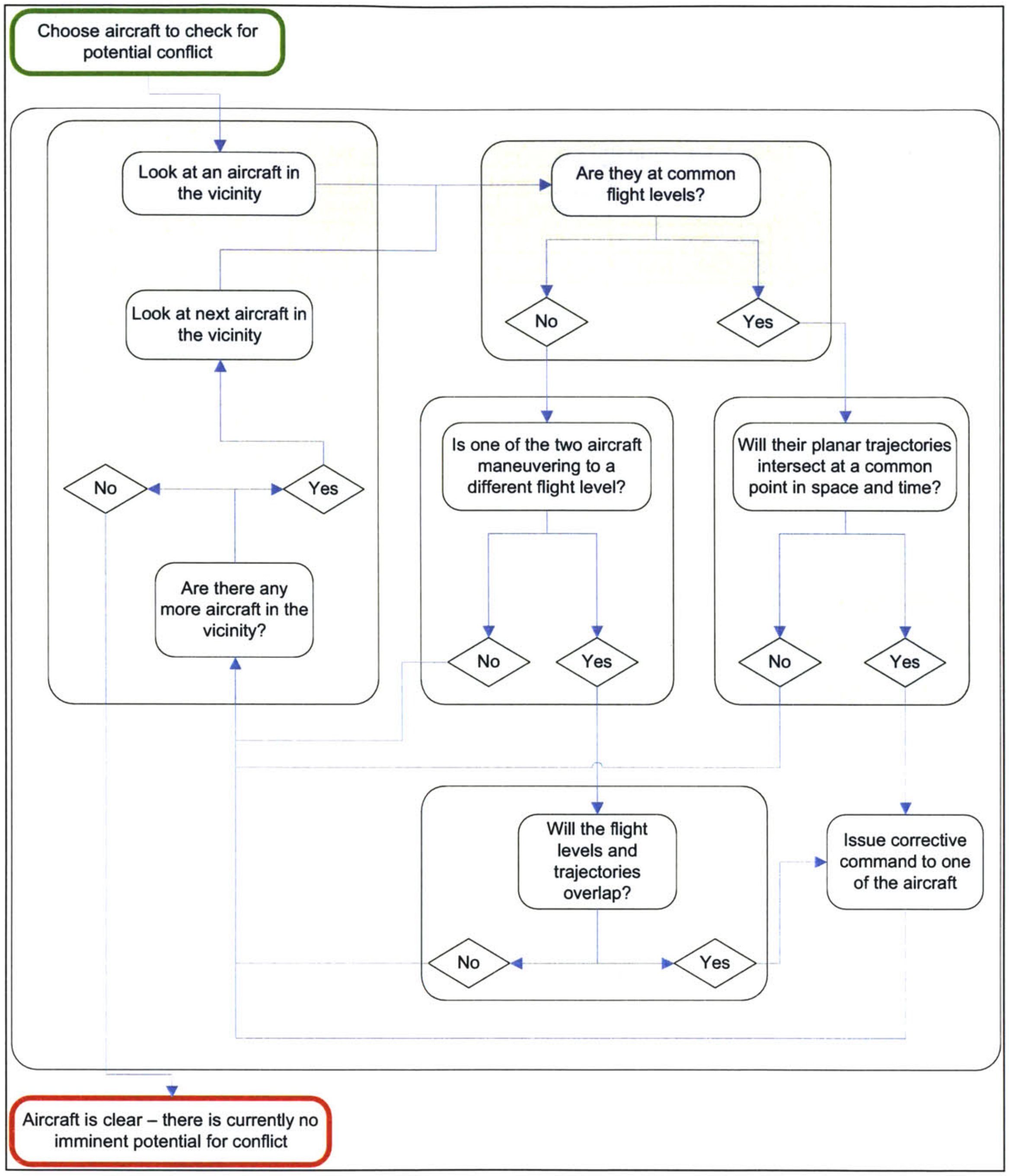

Choose aircraft to check for potential conflict
Look at an aircraft in the vicinity
Are they at common flight levels?
No
Yes
Look at next aircraft in the vicinity
Is one of the two aircraft maneuvering to a different flight level?
Will their planar trajectories intersect at a common point in space and time?
No
Yes
Are there any more aircraft in the vicinity?
No
Yes
No
Yes
Will the flight levels and trajectories overlap?
Issue corrective command to one of the aircraft
No
Yes
Aircraft is clear – there is currently no imminent potential for conflict

Figure 3-14. Cognitive process of conflict detection for a given aircraft relative to nearby aircraft

3.2.3.3.3 Secondary tasks: Question responses and alert confirmations

The first step in the cognitive process of responding to a question is the detection of the appearance of a new question. The operator is usually alerted to the new question by the aural alert, but can also notice this by the visually perceived change in the displayed question text. The cognitive process chart is shown in Figure 3-15. During the CTA procedure it was observed on a number of occasions that when under very high workload, operators would fail to consciously perceive the aural alert and know that a new question had been asked. This is similar to the phenomenon of inattentional blindness that happens within the visual system (Mack & Rock, 1998). Once the operators finally sense the alert consciously and decode its meaning, they recognize the need to carry out the question response task. This is given a priority and added to the task buffer. Once other higher priority subtasks have been fulfilled (such as completing an aircraft command), the operator shifts attention to the data-link interface, and reads the question. From this, the operator knows the aircraft flight number to search for, as well as the piece of information being requested.

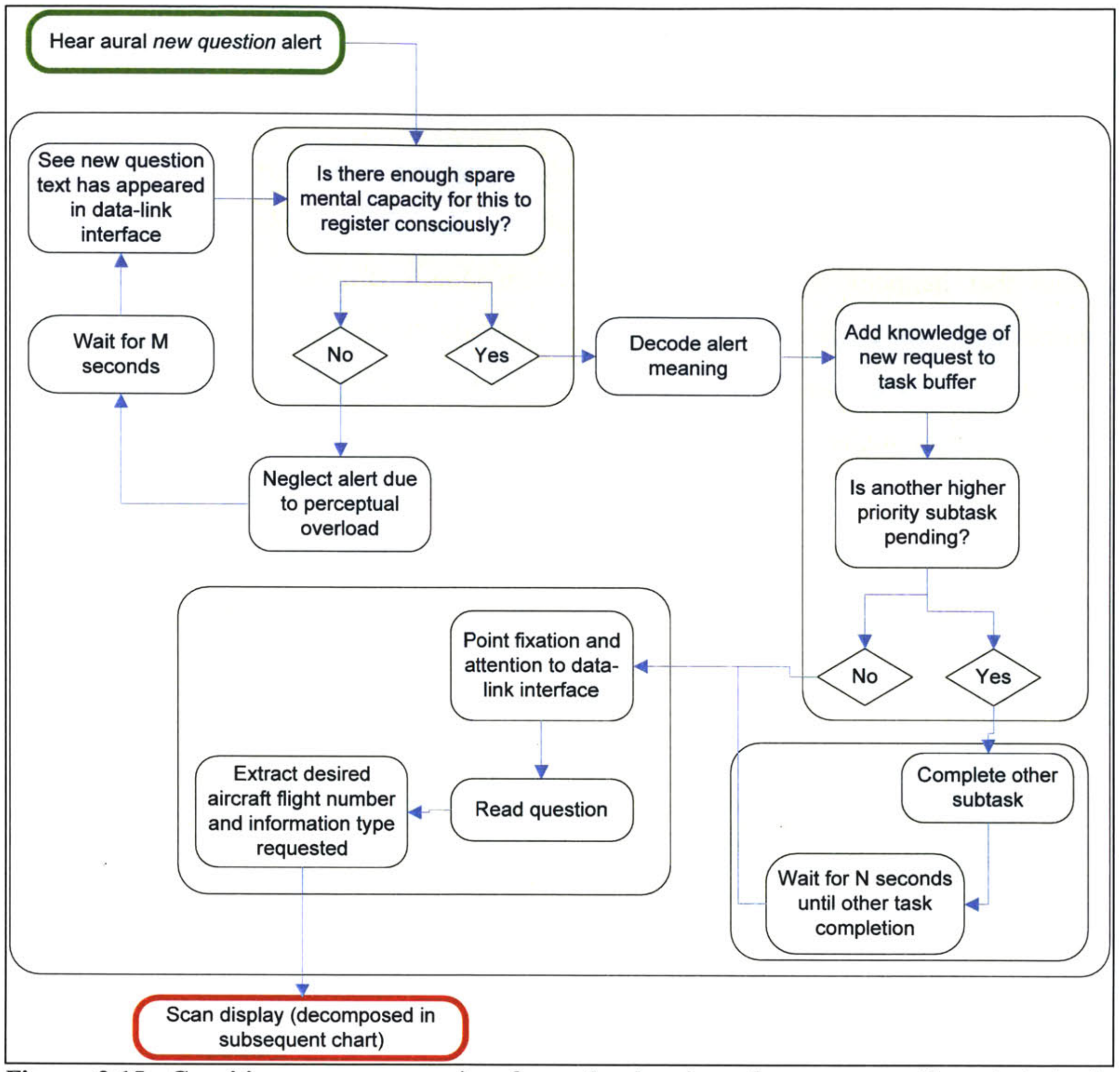

Hear aural *new question* alert
See new question text has appeared in data-link interface
Is there enough spare mental capacity for this to register consciously?
Wait for M seconds
No
Yes
Decode alert meaning
Add knowledge of new request to task buffer
Neglect alert due to perceptual overload
Is another higher priority subtask pending?
Point fixation and attention to data-link interface
No
Yes
Extract desired aircraft flight number and information type requested
Read question
Complete other subtask
Wait for N seconds until other task completion
Scan display (decomposed in subsequent chart)

Figure 3-15. Cognitive process spanning from the hearing of a new question alert to the commencement of scanning for a given aircraft.

Once these two pieces of information are known, the operators begin scanning the radar display for the given flight number temporarily held in short term memory. The cognitive process chart of this is shown in Figure 3-16. This process consists of eye movements to the various targets followed by a comparison of the fixated data-block flight number with the desired flight number. This scan sequence was interrupted when the operator believed that these two flight numbers matched. It was noted that operators sometimes performed a commission error by choosing the incorrect flight when it was of the same airline as the correct target. At that point the process of extraction of information from the data-block begins.

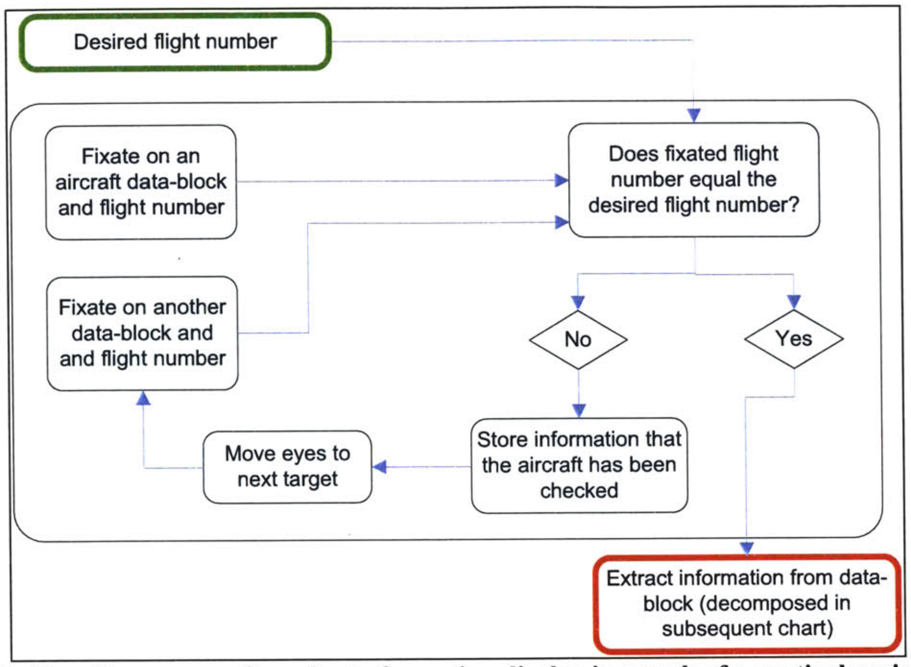

Desired flight number
Fixate on an aircraft data-block and flight number
Does fixated flight number equal the desired flight number?
Fixate on another data-block and and flight number
No
Yes
Move eyes to next target
Store information that the aircraft has been checked
Extract information from data-block (decomposed in subsequent chart)

Figure 3-16. Cognitive process flow chart of scanning display in search of a particular aircraft flight number.

The structure of the cognitive process of extracting information from the data-block depends upon the data-block type. Therefore two process charts follow: one for data-blocks with embedded information[11] and another for the data-block with all the information on the surface[12].

The cognitive process chart for the data-blocks which have embedded information is shown in Figure 3-17. The structure of this process is the same for the two, three and four line data-blocks. However the execution of the various loops changes between these three data-block types with embedded information. From the knowledge of the desired type of information, the controller determines whether that information is embedded or already visible. If the information is embedded the controller moves the cursor and clicks on the double arrowhead which opens the next line. If the information is still embedded, the controller must repeat the above process to open yet another line. Once the line with the desired information has been opened the operator can fixate directly on the location of the required information if its position is known. The other strategy would be to identify the piece of information by its format and scan multiple items in the data-block. It was

[11] These are the *2 line*, *3 line* and *4 line* data-blocks
[12] This is the *5 line* data-block

observed however that controllers memorized the position of the information items within the data-block relatively quickly.

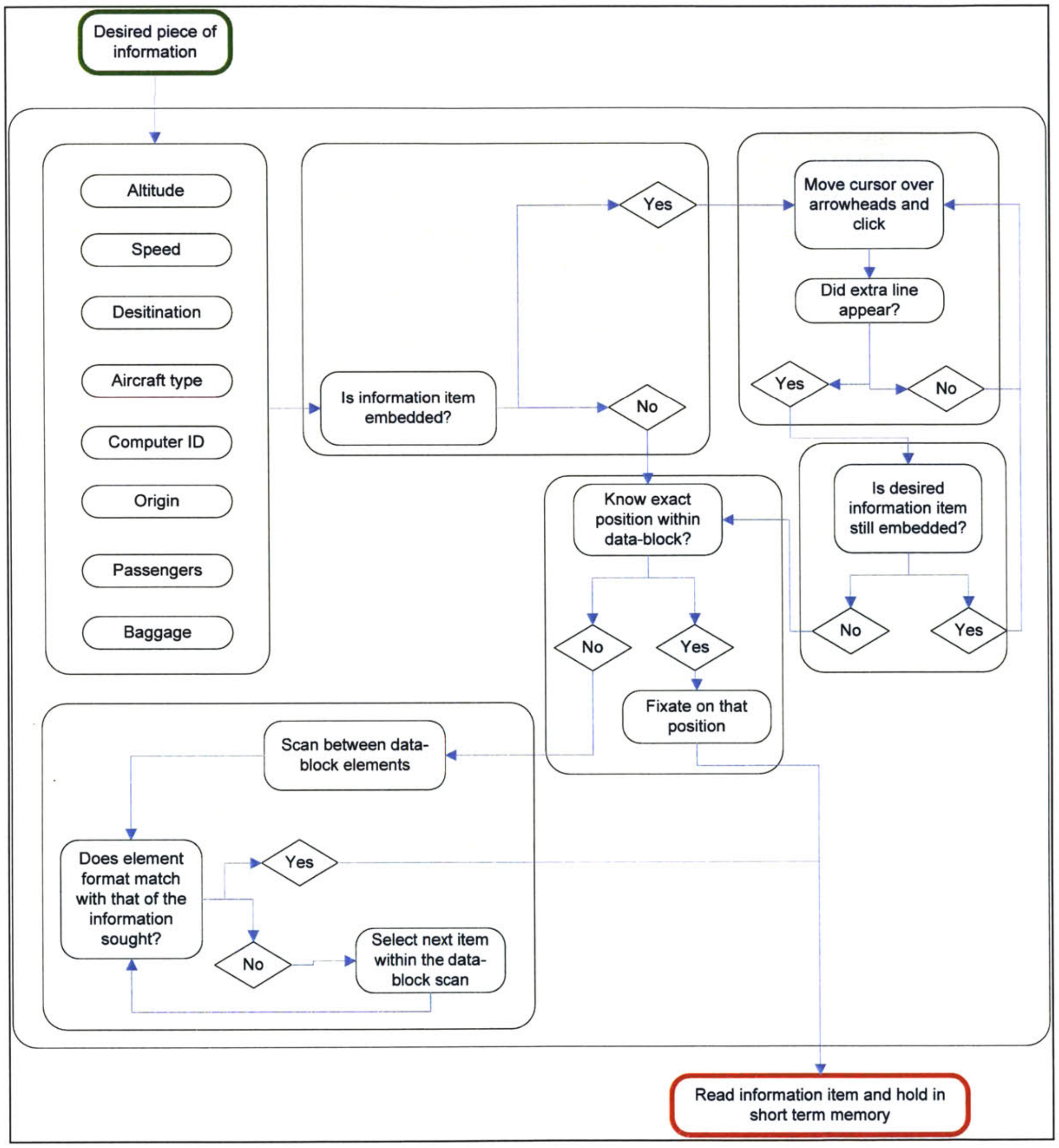

Desired piece of information
Altitude
Speed
Desitination
Aircraft type
Computer ID
Origin
Passengers
Baggage
Is information item embedded?
Yes
No
Move cursor over arrowheads and click
Did extra line appear?
Yes
No
Is desired information item still embedded?
No
Yes
Know exact position within data-block?
No
Yes
Fixate on that position
Scan between data-block elements
Does element format match with that of the information sought?
Yes
No
Select next item within the data-block scan
Read information item and hold in short term memory

Figure 3-17. Cognitive process chart for extraction of information from data-block with embedded information.

The cognitive process chart for the full data-block that always displayed five lines is shown in Figure 3-18. From the knowledge of the desired type of information, the controller can either identify the information by known position in the data-block, or by differences in the coding format of the information within the data-block. In other words one strategy would be to extract the information by knowing the spatial position of that information within the data-block. The other would be to identify the piece of information by its format (e.g., know R128 is a *computer identification number*[13] because it begins with a single letter and is followed by three digits). The second strategy would require a human to scan through the various elements of a data-block, checking at each step for a match. It is evident that this second strategy can result in increased errors if the format of two items is similar. These commission errors were noticed multiple times while observing novice participants, as they oftentimes would incorrectly type the destination instead of the origin and vice-versa.

[13] Abbreviated *CID* in ATC

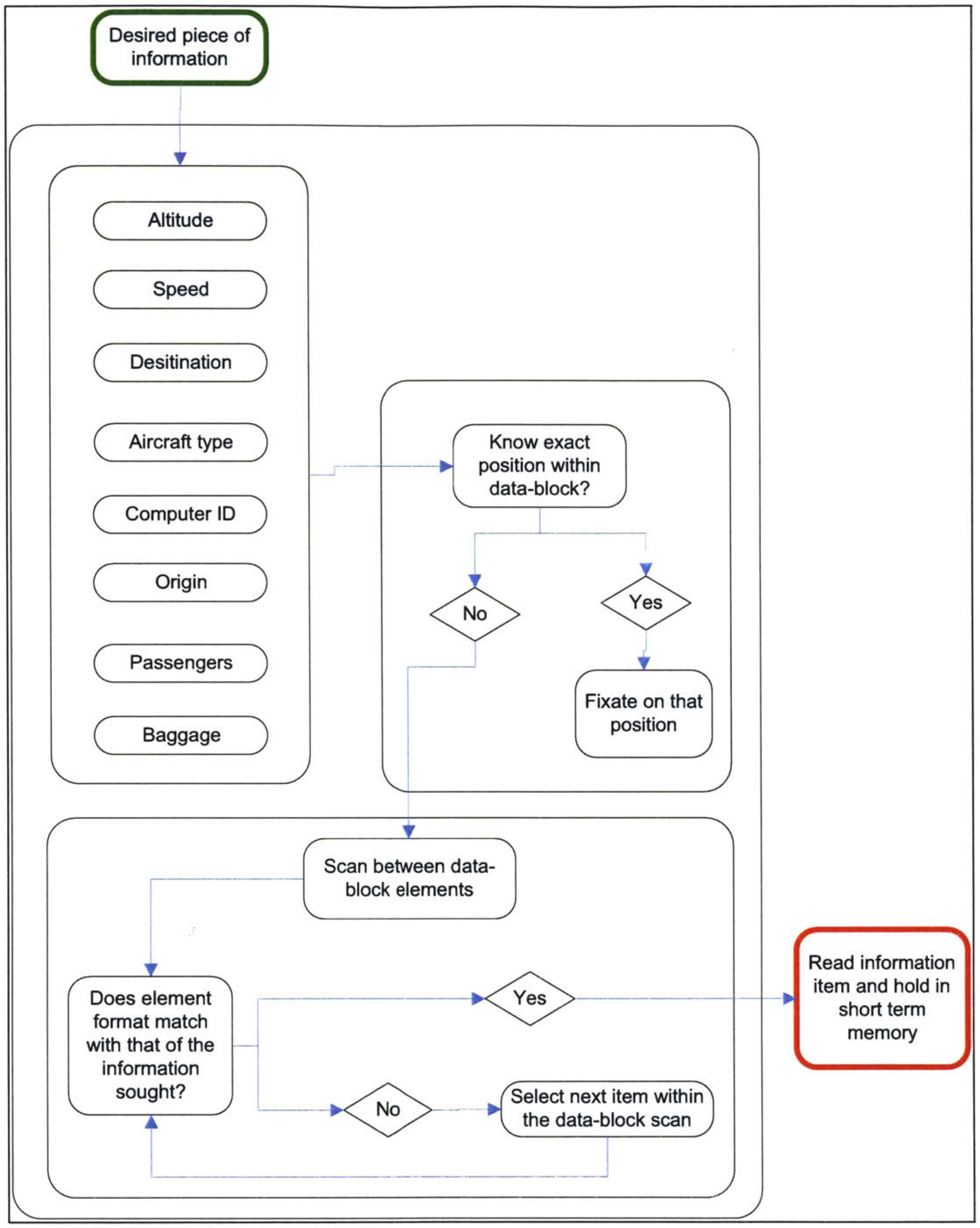
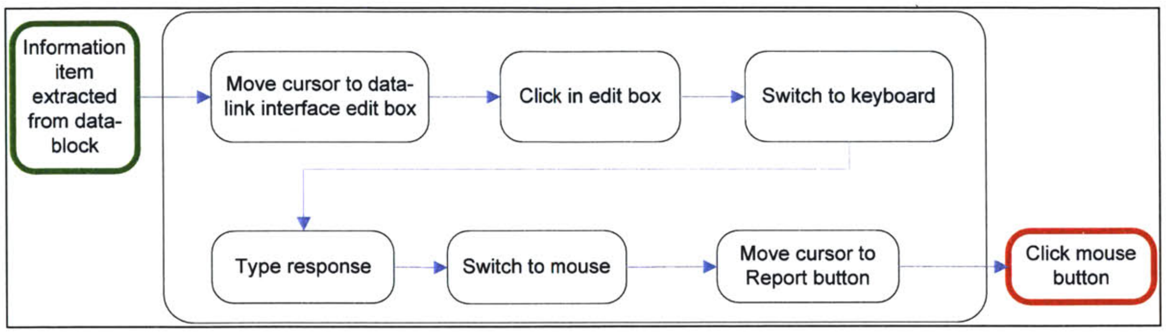

Desired piece of information
Altitude
Speed
Desitination
Aircraft type
Computer ID
Origin
Passengers
Baggage
Know exact position within data-block?
No
Yes
Fixate on that position
Scan between data-block elements
Does element format match with that of the information sought?
Yes
No
Select next item within the data-block scan
Read information item and hold in short term memory

Figure 3-18. Cognitive process chart for extraction of information from five line data-block.

Finally once the information is found and held in short-term memory, the controller submits it by using the data-link interface. The cognitive process of the submission of responses is shown in Figure 3-19. This simple process requires the operator to click in the edit box, type the response, and then press the *Report* button to submit.

Once this is complete, the question response subtask is concluded. At that point the operator takes on the next task held in the task buffer, or scans the display in order to develop new task objectives. At some future time this whole process is repeated when a new question appears.



Information item extracted from data-block
Move cursor to data-link interface edit box
Click in edit box
Switch to keyboard
Type response
Switch to mouse
Move cursor to Report button
Click mouse button

Figure 3-19. Cognitive process chart for submitting an answer using the data-link interface.

The data-block type has the primary effect of changing the extraction process of information from the data-block. The efficiency of this task affects the performance of other tasks. The time penalty for extracting information from the data-block reduces the time available for conducting other tasks, such as directing aircraft and avoiding conflicts. Estimates of these extraction times are shown in Table 3-4. These values are the averages of only three measurements, so no statistical analysis is conducted. The shaded cells in the table represent access times of embedded information. It is important to note that increasing the base layer lines in the data-block does not increase the time to access the second line which contains the altitude and velocity. Subjective observations indicate that participants perceive the entire data-block as a single entity in the periphery and then begin their within data-block scan from the top left corner of the data-block text if they are conducting a search for a piece of information. This may explain the result shown in Table 3-4 showing that the number of base layer lines in the data-block does not affect the time required to access information from second line.

Table 3-4. Average extraction time for information on each line for each data-block type. Shaded cells represent embedded extraction times for embedded information.

		Data-Block Type			
		2 Line	*3 Line*	*4 Line*	*5 Line*
Line	2nd	0.8 s	0.8 s	0.8 s	0.9 s
	3rd	2.2 s	1.1 s	1.0 s	1.2 s
	4th	3.1 s	2.7 s	1.5 s	1.4 s
	5th	4.7 s	3.1 s	2.3 s	1.1 s

The data-block types could have additional cognitive information processing consequences due to the different levels of display clutter which they each impose. These may have to be considered in the CIA functions for scanning the display for an aircraft. The results show however that the subtask times for searching for a specific flight number on the radar scope do not increase with the number of data-block base layer lines. The implication of this with respect to the resulting CIA is the assumption that the functions for scanning the display need not account for the increased number of data-block lines, but only for the number of aircraft. These time estimates are summarized in Table 3-5 and are from measurements taken from a single participant over multiple trials.

Table 3-5. Average time to locate aircraft flight on radar display for two levels of aircraft

Data-Block Type	Time to locate flight number (s)	
	Low Aircraft Count	*High Aircraft Count*
2 Line	2.7	3.2
3 Line	2.5	3.4
4 Line	2.3	3.7
5 Line	1.9	3.5

The purpose for extracting the subtask times is not a requirement for the methodology but the times are examined for two secondary purposes. First it helps determine in Chapter 5 whether algorithmic execution time (of the human) is correlated to the algorithmic complexity (information content). If the correlation between algorithmic complexity and execution time is high, this indicates that the additional effort of collecting CIA execution time is likely not warranted in future work. A high correlation would suggest that the CIA complexity and execution time measure the same construct. If the correlations are different, the question of whether cumulative CIA execution time is a better predictor of performance remains open. The second purpose is that subtask times can potentially help with the CIA function composition. For example, if the extraction time of information from the second line of a data-block is the same regardless of the data-block's size, this suggests that the cognitive search algorithm does not scan through each element in the data-block independently.

3.2.4. Cognitive task analysis conclusions

The CTA is the first step of the complexity analysis methodology and Section 3.2 illustrated a CTA for a representative ATC interface and en-route control task. The techniques used in this CTA included observations, interviews, task breakdown, and the construction of cognitive process charts. There exist a number of CTA methods in the literature which can form the basis of this step of the complexity analysis methodology. The scope of the CTA is to ultimately capture knowledge regarding the cognitive processes of operators which form the basis for the CIA. The accuracy and level of detail of the CTA plays an important part in determining the precision of the CIA, since it describes the processes that must be captured in an algorithm.

3.3. Step Two: Cognitive Interaction Algorithm

The purpose of this section is to express the interaction processes described by the CTA as an algorithm. This cognitive interaction algorithm (CIA) is intended to be a high level approximation of whatever neural algorithms are being executed within the human central nervous system (CNS), during the execution of some task, using a given interface. At the highest level, each configuration analyzed has a core CIA from which the required CIA functions are called. CIA functions are written for each of the subtasks and cognitive processes.

3.3.1. Developing the cognitive interaction algorithm

The CIA can be written in any formal computational language, as long there is consistency in each of the configurations or scenarios being compared. For example when wishing to compare display configuration *A* with *B*, one must apply the same language and the same level of detail to the algorithms representing each of the two.

Furthermore the display can never be considered independently of the task. The complexity of an interface only makes sense when considered relative to both the human and the task.

For the example case, since the air traffic control simulation itself was programmed in Matlab®, for consistency the syntax and language used for the expression of the CIA is pseudocode based on the Matlab® *M* language. The algorithm is based on the knowledge gained from the CTA performed in Section 3.2. The level of detail of the algorithmic expression used in this example primarily focuses on capturing the actions that occur in a nominal scenario. For example, the algorithms constructed from the CTA presented previously do not describe how the human brain recognizes an object but do capture mouse clicks, key strokes, eye movements and other such higher level processes. If the object of analysis dictated that further detail was needed, the researcher could draw relevant algorithms developed in the field of computational cognitive science or artificial intelligence (or attempt to develop new ones based on available cognitive science and neuroscience literature).

3.3.1.1. Algorithm syntax

The language used to express the cognitive interaction algorithm is briefly described in this section. Symbols used in the expression of the CIA are summarized in Table 3-6. In this language function calls begin by stating the variable that the function returns (if any), followed by an equals sign and then the actual function name and the input variables (e.g.: *function output = input(x,y)*). The code is generally intuitive, however for more information about the language one can refer to Magrab (2000) or any other similar text on M language programming.

Table 3-6. Description of symbols used in CIA

Logic Operators	
&	and
==	equal to
~=	not equal to
|	or
Other symbols	
=	assigns a value
.	subclass of a mental variable (e.g. aircraft.position)

The advantage of expressing a cognitive process as an algorithm as opposed to a flow chart is important. A programming language offers a more formal and concise way of describing a complex process and can capture processes that cannot be expressed graphically with a flow chart. The cognitive flow charts presented in 3.2.3.3 offer the advantage of yielding a more rapid and clear understanding of the cognitive processes, but do not lay the foundations or provide the means for a more rigorous and accurate expression of human information processing suitable for quantitative analysis. This is one of the propositions emphasized in this work: in order to analyze human machine interaction more rigorously, the cognitive interaction process should be expressed as an algorithm. The complexity of the algorithm then yields an approximation of the information complexity of the human machine interaction process associated with a given interface and task configuration.

3.3.1.2. Cognitive interaction algorithm functions

In this section the CIA functions are presented and described. These are the components that eventually compose the CIA for each interface and task configuration. Each of these functions contains algorithms which describe processes conducted by operators. For example, the function *new_question_info* contains the algorithmic representation of the process of detecting the appearance of a new question and gathering the required information from the question. Table 3-7 includes a list of the CIA functions for the example case along with a brief description of each. A CIA function may call other functions just as in a conventional computer program. For clarity, the structure of each of the CIA functions is also presented in flow charts indicating the number of calls to other CIA functions. A single CIA is ultimately written for each configuration being compared. This is referred to as the base (or core) CIA and it is from here where all CIA functions are called.

Table 3-7. List and description of CIA functions

CIA Function	Brief Description
command_hdg	operator commands a new aircraft heading
command_alt	operator commands a new aircraft altitude
command_spd	operator commands a new aircraft speed
selectAC	operator selects a new aircraft to direct
new_question_info	detects new question and gathers objective information
find_flight_low	find a given flight on the radar display with a low aircraft count
find_flight_high	find a given flight on the radar display with a high aircraft count
scanpath_low	scan between aircraft data-blocks on a radar with a low aircraft count
scanpath_high	scan between aircraft data-blocks on a radar with a high aircraft count
submit	submit a question response via data-link
seedot	detect the appearance of an alert besides a data-block
read	read a portion of text
report	report alert confirmation
move_eyes_to	move eyes to a target onscreen
move_hand_to	move hand to a different position
move_cursor_to	move cursor to a target onscreen
clickbutton	click the mouse button
findinDB2_2	find information from 2nd line of 2 Line data-block
findinDB2_3	find information from 3rd line of 2 Line data-block
findinDB2_4	find information from 4th line of 2 Line data-block
findinDB2_5	find information from 5th line of 5 Line data-block
findinDB3_2	find information from 2nd line of 3 Line data-block
findinDB3_3	find information from 3rd line of 3 Line data-block
findinDB3_4	find information from 4th line of 3 Line data-block
findinDB3_5	find information from 5th line of 3 Line data-block
findinDB4_2	find information from 2nd line of 4 Line data-block
findinDB4_3	find information from 3rd line of 4 Line data-block
findinDB4_4	find information from 4th line of 4 Line data-block
findinDB4_5	find information from 5th line of 4 Line data-block
findinDB5_2	find information from 2nd line of 5 Line data-block
findinDB5_3	find information from 3rd line of 5 Line data-block
findinDB5_4	find information from 4th line of 5 Line data-block
findinDB5_5	find information from 5th line of 5 Line data-block
type	type a string of characters into an edit box
press	press a specific button on the keyboard with a finger
motor	neural motor command
angle	mental geographic angle computation between 2 points

Only a subset of the CIA functions are presented and described below. These serve to illustrate how the interaction process can be expressed algorithmically by combining a certain set of CIA functions. All CIA functions are written in a standard form without any spaces[14]. This is required in order to maintain the consistency required for the complexity estimation. The remainder of the CIA functions are placed in Appendix B.

Of the functions presented in the main body of this thesis, command functions are first discussed. There are three types of command functions: *command_alt*, *command_hdg* and *command_spd*, for commanding the altitude, heading and speed, respectively. These functions are similar so only the heading CIA function is presented here. The other two are included in Appendix B. The chart showing the functions called by the heading command CIA function is shown in Figure 3-20. The algorithm follows the chart below, and captures the interaction process of clicking an aircraft, deciding upon the desired command value, and issuing that command using the panel in the display.

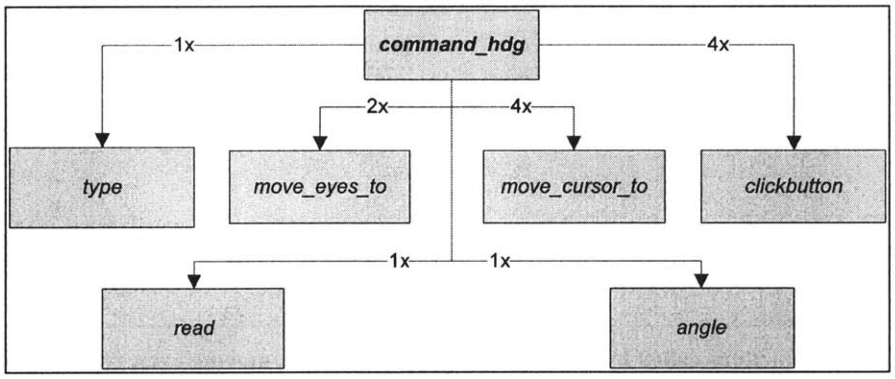


Figure 3-20. CIA functions called by heading command

```
function command_hdg(p)
move_eyes_to(cursor.xy)
move_cursor_to(ac(p).symbol.xy)
clickbutton
ac(p).exit = read(ac(p).exit)
move_eyes_to(chart.xy)
if ac(p).exit == Z1
hdgrequired = 'NW'
elseif ac(p).exit == Z2
hdgrequired = 'NE'
elseif ac(p).exit == Z3
hdgrequired = 'SE'
elseif ac(p).exit == Z4
hdgrequired = 'SW'
end
move_cursor_to(ctrpanel.radiol.xy)
clickbutton
```

[14] By the method by which CIA function information content is quantified, spaces require information to be described.

```
command = angle(hdgrequired,ac(p).xy)
move_cursor_to(ctreditbox.xy)
clickbutton
type(command)
move_cursor_to(cmndbutton.xy)
clickbutton
```

The above CIA function calls other functions. These are *move_eyes_to, move_cursor_to, clickbutton, read, angle* and *type*. The charts for these are all shown in Figure 3-21. The algorithms for each are included in Appendix B.

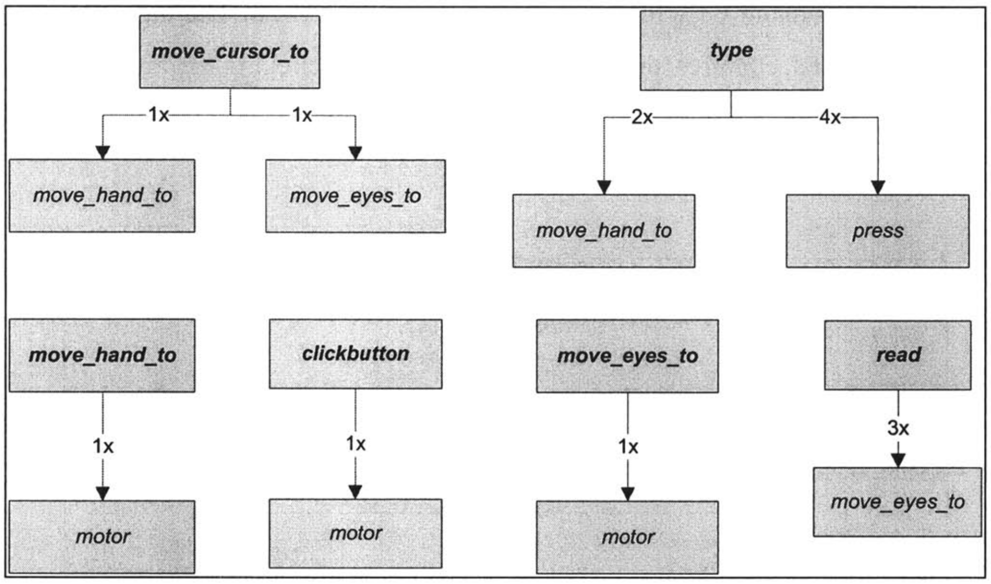


Figure 3-21. CIA functions called by moving cursor, moving hand, moving eyes, typing and reading

The function describing the cursor movements calls the hand movement function, as this is required to move the mouse by some amount proportional to the desired on screen cursor displacement. The eye movement function is also called, as the assumption is made that participants track intentional cursor movements with their eyes. The hand movement function moves the hand to a new position by issuing a motor command of a magnitude consisting of the difference between the current hand position and desired hand position. The eye movement function has the same form as the hand movement function. The *clickbutton* CIA function simply executes the motor command of moving the index finger down to press the left mouse button. The motor command function called is where the level of detail in the CIA ceases. The motor function is a neural command from the brain to whatever muscles are being moved. If this were critical to the configurations being compared by the complexity analysis procedure, one could write a detailed algorithm out for *motor*. The *type* function describes the process of typing a desired command in the edit box, including the hand transitions to and from the keyboard. The *type* function calls the *press* function which describes the actual key pressing.

The following set of CIA functions are those required to respond to a question which has been prompted through data-link. The description of several of these functions follows. The first step in responding to the question is to detect the appearance of a new question. It is assumed that participants always detect the appearance of a new question aurally or visually. Following this the participants must determine what is being asked. Specifically two critical pieces of information are needed: the flight for which the question is being asked and the type of information requested (e.g. DAL341 + destination). The CIA function describing this process is called *new_question_info*. A chart showing the functions called by *new_question_info* is shown in Figure 3-22. The actual CIA function follows the chart.

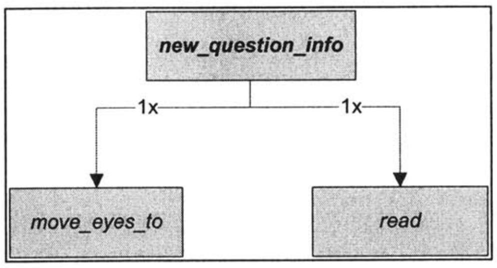


Figure 3-22. Functions called by the CIA function describing the process of gathering new question information.

```
function [target info] = new_question_info
move_eyes_to(datalink.xy)
[target info] = read(newquestion)
```

In the above function it is assumed that participants already have internalized a spatial map of the layout of the display and hence know exactly where to target their eye fixation in the data-link to read the new question. The assumption that humans have internally stored spatial maps is consistent with general animal and human spatial representation research (Wang & Spelke, 2002). The two output variables of this function are stored in short term memory until the information is retrieved from the aircraft data-block.

Once operators know what aircraft to search for, the following step is to locate and fixate upon the target aircraft on the display. This function is called by *find_flight_high* or *find_flight_low* (depending on the number of aircraft). The charts of these two CIA functions are shown in Figure 3-23. The number of calls from each function is based on the assumption that the human searches through approximately 50% of the aircraft on screen prior to locating the desired target. The algorithm for *find_flight_low* follows the chart and the other is included in Appendix B.

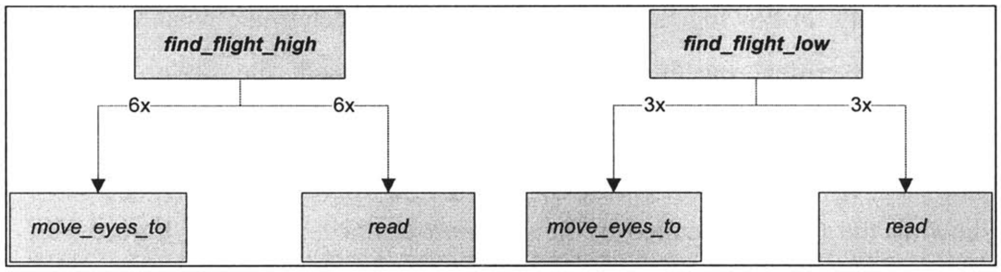


Figure 3-23. CIA functions called by the two find flight functions

```
function targetpos = find_flight_low(fl_no)
temp = 0
while temp ~=target
j=1
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=2
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=3
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=4
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=5
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
end
targetpos = ac(j).xy
```

In the above function, eye movements shift the fixation to the next aircraft in the scan sequence, and the flight number is read. The human brain then performs the simple comparison to see whether this matches with the desired flight number (denoted by the variable *temp* in the above algorithm). Once the target aircraft has been located, the location is output from the CIA function.

Once the aircraft has been located, the process of extracting information from within the data-block commences (i.e. the operator knows that information must be extracted from the data-block in a certain position on screen). It is assumed that the operator knows where the position of the information is, based on the information type.

The extraction process depends on the data-block type and on the line in which the information is contained. There are four CIA functions for extracting information from each data-block type. The chart of the functions called by the two line data-block is shown in Figure 3-24. The CIA functions for extracting information from the data-block follow the chart. The last digit in the function name corresponds to the line from which the information is extracted.

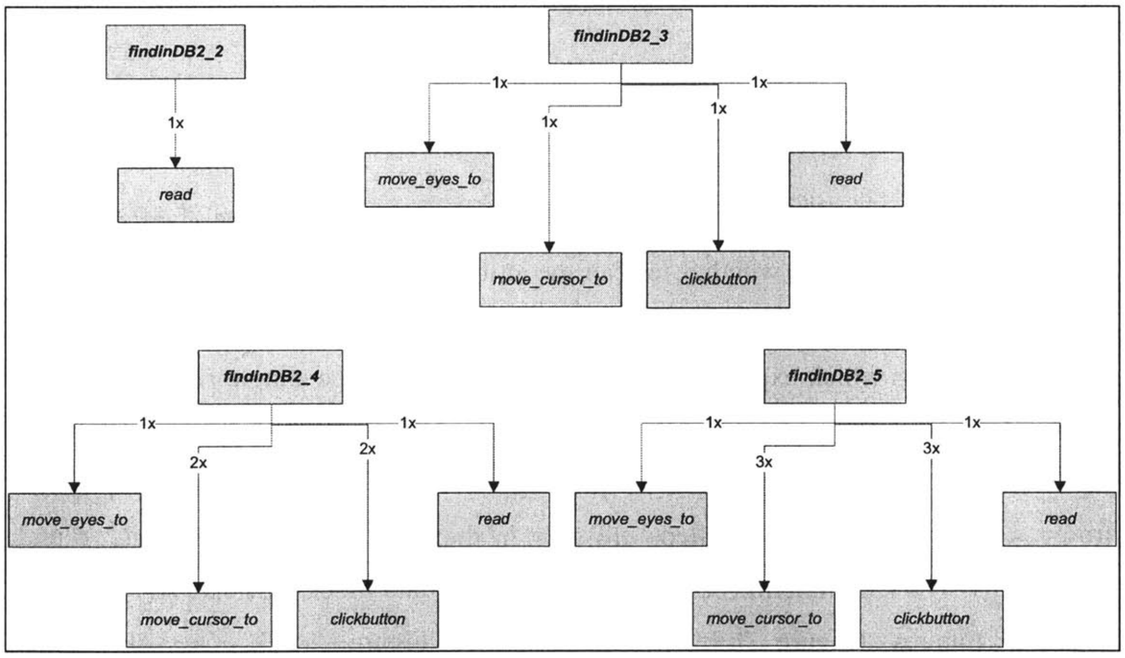


Figure 3-24. CIA functions called by the data-block information extraction functions. The above figure is for the two line data-block.

```
function item = findinDB2_2(j,info)
if info == alt
item = read(ac(j).alt)
else
item = read(ac(j).spd)
end
```

```
function item = findinDB2_3(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == dest
```

```
item = read(ac(j).dest)
else
item = read(ac(j).type)
end
```

```
function item = findinDB2_4(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == CID
item = read(ac(j).CID)
else
item = read(ac(j).origin)
end
```

```
function item = findinDB2_5(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
move_cursor_to(ac(j).downarrows.xy)
clickbutton
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
```

If the information is on the second line of the data-block of the j^{th} aircraft, the human must simply decide whether to read the altitude or speed. If the information is in the second line, the human must move the fixation to the cursor and then move to cursor towards the arrows which expand the data-block, and execute the simple *clickbutton* CIA function explained earlier (see Figure 3-21). If the information is in the fourth or fifth lines of the data-blocks, this process of moving the cursor and clicking the expansion arrows is repeated. These extra steps are seen in the CIA functions *findinDB2_4* and *findinDB2_5*. The CIA functions called by for the other data-block types are shown in the following Figures (Figure 3-25, Figure 3-26 & Figure 3-27). The algorithms corresponding to these are presented in Appendix B.

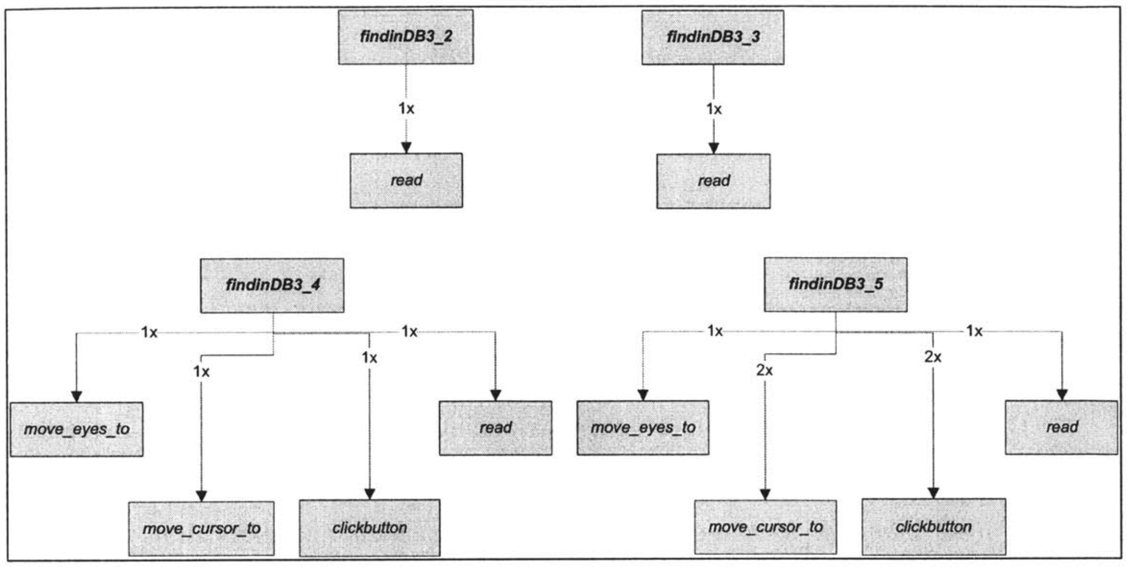


Figure 3-25. CIA functions called by the data-block information extraction functions. The above figure is for the three line data-block.

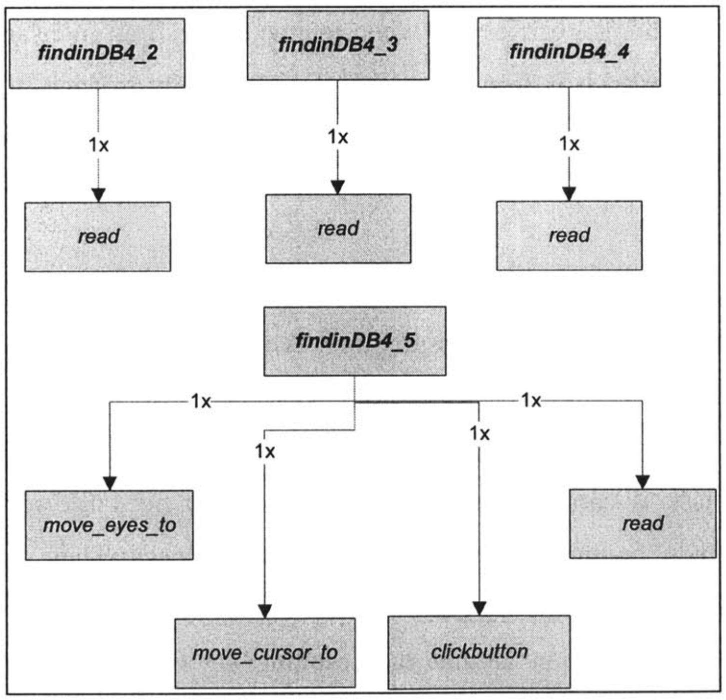


Figure 3-26. CIA functions called by the data-block information extraction functions. The above figure is for the four line data-block.

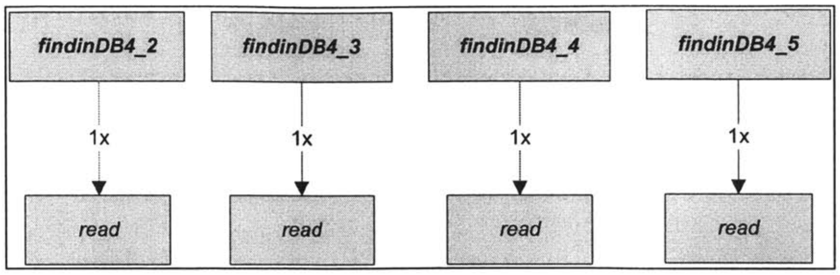


Figure 3-27. CIA functions called by the data-block information extraction functions. The above figure is for the five line data-block.

The final set of CIAs capture general scanning of the display and the detection and reporting of alerts (detecting the yellow dot). These include the functions *scanpath_high, scanpath_low*, *seedot,* and *report.* The charts and algorithms for these CIA functions are included in Appendix B.

This section has presented and explained representative CIA functions that were written to reflect the tasks outlined in the CTA. These algorithms describe the interaction processes which the human carries out when interacting with an interface and carrying out a certain set of tasks. It is from these that the complexity of each of the data-block configurations can be estimated within the context of the overall human machine interaction complexity.

3.3.1.3 Cognitive interaction algorithm assumptions

This section outlines several assumptions which were made in the expression of the CIA. The first assumption is that of a nominal scenario. In each scenario a given set of events take place, each with a given frequency. These events are nominal in that they occur when the task is carried out correctly, without any need for additional aircraft vectoring commands, or unnecessary clicks within data-blocks. Thus stable conditions are assumed for this complexity analysis.

The second assumption is that the resulting algorithms that approximate the human cognitive interaction are purely deterministic. In reality, a human executing a task has stochastic elements. This deterministic assumption however does not eliminate the value of the complexity analysis as it applies across the conditions being compared, and therefore relative complexity estimates are not significantly affected.

There are numerous other assumptions concerning the way the humans perform the task and perceive information. One assumption is that the operator knows where a specific type of information is located within a data-block. This assumption is valid as operators were trained and tested to ensure they memorized the data-block elements. A further assumption is that the eye fixation always tracks every purposeful cursor movement. In addition it is assumed the commands are issued appropriately, thus conflict avoidance vectoring is not needed. When scanning for a specific flight number, it is assumed that the operator usually checks around 50% of the aircraft on screen, before finding the desired target.

When typing a response or a command, it is assumed that operators can touch type (i.e., no additional eye movements are needed away from the screen). The length of these responses is assumed to be four keystrokes, which accounts for one to clear the edit box, and three to type the actual response. Finally the ordering of the contents (i.e. various function calls) within the algorithm is not important, and it is assumed that there is no interaction between the sub-tasks. What is important is the number of function calls within each scenario.

In conclusion, the validity of these assumptions can only truly be assessed by the adequacy of the resulting complexity measure, and replication of this methodology on a different task and interface. The assumptions keep the algorithms reasonably simple, and allow greater focus to outlining the general methodology rather than the details of each CIA. These assumptions are not expected to significantly affect the results, as all assumptions hold across all conditions.

3.4. Step Three: Estimation of complexity

As presented in Chapter 2 the complexity of human machine interaction is defined as *the minimum amount of information required to describe the human-machine interaction process in some fixed description language and chosen level of detail.* The language of the CIA has now been presented and the selected level of detail of the description of the interaction process is established for an example case. As discussed previously, human machine interaction process X is more complex than human machine interaction process Y, if it requires more information to fully describe X than Y (in some fixed description language and level of detail). Between competing interfaces, the least complex configuration (relative to a nominal task) is that which results in the interaction process described in the least amount of information.

Based on the definition, complexity is therefore estimated from 1) the algorithmic information content (AIC) and also 2) the compressed information content of the algorithms. The AIC is computed from the size of the file containing the CIA function. The compressed information content approximates the MDL of the interaction. MDL is hypothetically the better complexity estimate because it is most compatible with the definition of complexity, as the compression removes redundancy from a CIA description. The compression algorithm used is that of the standard Windows® compression software. The units of complexity are bytes.

This paragraph describes the general method by which complexity is estimated for a certain configuration. The complexity estimate of each CIA function is multiplied by the number of times that the function is called during whatever nominal task forms the context in which the interface is used. The complexity estimate is computed from Equation 3-1.

$$\kappa_{human_j} = a_j + \sum_{i=1}^{m} n_i \kappa_i \qquad \text{(E.3-1)}$$

In Equation 3-1 κ_{human_j} is the complexity estimate of the j^{th} interface and task configuration. The term a_j is the complexity of the base CIA which scripts the nominal task, and which calls a number of other CIA functions. This term can be neglected in practice, as in most cases it should be substantially smaller than the summation. The term n_i corresponds to the total number of times which the i^{th} CIA function is called. The term κ_i is the complexity of the i^{th} CIA function and finally m is the total number of CIA functions called. It is of interest to note the similarity of the above formula with Shannon's equation for the entropy of a discrete message space presented in Chapter 2 (Equation 2-1). Instead of probabilities the complexity formula developed in this thesis uses counts (n_i) and the complexity term (κ_i) replaces the term representing the self-information of a message.

The complexity and algorithmic execution times for the CIA functions of the representative ATC task and interface are shown in Table 3-8. The tabulated AIC and

MDL values enter Equation 3-1 through the complexity term (κ_i). The algorithmic execution times correspond to rough approximations subtask duration, or in other words how long it takes a typical human operator to execute the CIA function. The execution times are estimated from averaging measurements taken from only single participant.

Table 3-8. Summary of CIA functions with complexity estimates for each function and algorithmic execution times.

		κ_i		
CIA Function (i)	**Brief Description**	**Algorithmic Information Content (bytes)**	**MDL of Algorithmic information content (bytes)**	**Algorithmic Execution Time Estimate (seconds)**
command_hdg	operator commands a new aircraft heading	522	331	7
command_alt	operator commands a new aircraft altitude	487	314	7
command_spd	operator commands a new aircraft speed	483	312	7
selectAC	operator selects a new aircraft to direct	158	158	3
new_question_inf	detects new question and gathers objective information	108	108	N/A
find_flight_low	find a given flight on the radar display with a low aircraft count	372	263	3
find_flight_high	find a given flight on the radar display with a high aircraft count	699	282	4
scanpath_low	scan between aircraft data-blocks on a radar with a low aircraft count	151	151	4
scanpath_high	scan between aircraft data-blocks on a radar with a high aircraft count	292	211	6
submit	submit a question response via data-link	173	173	4
seedot	detect the appearance of an alert besides a data-block	53	53	N/A
read	read a portion of text	77	77	N/A
report	report alert confirmation	55	55	0.8
move_eyes_to	move eyes to a target onscreen	61	61	N/A
move_hand_to	move hand to a different position	64	64	N/A
move_cursor_to	move cursor to a target onscreen	67	67	N/A
clickbutton	click the mouse button	40	40	0.2
findinDB2_2	find information from 2nd line of 2 Line data-block	111	111	0.8
findinDB2_3	find information from 3rd line of 2 Line data-block	192	192	2.2
findinDB2_4	find information from 4th line of 2 Line data-block	239	239	3.1
findinDB2_5	find information from 5th line of 5 Line data-block	286	259	4.7
findinDB3_2	find information from 2nd line of 3 Line data-block	203	203	0.8
findinDB3_3	find information from 3rd line of 3 Line data-block	202	202	1.1
findinDB3_4	find information from 4th line of 3 Line data-block	192	192	2.7
findinDB3_5	find information from 5th line of 3 Line data-block	239	239	3.1
findinDB4_2	find information from 2nd line of 4 Line data-block	294	245	0.8
findinDB4_3	find information from 3rd line of 4 Line data-block	294	245	1
findinDB4_4	find information from 4th line of 4 Line data-block	294	245	1.5
findinDB4_5	find information from 5th line of 4 Line data-block	185	185	2.3
findinDB5_2	find information from 2nd line of 5 Line data-block	388	258	0.9
findinDB5_3	find information from 3rd line of 5 Line data-block	388	258	1.2
findinDB5_4	find information from 4th line of 5 Line data-block	388	258	1.3
findinDB5_5	find information from 5th line of 5 Line data-block	388	258	1.1
type	type a string of characters into an edit box	146	146	2.5
press	press a specific button on the keyboard with a finger	35	35	0.3
motor	neural motor command	17	17	0.3
angle	mental geographic angle computation between 2 points	25	25	1

The complexity estimation is applied to the competing data-block configurations using the data from Table 3-8. It is assumed that the nominal task consists of directing ten aircraft, responding to sixteen questions and confirming four alerts. These sixteen questions request information contained in the data-blocks. The information required from the questions is evenly distributed between the bottom four lines. This means that four questions request information contained in the second line, four questions request information contained in the third line, and so forth. A base CIA is written for each of the four configurations. The base CIA contains the highest level description of the interaction process. The chart of the base CIA for the two-line data-block is shown in Figure 3-28. The charts for the other three configurations and all base algorithms are included in Appendix B. The complexity is computed by Equation 3-1. The resulting complexity estimates for the four data-block variants are presented in Table 3-10.

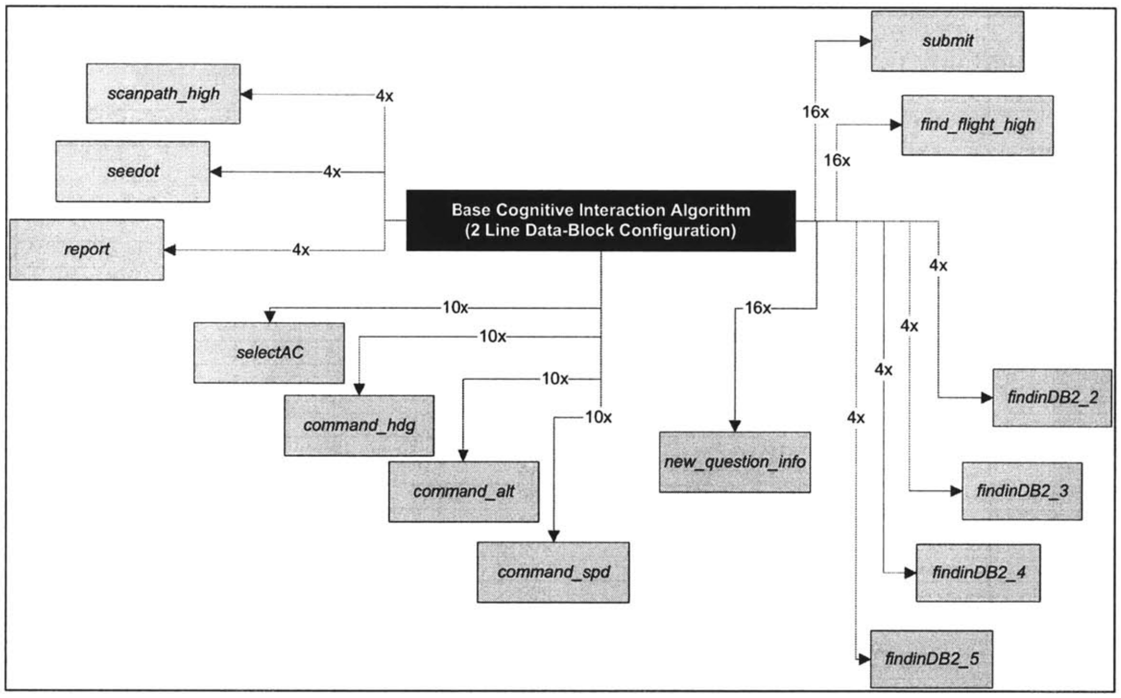


Figure 3-28. Structure of base CIA for the two-line data-block configuration. The numbers on the lines indicate the number of times each CIA function is called from the base algorithm.

Based on the assumed nominal task and sequence of events the cumulative number of executions of each CIA function (n_i) can be computed for each interface configuration j. For this example the number of calls, n of each i^{th} CIA function are shown in Table 3-9. This is the final piece required for the complexity to be computed by Equation 3-1. The resulting complexity estimates for the four data-block variants are presented in Table 3-10.

Table 3-9. Number of calls of each CIA function for example configurations. These cells include the n_i values that enter the complexity equation (Equation 3-1).

CIA function (i)	Data-Block Type (j configurations)			
	2 Line	3 Line	4 Line	5 Line
command_hdg	10	10	10	10
command_alt	10	10	10	10
command_spd	10	10	10	10
selectAC	10	10	10	10
new_question_info	16	16	16	16
find_flight_high	16	16	16	16
scanpath_high	4	4	4	4
submit	16	16	16	16
seedot	4	4	4	4
read	158	158	158	158
report	4	4	4	4
move_eyes_to	1006	990	978	970
move_hand_to	272	260	252	160
move_cursor_to	180	168	160	156
clickbutton	180	168	160	156
findinDB2_2	4	0	0	0
findinDB2_3	4	0	0	0
findinDB2_4	4	0	0	0
findinDB2_5	4	0	0	0
findinDB3_2	0	4	0	0
findinDB3_3	0	4	0	0
findinDB3_4	0	4	0	0
findinDB3_5	0	4	0	0
findinDB4_2	0	0	4	0
findinDB4_3	0	0	4	0
findinDB4_4	0	0	4	0
findinDB4_5	0	0	4	0
findinDB5_2	0	0	0	4
findinDB5_3	0	0	0	4
findinDB5_4	0	0	0	4
findinDB5_5	0	0	0	4
type	46	46	46	46
press	184	184	184	184
motor	1642	1602	1574	1558
angle	10	10	10	10

Table 3-10. Data-block complexity estimates computed by Equation 3-1.

Data-block Configuration	Algorithmic Information Content (bytes)	Minimum Description Length (bytes)
2 Line	191140	176638
3 Line	187464	172590
4 Line	185812	170350
5 Line	186308	169354

Figure 3-29 plots the complexity estimates for each data-block type. Both AIC and MDL are shown. Based on the MDL, the results show that the complexity of the 5-line data-block is the lowest. The AIC shows that the complexity of the 4-line data-block is the lowest. In reality however, the maximum complexity differential between data-blocks is only 0.04%. This suggests that the data-block configurations do not differ in complexity within the context of the nominal task.

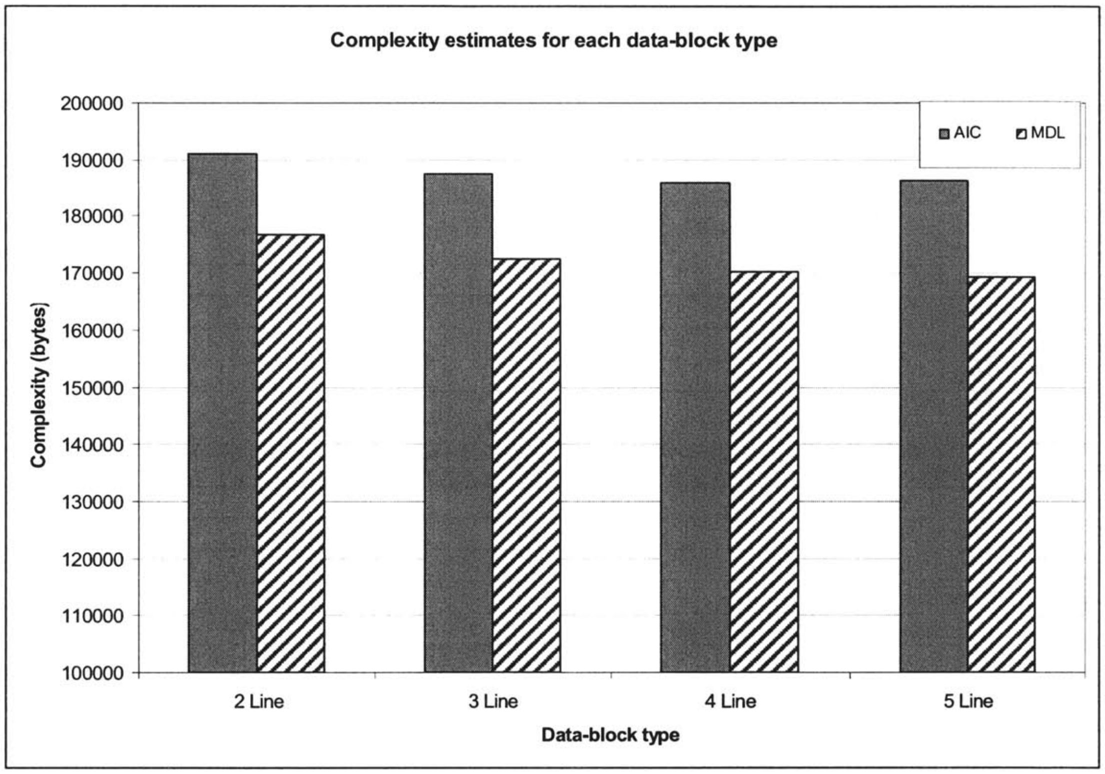


Figure 3-29. Complexity estimates for each interface configuration.

Between AIC and MDL, the more appropriate complexity estimate is MDL. This is because the compression removes redundancy and approaches a *minimum* amount of information. This is compatible with the chosen definition of complexity in human machine interaction. However in this case the Pearson correlation between MDL and AIC is 0.97 (p = .031). Therefore these two metrics are essentially measuring the same construct, and while in this case either can be used, the remaining discussions of complexity will focus on MDL, since it captures minimum information more closely.

Based on the ATC task context presented in this section it is found that the five line data-block is the least complex, however it differs in complexity by only 0.04% from the most complex data-block. In the following two chapters additional task factor levels are introduced and the complexity results are validated against human performance results obtained in experimental trials.

3.5. Conclusions

In this chapter, the methodology for computing complexity of human machine interaction is established. The methodology is intended for designers and evaluators who need to select between completing interface and/or task variants based on minimum complexity. This includes the first step of conducting a CTA to elicit the mechanisms with which operators interact with a display and process information as they execute specific tasks. CIA functions are then written based on the processes identified. Each CIA function has an associated AIC and MDL. A base CIA is written for each configuration being compared. The base CIA is the highest level program from which other CIA functions that represent subprocesses are called. The total complexity of a given configuration is computed relative to a nominal task, by multiplying the cumulative number of CIA function calls by the complexity of each function (using Equation 3-1).

ATC data-block configurations are used as an illustrative example, and the method indicates that these four sample configurations differ in complexity by only 0.04%, with the five-line data-block being the least complex. The complexity of the interface variants is estimated within the context of an ATC task of directing ten aircraft, responding to sixteen questions and confirming four alerts. In Chapter 5 the CIA complexity estimation methodology is applied within the context of two additional task factors, and the complexity estimates are validated against human performance results obtained in experimental trials and presented in Chapter 4.

Page intentionally left blank

4. Human Performance Results

The validity of the resulting complexity measure cannot be accurately assessed without comparing complexity to performance results. More complex human-machine interaction, which may be the result of more complex displays, should result in degraded human performance. Therefore this chapter discusses the experiment conducted to investigate the performance consequences of different interface (data-block) and task configurations. Chapter 5 compares the complexity estimation results to the experimental performance results in order to determine whether the complexity metric methodology is an adequate predictor of operator performance and workload.

4.1. Experiment Motivation

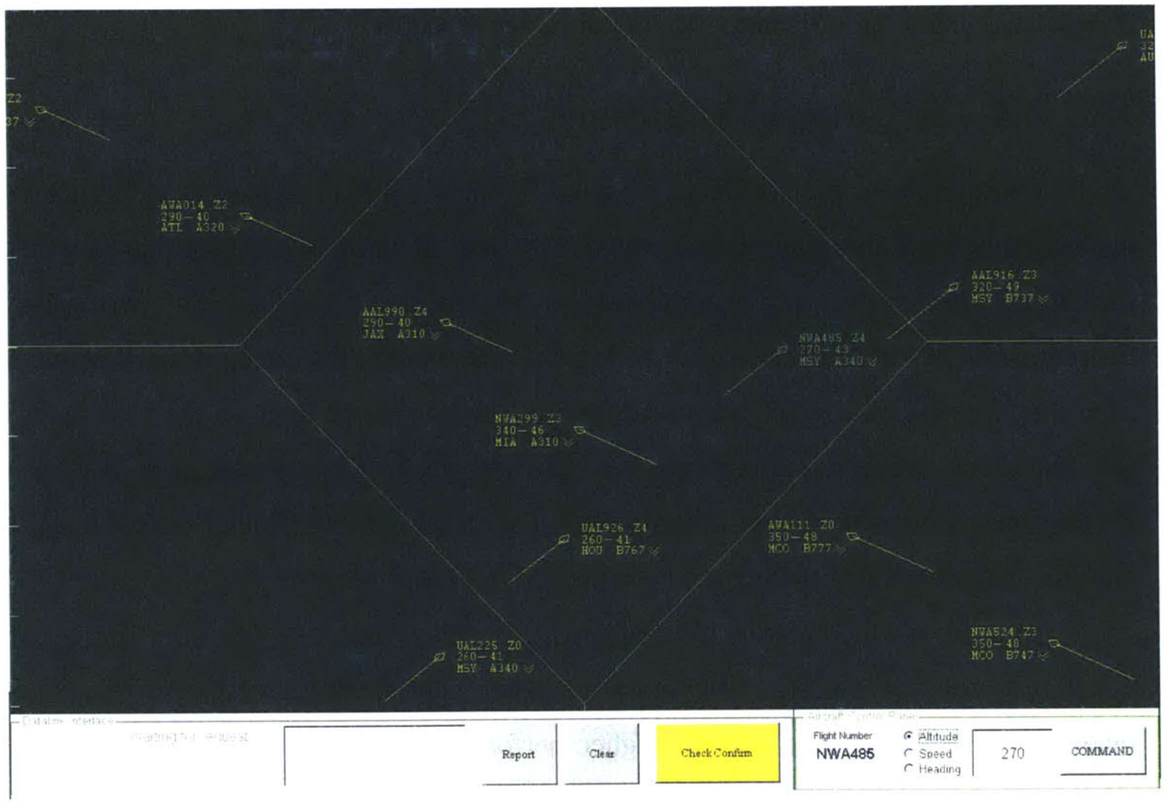
As mentioned previously in Chapter 3, the aircraft data-block is a key display element in which information is presented to the operator, yet which has not been researched extensively. This experiment was conducted to investigate the performance consequences of four different data-block types; specifically the configurations shown in Figure 3-2. It was of interest to investigate how the number of lines on the base layer of the data-block affected performance on the ATC task. The hypothesized drivers of performance were the clutter imposed by the different data-blocks and the action requirements to extract information from the different data-blocks.

In order for the complexity analysis methodology to be successful, experimental conditions with low performance measures should consistently have higher complexity than experimental conditions with higher performance measures. In other words the relationship between complexity and performance scores should be inverse, while the relationship between complexity and secondary task response times should be direct. Within the context of this thesis, the experimental results reported are used to determine if the relationship between the complexity estimates and performance is correct.

4.2. Method

4.2.1. Apparatus

A human-in-the-loop simulation ATC interface was programmed in Matlab® and intended to emulate a Display System Replacement (DSR) type interface used in en-route ATC operations and shown in Chapter 3. The simulation interface is called the Reprogrammable Air Traffic Experimental Testbed (RATE). A screenshot of RATE is shown in Figure 4-1. The functionality of the interface is discussed in depth in Chapter 3 and therefore is not repeated here.



AWA014 Z2 290-40 ATL A320
AAL990 Z4 290-40 JAX A310
NWA485 Z4 270-43 MSY A340
AAL916 Z3 320-49 MSY B737
NWA299 Z3 340-46 MIA A310
UAL926 Z4 260-41 HOU B767
AWA111 Z0 350-48 MCO B777
UAL225 Z0 260-41 MSY A340
NWA524 Z3 350-48 MCO B747
Report
Clear
Check Confirm
Flight Number
NWA485
Altitude
Speed
Heading
270
COMMAND

Figure 4-1. The RATE interface displaying a screenshot from a scenario with high aircraft count and three line data-block. Flight NWA485 is selected.

Training and testing were conducted on three identical Fujitsu tablet PC computers connected to external Dell® 19 inch monitors, external keyboards and mice. The resolution was set at 1024x768 pixels with 16 bit color. The laptops ran on a Pentium 1.73 gigahertz processor with 1GB of random access memory (RAM). Matlab® version

7.0.1 was used to run the simulation. During testing, user responses were recorded in a single file for each scenario. After signing required consent forms, subjects completed a detailed tutorial presentation that discussed the nature of the experiment, and explained the task and the software usage. Subjects were then quizzed to ensure that they had memorized the meaning of each data-block information entry[15]. If they made errors, they were corrected prior to continuing with the experiment. Subjects then completed a single five minute practice scenario before beginning the four test sessions, which lasted nine minutes each. After completing all sessions, they completed a brief demographic survey.

4.2.2. Participants and procedure

The subjects consisted of 23 Navy ATC trainees. For the base en-route ATC task controllers had to observe the incoming traffic, determine from which of the four possible egress directions each flight had to exit from, and command the altitude and velocity required for that particular egress. The egress label for a particular flight was displayed on the first line of the data-block, adjacent to the flight number. Subjects also had to avoid conflicts, and had the option of bringing up three nautical mile diameter circles around each aircraft. Although controllers were formally in charge of the central diamond sector (shown in Figure 4-2), as soon as an aircraft appeared anywhere on screen, controllers could begin to issue commands immediately. In other words, every aircraft appearing on screen had been automatically handed off to them. To command the aircraft, controllers had to first click on the aircraft symbol (diamond or heading vector) and then use the control panel that appeared on the bottom right hand side of the screen. As the commands were issued by data-link, aircraft began maneuvering almost immediately at rates consistent with current commercial aircraft climb, acceleration and turn rates. These were derived from typical g-forces that passengers experience in flight and adjusted based on available turn rate data. Furthermore all aircraft were fully compliant with commands, although maneuver rates included small random variations. The names, locations and exit requirements of the four egresses are shown in Figure 4-2.

[15] The data-block entries were discussed and labeled in Chapter 3, Figure 3-9.

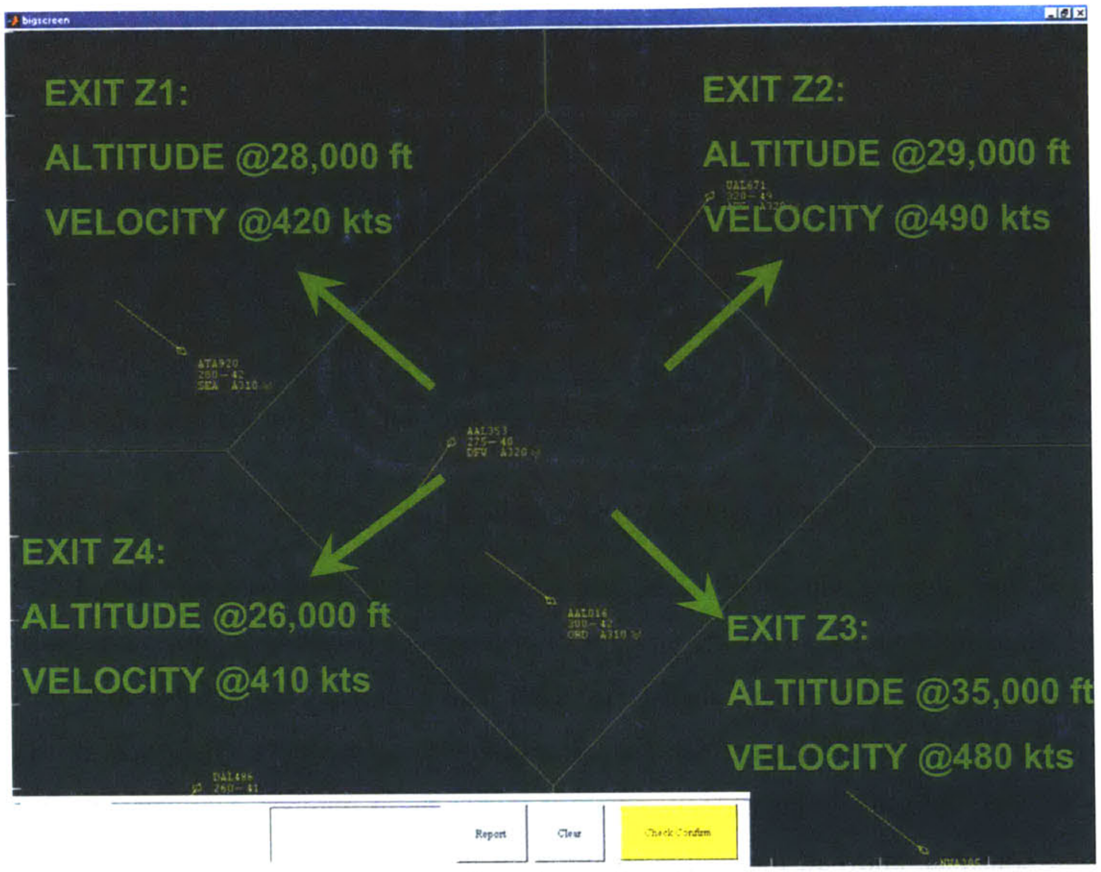


Figure 4-2. Sector exits and rules. An aircraft entering through one of the four quadrants was required to exit through one of the other three, at the altitudes and velocities shown above.

The secondary task consisted of answering a series of sixteen questions per scenario. These questions were equally spaced in time (31 seconds apart) and concerned with information about a specific flight. The question type was randomly preset within the sequences prior to the experiment and did not change between subjects. However the specific aircraft, with which the question was concerned with, was selected randomly during the scenario.

4.2.3. Experimental design

There were three independent variables in this experiment. The first was the type of data-block, the second was the frequency of information access, otherwise known as question distribution, and the third was the aircraft count. The combinations of these three variables formed sixteen experimental scenarios. Each scenario consisted of a specific interface and task configuration.

The independent variables are explained and discussed in greater detail in the following paragraphs and are summarized in Table 4-1. As introduced in the previous chapter, the independent variable of data-block type had four levels (Figure 3-1, Chapter 3). Every data-block level contained identical information, however the four factor levels varied by how much information was presented in the single base layer. The purpose of this factor was to determine whether any particular data-block resulted in the best performance and if so under which task conditions.

Table 4-1. Independent variables and factor levels

1	**Data-block Type**
→	Two line data-block
→	Three line data-block
→	Four line data-block
→	Five line data-block
2	**Question Distribution**
→	Equal number of questions per line
→	Spread number of questions per line
3	**Number of aircraft**
→	Low count (~5-6 aircraft)
→	High count (~11-12 aircraft)

The second independent variable concerned the question distribution of the secondary task. There were two factor levels, *equal* and *spread.* For the first level, subjects were asked an equal number of questions (four) about each of lines two through five in the data-block. The second level asked a far greater number of questions about the second line (ten) and far fewer for the other three data-block lines (six in total), thus this is termed the *spread* factor. Table 4-2 summarizes the consequences of the question distribution variable for each data-block. The distribution changed the level of interaction required, by increasing the number of mouse clicks required to access information.

The final independent variable was the number of aircraft and under low aircraft counts approximately 5-6 aircraft were on the display at a given time as opposed to 11-12 at high counts.

Table 4-2. Percentage of time that operators have to click in the data-blocks in order access a piece of information and number of questions per data-block line under each question distribution level.

Two Line Data-block Question Distribution Level	Interaction required	Number of questions per data-block line # 1	# 2	# 3	# 4
Equal	75% of the time (25%-1 click, 25%-2 clicks, 25%-3 clicks)	4	4	4	4
Spread	38% of the time (19%-1 click, 13%-2 clicks, 6%-3 clicks)	10	3	2	1
Three Line Data-block Question Distribution Level	Interaction required	Number of questions per data-block line # 1	# 2	# 3	# 4
Equal	50% of the time (25%-1 click, 25%-2 clicks)	4	4	4	4
Spread	19% of the time (13%-1 click, 6%-2 clicks)	10	3	2	1
Four Line Data-block Question Distribution Level	Interaction required	Number of questions per data-block line # 1	# 2	# 3	# 4
Equal	25% of the time (25%-1 click)	4	4	4	4
Spread	6% of the time (6%-1 click)	10	3	2	1
Five Line Data-block Question Distribution Level	Interaction required	Number of questions per data-block line # 1	# 2	# 3	# 4
Equal	0% of the time No clicks ever required	4	4	4	4
Spread	0% of the time No clicks ever required	10	3	2	1

The experiment therefore consisted of a 4x2x2 mixed factorial design. The first independent variable was administered between subjects and the other two independent variables were administered within subjects. Scenarios were counterbalanced to control for practice effects. Multiple dependent variables were captured to determine the performance of subjects due to the influence of the independent variables on the various tasks. The dependent measures gathered and used in this thesis are summarized in Table 4-3 and discussed in the subsequent paragraph. The complexity estimates computed for each scenario are compared with respect to the dependent variables.

Table 4-3. Summary of dependent variables

Dependent Measures		
1	**Egress Fraction**	Percentage of correct egresses from sector. This fraction is given by Equation 4-1.
2	**Question Response Accuracy**	Percentage of correct answers to questions of secondary task.
3	**Mean Question Response Time**	Mean response time in seconds to correct answers to questions of secondary task
Note: All measures are per scenario		

In terms of performance on the primary en-route vectoring task, algorithms detected if an aircraft left the sector at the correct location, altitude and velocity. Post-experimental analysis determined the actual maximum number of correct egresses. Thus egress fraction (*EF*) consists of the sum of the number of correct egress locations (*e*), correct egress altitudes (*a*) and correct egress velocities (*v*), divided by three times the maximum number of possible correct egresses (*T*) within a scenario. This relationship is shown in Equation 4-1.

$$EF = \frac{e+a+v}{3T} \tag{4-1}$$

In addition, the correctness of the question responses submitted through the data-link, as well as the answer times of correct responses, were recorded.

4.3. Results

Statistical analyses were conducted to determine the effect of the independent variables on each of the dependent variables. The sub-sections that follow include the analyses and results for each of the dependent variables.

4.3.1. Egress fraction

This dependent measure is an objective assessment of the performance on the primary en-route air traffic control vectoring task. To analyze the egress fraction variable, a three factor mixed analysis of variance (ANOVA) test is applied. This analysis uncovers whether there are significant main effects due to the independent variables as well as the significant interactions between any combination of the three independent variables of data-block type (between subjects), question distribution (within subjects) and aircraft count (within subjects).

Data-block type is found not to be a statistically significant factor ($F(3,19) = 1.91$, $p = .16$) of egress fraction. The trend however indicates that performance decreases between the three-line and four-line data-block, as seen in Figure 4-3.

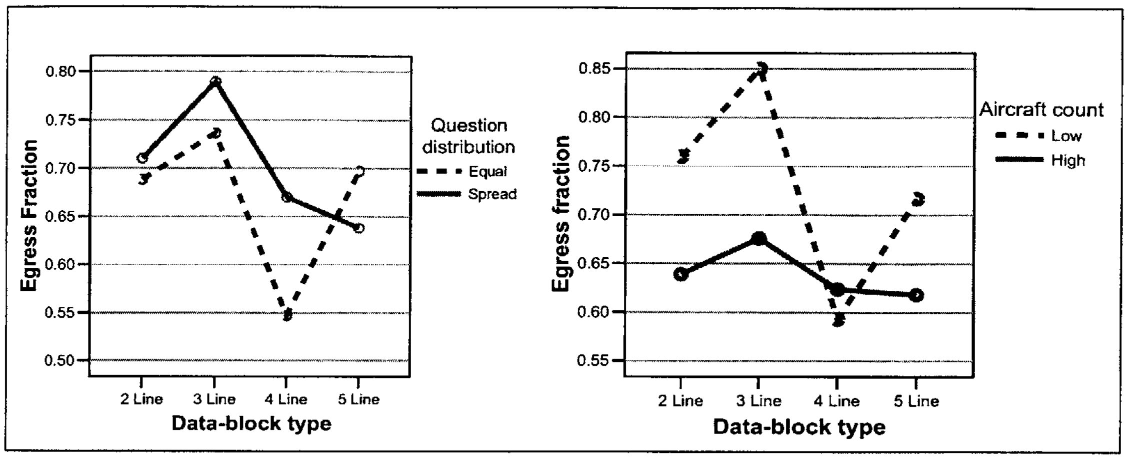


Figure 4-3. Performance on base ATC task across data-block types. Shown for each aircraft count (right) and question distribution level (left).

As expected, the number of aircraft is highly significant ($F(1,19) = 9.46, p = .006$) in determining performance on the base task. This corroborates previous studies that have reported aircraft count as the primary source of controller workload and cognitive complexity (Hilburn, 2004; Kopardekar, 2003; Majumdar & Ochieng, 2002). With the higher number of aircraft, the vectoring task becomes more difficult and performance drops significantly. Furthermore, the interaction between question distribution and number of aircraft is also statistically significant ($F(1,19) = 14.71, p = .001$). The *equal* question distribution factor level adds to the overall workload driven primarily by aircraft count, by imposing a greater level of interaction than the *spread* level. This means that with high aircraft count and an equal question distribution, there is a significant drop in primary task performance.

4.3.2. Question responses

In this section two variables are analyzed: question response accuracy and response time to correct answers. Question response accuracy measures the cumulative count of correct responses to the questions of the secondary task. Reduced response accuracy to a secondary task can be an indicator of increased workload (Vidulich, 2003). Therefore different response accuracies across data-blocks could be interpreted to signify that the data-block types affect workload. As the question response data is not interval data, a non-parametric Kruskal-Wallis test is conducted. For the effect of data-block type, the chi-square statistic of the ranks is not significant ($\chi^2(3, N = 22) = .41, p = .94$). Therefore question response accuracy does not significantly depend upon data-block type in the representative ATC task. This can be interpreted to signify that data-block type does not affect operator workload. The mean numbers of correct responses for each data-block type are plotted in Figure 4-4.

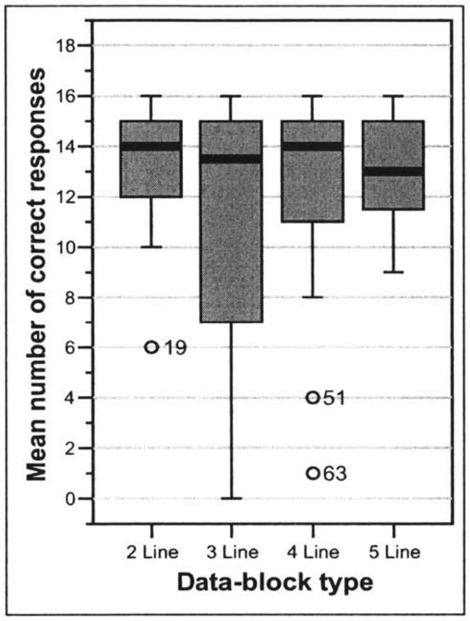


Figure 4-4. Box plot of mean number of correct responses for each data-block type.

The next independent variable is the question distribution factor. In order to determine whether question distribution factor affected accuracy, a Wilcoxon Signed Ranks test is performed. These results show that question distribution is not significant ($Z = -.51, p = .61$). Therefore changing the amount of interaction with the data-block does not affect question response accuracy.

The third factor is the number of aircraft. The significance of the number of aircraft factor is also examined with a Wilcoxon Signed Ranks test. This shows that as expected the number of aircraft factor is highly significant ($Z = -4.41, p = .000$). This result also is evidence to support that secondary task question response accuracy is a valid measure of workload in an ATC task (the measure captures the known workload driver of number of aircraft). A plot of mean correct responses versus number of aircraft is shown in Figure 4-5.

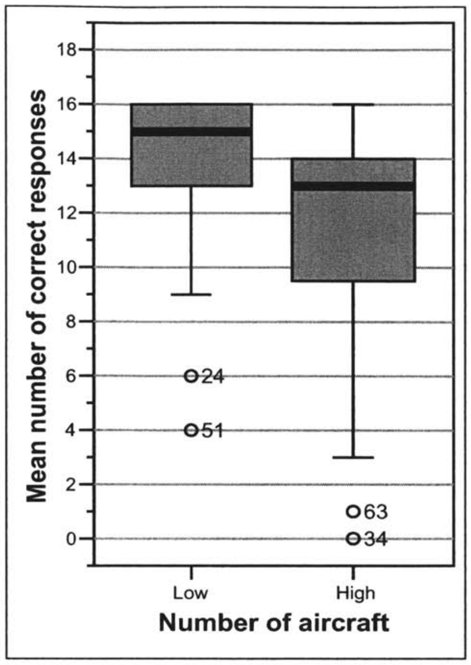


Figure 4-5. Box plot of question response accuracy versus number of aircraft

The second dependent measure analyzed in this section is the mean response time to correct answers. Secondary task reaction times are also an indirect measure of spare mental capacity and thus give insight to mental workload (Vidulich, 2003). As this response time data is on an interval scale, a mixed 4x2x2 ANOVA is carried out in order to elucidate the significant main effects and interactions. It is found that the data-block factor is not statistically significant in affecting the mean question response times ($F(3,17) = .24$, $p = .87$). The estimated marginal means of question response time are plotted in Figure 4-6 for each data-block type, at each of the two levels of aircraft count. For high aircraft counts the trend of data-block type appears flat. For low aircraft counts there is an apparent decrease in response time with increasing number of data-block base layer lines.

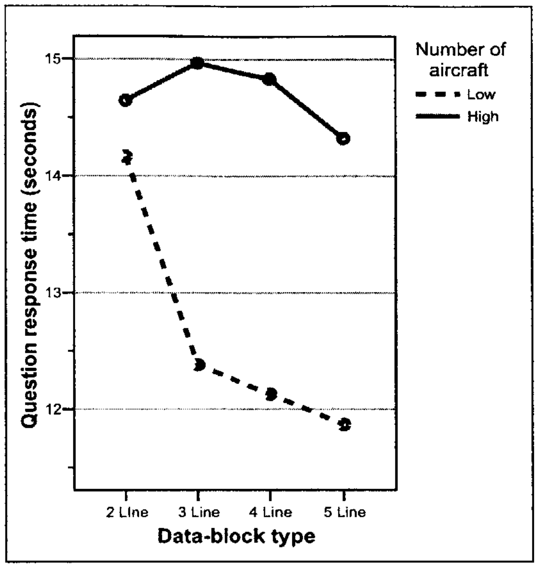


Figure 4-6. Estimated question response times for each data-block type at two levels of aircraft count.

The question distribution factor is significant ($F(1,17) = 5.17, p = .036$). An equal task distribution results in higher response times. The number of aircraft factor is also significant in determining question response time ($F(1,17) = 17.84, p = .001$). Since this is an expected result, this corroborates the use of secondary task reaction time as a measure of workload.

4.4. Conclusions

In the experiment, four data-block designs are compared along with two other factors: question distribution and number of aircraft. The impacts of these factors on operator performance are quantified and analyzed. The results indicate that the data-block type does not have a significant effect on any of the dependent measures. For the primary ATC task performance results (even though the results are not statistically significant), a pattern of decreased performance is observed with increasing data-block base layer lines. The question response time shows a decreasing trend with increased data-block base layer lines. Because the patterns in the performance trends differ, and the absence of statistical significance, it is concluded that the data-block type has little effect on the performance and workload of the operator. The question distribution factor is significant across secondary task response time and as expected the number of aircraft is found to be significant across all dependent measures.

Page intentionally left blank

5. Complexity Metric Validation

This chapter formulates a complexity metric for each experimental scenario from Chapter 4 and compares the estimated complexity (computed by the methods introduced in Chapter 3) to the performance results. The complexity results are normalized and compared to the normalized experimental results in order to determine whether the complexity measure can be predictive of operator performance and workload. These results in part determine the effectiveness of the complexity analysis methodology proposed in this thesis and help determine the course of future research.

5.1. Experimental Scenario Cognitive Interaction Algorithms

The experimental factors determine the interface configuration, task and nominal sequence of events that occur in each scenario, and Table 5-1 shows the correspondence between scenario numbers and experimental factors. This table also shows the nominal counts of the events that occur within each scenario[16]. These are used to determine the number of CIA function calls for each scenario (the n_i values in Equation 3-1). The CIA functions are first presented and described in Chapter 3. Each experimental scenario requires the execution of a specific set of CIA functions a nominal number of times from within the base CIA.

Each scenario has a base CIA from which all the functions are called[17]. Rather than presenting the algorithms for each of the sixteen scenarios, the CIA for scenario 112 is presented and used as a benchmark example. This is the scenario with the two line data-block, an equal question distribution and high aircraft count. The structure of this algorithm is shown in Figure 5-1 and includes the number of each of the CIA function calls. The remaining fifteen scenario CIA codes and flow charts are contained in Appendix C.

[16] The nominal counts of events are obtained by running the scenario with the assumption that all aircraft that can potentially be directed are in fact directed, and in a manner which avoids all conflicts.
[17] The simple concept of the base (or core) CIA is discussed in Chapter 3.

Table 5-1. Scenario code legend specification and nominal event decomposition per scenario

SCENARIO SPECIFICATION				*EVENT DECOMPOSITION*					
Scenario Code	**Data-Block Type**	**Question Distribution**	**Aircraft Count**	**Number of clicks to expand data-block**	**Altitude Commands**	**Speed Commands**	**Heading Commands**	**Number of questions**	**Number of alerts**
111	2 Line	Equal	Low	24	7	7	5	16	4
112	2 Line	Equal	High	24	12	12	7	16	4
121	2 Line	Spread	Low	10	7	7	4	16	4
122	2 Line	Spread	High	10	10	12	6	16	4
211	3 Line	Equal	Low	12	7	7	5	16	4
212	3 Line	Equal	High	12	12	12	7	16	4
221	3 Line	Spread	Low	4	7	7	4	16	4
222	3 Line	Spread	High	4	10	12	6	16	4
311	4 Line	Equal	Low	4	7	7	5	16	4
312	4 Line	Equal	High	4	12	12	7	16	4
321	4 Line	Spread	Low	1	7	7	4	16	4
322	4 Line	Spread	High	1	10	12	6	16	4
411	5 Line	Equal	Low	0	7	7	5	16	4
412	5 Line	Equal	High	0	12	12	7	16	4
421	5 Line	Spread	Low	0	7	7	4	16	4
422	5 Line	Spread	High	0	10	12	6	16	4

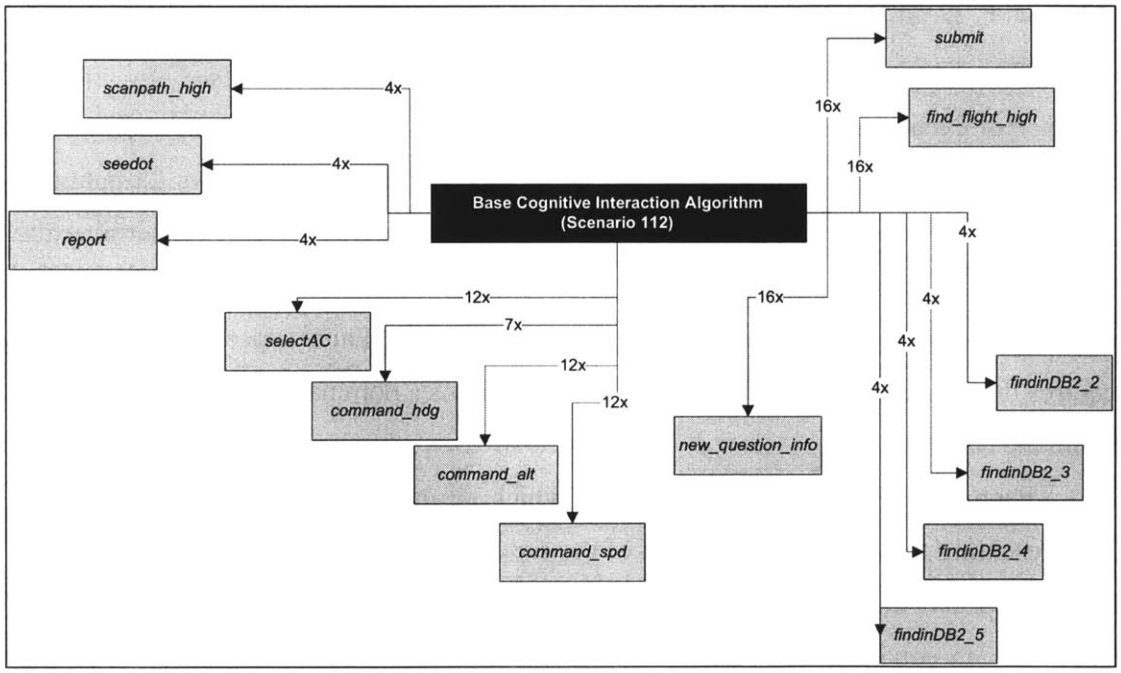


Figure 5-1. CIA functions called by scenario 112. Numbers denote the frequency with which each of the functions is called.

The functions called by the base CIA for scenario 112 have been presented and described in Chapter 3. The base CIA for scenario 112 is shown on the following page and essentially scripts, at a high level, the nominal events that occur in the scenario. For clarity, annotations are made beside the code. These events are not necessarily described in sequential order, and do not need to be, as this does not influence the resulting metric.

```
1.     ac1 = selectAC
2.     command_alt(a1)
3.     command_spd(a1)
4.     ac2 = selectAC
5.     command_hdg(a2)
6.     command_alt(a2)
7.     command_spd(a2)
8.     ac3 = selectAC
9.     command_alt(a3)
10.    command_spd(a3)
11.    ac4 = selectAC
12.    command_alt(a4)
13.    command_spd(a4)
14.    ac5 = selectAC
15.    command_hdg(a5)
16.    command_alt(a5)
17.    command_spd(a5)
18.    ac6 = selectAC
19.    command_hdg(a6)
20.    command_alt(a6)
21.    command_spd(a6)
22.    ac7 = selectAC
23.    command_hdg(a7)
24.    command_alt(a7)
25.    command_spd(a7)
26.    ac8 = selectAC
27.    command_hdg(a8)
28.    command_alt(a8)
29.    command_spd(a8)
30.    ac9 = selectAC
31.    command_alt(a9)
32.    command_spd(a9)
33.    ac10 = selectAC
34.    command_hdg(a10)
35.    command_alt(a10)
36.    command_spd(a10)
37.    ac11 = selectAC
38.    command_alt(a11)
39.    command_spd(a11)
40.    ac12 = selectAC
41.    command_alt(a12)
42.    command_hdg(a12)
43.    command_spd(a12)
```

The first portion (lines 1-43) consists of the calls to functions concerned with the direction of aircraft in either altitude, speed or heading. The first line of code calls the *select_ac* function which selects an aircraft based on proximity to the exit line, and also one that has not been selected previously. This aircraft is assigned a short term memory variable (called *ac1* in this algorithm) and then the required commands are carried out on that particular aircraft.

```
44.    q1 = new_question_info
45.    targetAC = find_flight_high(q1(1))
46.    ans = findinDB2_5(targetAC,q1(2))
47.    submit(ans)
48.    q2 = new_question_info
49.    targetAC = find_flight_high(q2(1))
50.    ans = findinDB2_3(targetAC,q2(2))
51.    submit(ans)
52.    q3 = new_question_info
53.    targetAC = find_flight_high(q3(1))
54.    ans = findinDB2_2(targetAC,q3(2))
55.    submit(ans)
56.    q4 = new_question_info
57.    targetAC = find_flight_high(q4(1))
58.    ans = findinDB2_3(targetAC,q4(2))
59.    submit(ans)
60.    q5 = new_question_info
61.    targetAC = find_flight_high(q5(1))
```

Description for lines 44-107 is on the following page.

```
62.     ans = findinDB2_2(targetAC,q5(2))
63.     submit(ans)
64.     q6 = new_question_info
65.     targetAC = find_flight_high(q6(1))
66.     ans = findinDB2_4(targetAC,q6(2))
67.     submit(ans)
68.     q7 = new_question_info
69.     targetAC = find_flight_high(q7(1))
70.     ans = findinDB2_4(targetAC,q7(2))
71.     submit(ans)
72.     q8 = new_question_info
73.     targetAC = find_flight_high(q8(1))
74.     ans = findinDB2_4(targetAC,q8(2))
75.     submit(ans)
76.     q9 = new_question_info
77.     targetAC = find_flight_high(q9(1))
78.     ans = findinDB2_5(targetAC,q9(2))
79.     submit(ans)
80.     q10 = new_question_info
81.     targetAC = find_flight_high(q10(1))
82.     ans = findinDB2_5(targetAC,q10(2))
83.     submit(ans)
84.     q11 = new_question_info
85.     targetAC = find_flight_high(q11(1))
86.     ans = findinDB2_2(targetAC,q11(2))
87.     submit(ans)
88.     q12 = new_question_info
89.     targetAC = find_flight_high(q12(1))
90.     ans = findinDB2_2(targetAC,q12(2))
91.     submit(ans)
92.     q13 = new_question_info
93.     targetAC = find_flight_high(q13(1))
94.     ans = findinDB2_3(targetAC,q13(2))
95.     submit(ans)
96.     q14 = new_question_info
97.     targetAC = find_flight_high(q14(1))
98.     ans = findinDB2_4(targetAC,q14(2))
99.     submit(ans)
100.    q15 = new_question_info
101.    targetAC = find_flight_high(q15(1))
102.    ans = findinDB2_5(targetAC,q15(2))
103.    submit(ans)
104.    q16 = new_question_info
105.    targetAC = find_flight_high(q16(1))
106.    ans = findinDB2_4(targetAC,q16(2))
107.    submit(ans)
```

The following section of code (lines 44-107) consists of the responses to the sixteen questions that appeared. The first part (e.g.: line 76) of each question consists of calling the function *new_question_info*. This function gathers information about the question being asked from the data-link and stores this in a temporary mental variable. The information stored is a target aircraft flight number and the type of information requested about that flight. The next part (e.g.: line 101) runs a search on the radar display to locate the particular flight and stores the position. The following line (e.g.: 102) runs the search for the piece of information from within the data-block. This depends on the data-block type and the line in which the information sought is embedded. The answer is gathered and submitted to the data-link interface as is represented by the *submit* function.

```
108.    scanpath_high
109.    d1 = seedot
110.    if d1 == 1
111.    report
112.    end
113.    scanpath_high
114.    d2 = seedot
115.    if d2 == 1
116.    report
117.    end
118.    scanpath_high
119.    d3 = seedot
120.    if d3 == 1
121.    report
122.    end
123.    scanpath_high
124.    d4 = seedot
```

The final section of code (lines 108-127) consists of the alert detection process and general display scans. It is assumed that in conjunction with such scans each alert is detected. This is a valid assumption as subjects generally failed to detect the alert when fully engaged in a subtask. If the alert dot is active (checked by the *seedot* function), the algorithm calls the report function which describes the simple process of clicking the report button.

```
if d4 == 1
report
end
```

Now that the base CIA has been described in detail for an example scenario, and the information is available to compute the cumulative number of CIA function calls, the following section presents the complexity estimates.

5.2. Scenario Complexity Estimates

Table 5-2 shows the total number of calls of each CIA function from within each of the sixteen scenarios and also includes the complexity estimates of each function.

Table 5-2. CIA functions including complexity estimates, algorithmic execution time estimates and number of calls for each of the sixteen scenarios.

		κ_i																		
CIA Function (i)	Brief Description	Algorithmic information content (bytes)	MDL of algorithmic information content (bytes)	Algorithmic Execution Time Estimate (s)	*Total number of calls for each scenario CIA* (n_i)															
					111	*112*	*121*	*122*	*211*	*212*	*221*	*222*	*311*	*312*	*321*	*322*	*411*	*412*	*421*	*422*
command_hdg	operator commands a new aircraft heading	522	331	7	5	7	4	6	5	7	4	6	5	7	4	6	5	7	4	6
command_alt	operator commands a new aircraft altitude	487	314	7	7	12	7	10	7	12	7	10	7	12	7	10	7	12	7	10
command_spd	operator commands a new aircraft speed	483	312	7	7	12	7	12	7	12	7	12	7	12	7	12	7	12	7	12
selectAC	operator selects a new aircraft to direct	158	158	3	7	12	7	12	7	12	7	12	7	12	7	12	7	12	7	12
new_question_info	detects new question and gathers objective information	108	108	N/A	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
find_flight_low	find a given flight on the radar display with a low aircraft count	372	263	3	16	0	16	0	16	0	16	0	16	0	16	0	16	0	16	0
find_flight_high	find a given flight on the radar display with a high aircraft count	699	282	4	0	16	0	16	0	16	0	16	0	16	0	16	0	16	0	16
scanpath_low	scan between aircraft data-blocks on a radar with a low aircraft count	151	151	4	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0
scanpath_high	scan between aircraft data-blocks on a radar with a high aircraft count	292	211	6	0	4	0	4	0	4	0	4	0	4	0	4	0	4	0	4
submit	submit a question response via data-link	173	173	4	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
seedot	detect the appearance of an alert besides a data-block	53	53	N/A	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
read	read a portion of text	77	77	N/A	99	159	98	156	99	159	98	156	99	159	98	156	99	159	98	156
report	report alert confirmation	55	55	0.8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
move_eyes_to	move eyes to a target onscreen	61	61	N/A	599	923	570	876	583	907	561	867	571	895	556	862	563	887	564	860
move_hand_to	move hand to a different position	64	64	N/A	206	278	186	246	194	266	180	240	186	258	177	237	182	254	186	236
move_cursor_to	move cursor to a target onscreen	67	67	N/A	136	184	118	158	124	172	112	152	116	164	109	149	112	160	118	148
clickbutton	click the mouse button	40	40	0.2	136	184	118	158	124	172	112	152	116	164	109	149	112	160	118	148
findinDB2_2	find information from 2nd line of 2 Line data-block	111	125	0.8	4	4	10	10	0	0	0	0	0	0	0	0	0	0	0	0
findinDB2_3	find information from 3rd line of 2 Line data-block	192	192	2.2	4	4	3	3	0	0	0	0	0	0	0	0	0	0	0	0
findinDB2_4	find information from 4th line of 2 Line data-block	239	239	3.1	4	4	2	2	0	0	0	0	0	0	0	0	0	0	0	0
findinDB2_5	find information from 5th line of 5 Line data-block	286	259	4.7	4	4	1	1	0	0	0	0	0	0	0	0	0	0	0	0
findinDB3_2	find information from 2nd line of 3 Line data-block	203	203	0.8	0	0	0	0	4	4	10	10	0	0	0	0	0	0	0	0
findinDB3_3	find information from 3rd line of 3 Line data-block	202	202	1.1	0	0	0	0	4	4	3	3	0	0	0	0	0	0	0	0
findinDB3_4	find information from 4th line of 3 Line data-block	192	192	2.7	0	0	0	0	4	4	2	2	0	0	0	0	0	0	0	0
findinDB3_5	find information from 5th line of 3 Line data-block	239	239	3.1	0	0	0	0	4	4	1	1	0	0	0	0	0	0	0	0
findinDB4_2	find information from 2nd line of 4 Line data-block	294	245	0.8	0	0	0	0	0	0	0	0	4	4	10	10	0	0	0	0
findinDB4_3	find information from 3rd line of 4 Line data-block	294	245	1	0	0	0	0	0	0	0	0	4	4	3	3	0	0	0	0
findinDB4_4	find information from 4th line of 4 Line data-block	294	245	1.5	0	0	0	0	0	0	0	0	4	4	2	2	0	0	0	0
findinDB4_5	find information from 5th line of 4 Line data-block	185	185	2.3	0	0	0	0	0	0	0	0	4	4	1	1	0	0	0	0
findinDB5_2	find information from 2nd line of 5 Line data-block	388	258	0.9	0	0	0	0	0	0	0	0	0	0	0	0	4	4	10	10
findinDB5_3	find information from 3rd line of 5 Line data-block	388	258	1.2	0	0	0	0	0	0	0	0	0	0	0	0	4	4	3	3
findinDB5_4	find information from 4th line of 5 Line data-block	388	258	1.3	0	0	0	0	0	0	0	0	0	0	0	0	4	4	2	2
findinDB5_5	find information from 5th line of 5 Line data-block	388	258	1.1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	1	1
type	type a string of characters into an edit box	146	146	2.5	35	47	34	44	35	47	34	44	35	47	34	44	35	47	34	44
press	press a specific button on the keyboard with a finger	35	35	0.3	140	188	136	176	140	188	136	176	140	188	136	176	140	188	136	176
motor	neural motor command	17	17	0.3	1081	1573	1010	1456	1041	1533	989	1435	1013	1505	978	1424	997	1489	1004	1430
angle	mental geographic angle computation between 2 points	25	25	1	5	7	4	6	5	7	4	6	5	7	4	6	5	7	4	6

Table 5-2 provides important insight into the frequency and complexity of various cognitive sub-processes such as neural motor commands, hand movements, eye movements, the issuing of aircraft commands, etc. In addition this table demonstrates how the CIA provides an estimate of complexity for each process. As outlined in Chapter 3 (section 3.4), from the complexity estimates of each individual CIA function and the number of calls of each function, the total complexity estimates of each scenario are computed. This is done by multiplying the complexity (κ_i) of each CIA function by the number (n_i) of times it is evoked in each scenario. The summation of these is then added to the complexity of the base scenario CIA (a_j). This is illustrated by Equation 3-1 (described in Chapter 3).

$$\kappa_{human_j} = a_j + \sum_{i=1}^{m} n_i \kappa_i \qquad \text{(E.3-1)}$$

These results are tabulated in Table 5-3. The total complexity estimates are shown in the two rightmost columns of the table.

Table 5-3. CIA complexity estimates for each of the sixteen experimental scenarios being compared.

Scenario Code	Core CIA complexity (a)		Complexity of CIA functions evoked ($\Sigma n\kappa$)		Total CIA Complexity (κ_{human})	
	AIC (bytes)	MDL (bytes)	AIC (bytes)	MDL (bytes)	AIC (bytes)	MDL (bytes)
111	2300	499	125712	120058	128012	120557
112	2620	556	184166	170858	186786	171414
121	2300	498	117758	112465	120058	112963
122	2570	538	171008	158417	173578	158955
211	2300	499	122036	116434	124336	116933
212	2620	556	180490	167234	183110	167790
221	2300	498	116635	111229	118935	111727
222	2570	538	169885	157181	172455	157719
311	2300	499	119644	114194	121944	114693
312	2620	556	178838	164994	181458	165550
321	2300	498	116966	110825	119266	111323
322	2570	538	170216	156777	172786	157315
411	2300	499	120880	113198	123180	113697
412	2620	556	179334	163998	181954	164554
421	2300	498	121048	113562	123348	114060
422	2570	538	171638	156854	174208	157392

It also appears from Table 5-3 that there is a very high correlation between AIC and MDL. The statistical value for this correlation is computed to 0.999 (p = .000). Therefore across all factors the AIC measure is essentially equivalent to MDL. This very high correlation is due the fact that most CIA functions are not compressible and are called up many times, accounting for most of the measured information. It is also important to note that the a_j term can essentially be neglected in Equation 3-1.

The overall CIA complexity estimates for each scenario are plotted in Figure 5-2. From this figure it is evident that the number of aircraft is the primary driver of complexity in the human machine interaction process (all scenarios ending with the digit 2, are scenarios with 11-12 aircraft). The number of aircraft has been shown to be the dominating complexity driver in air traffic control tasks on numerous occasions (e.g.: Hilburn, 2004; Kopardekar, 2003; Majumdar & Ochieng, 2002). Therefore the validity of the proposed complexity analysis methodology is illustrated for this interface, task, and associated CIA codes.

The second observation is that the scenarios with the equal question distribution are generally more complex than those with a spread question distribution (e.g.: compare scenarios 112 and 122). This is an expected result for those data-blocks with embedded information, as this condition imposed a higher degree of interaction with the data-blocks. This observation is also consistent with expectations and the complexity analysis methodology is able to capture it.

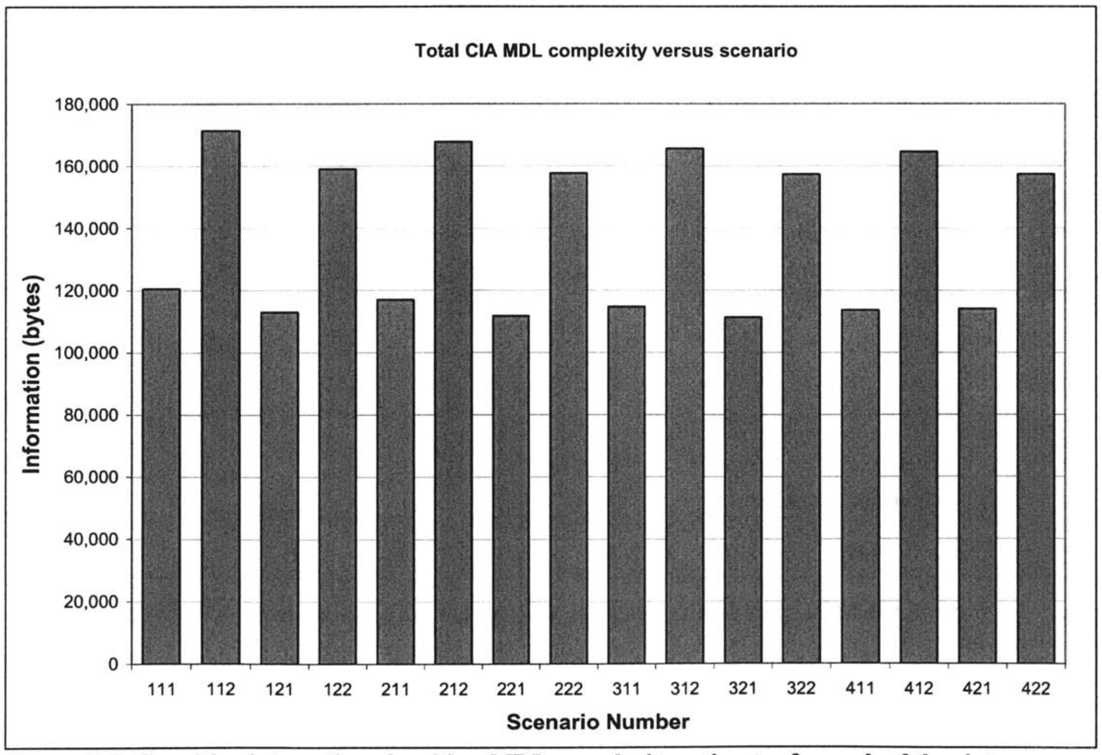


Figure 5-2. Cognitive interaction algorithm MDL complexity estimates for each of the sixteen experimental scenarios.

Relative to the strong effect of the number of aircraft, no pattern is apparent in distinguishing between the relative complexities of the data-block factor levels. The complexity analysis method predicts that the data-block type does not significantly affect complexity and hence should not significantly affect performance within the context of the representative ATC task.

Figure 5-3 breaks down the complexity for the question distribution factor and the data-block factor more clearly. In these figures the complexity measure has been normalized with respect to the largest MDL value. Figure 5-3A shows the normalized complexity due to the experimental factor of question distribution. The result shows that the equal question distribution results in higher complexity and therefore should result in lower performance.

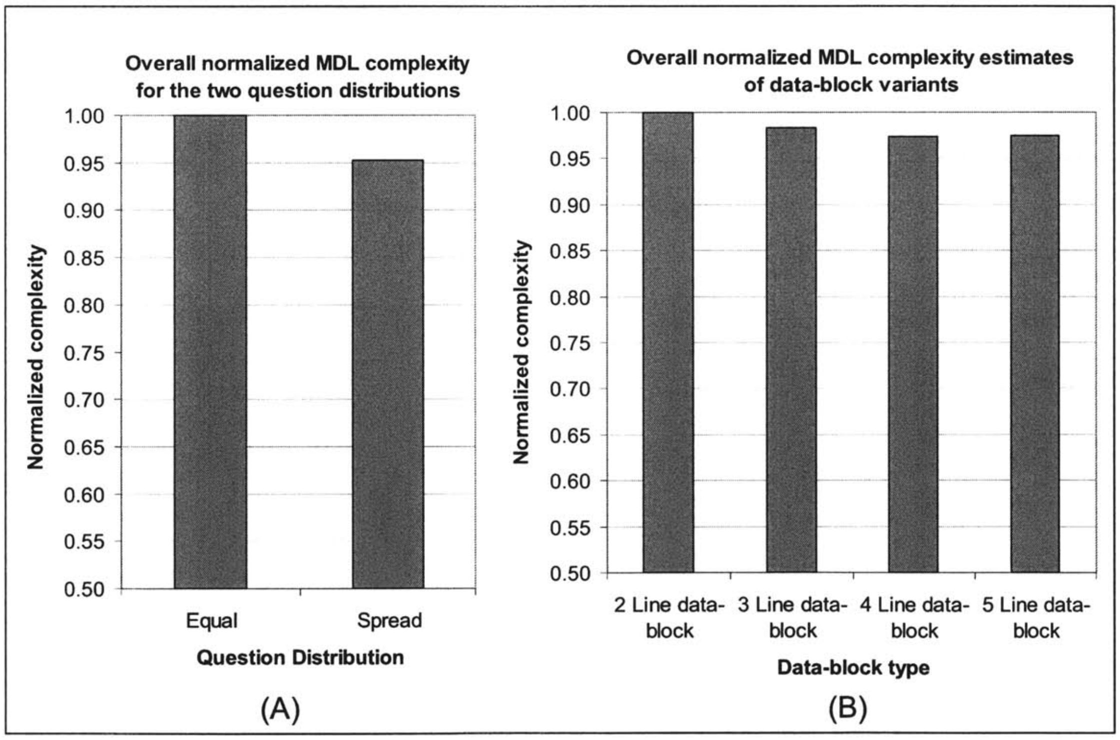


Figure 5-3. Normalized MDL complexity estimate for (A) each question distribution and (B) each data-block type.

The complexities of the competing data-block elements are seen more clearly in Figure 5-3B (i.e.: isolated from Figure 5-2). This plot shows that the two-line data-block has the largest MDL, followed by the three-line data-block and then the three and four-line data-blocks. Note that difference between the most and least complex data-block type is only about 0.025. This graph therefore suggests that based on this complexity analysis the difference between the four display design configurations is likely not significant. If as a formality one had to select a data-block type based only on these results, the four-line data-block would be chosen, as its MDL complexity is a fraction (0.002%) lower than the five-line data-block.

5.3. Comparison of Complexity and Experimental Results

In this section the MDL complexity results are compared to the experimental performance results. The section is divided into two parts. The first compares the complexity estimates to the ATC vectoring task performance results for each of the independent variable categories. The second portion compares the complexity estimates to the workload measures (question response accuracy and question response time). The results are normalized with respect to the maximum measured quantity of the set being compared.

5.3.1. Complexity and performance

The ATC primary task performance measure consisted of the egress fraction. By comparing the complexity estimates to these performance results, it is determined whether the MDL complexity of the CIA can predict performance in a representative ATC task. The relationship between complexity and performance is expected to be inversely proportional, since increasing complexity should cause decreasing performance. Figure 5-4 shows the plot of the normalized MDL graphed with the normalized egress fraction. Although it appears in part A to show that the egress fraction has a large drop between the three-line and four-line data-block, the data-block factor is not statistically significant, so essentially all four levels do not differ from one another. Thus, it is argued that the complexity prediction is satisfactory, as it also shows no significant difference across the data-block factor.

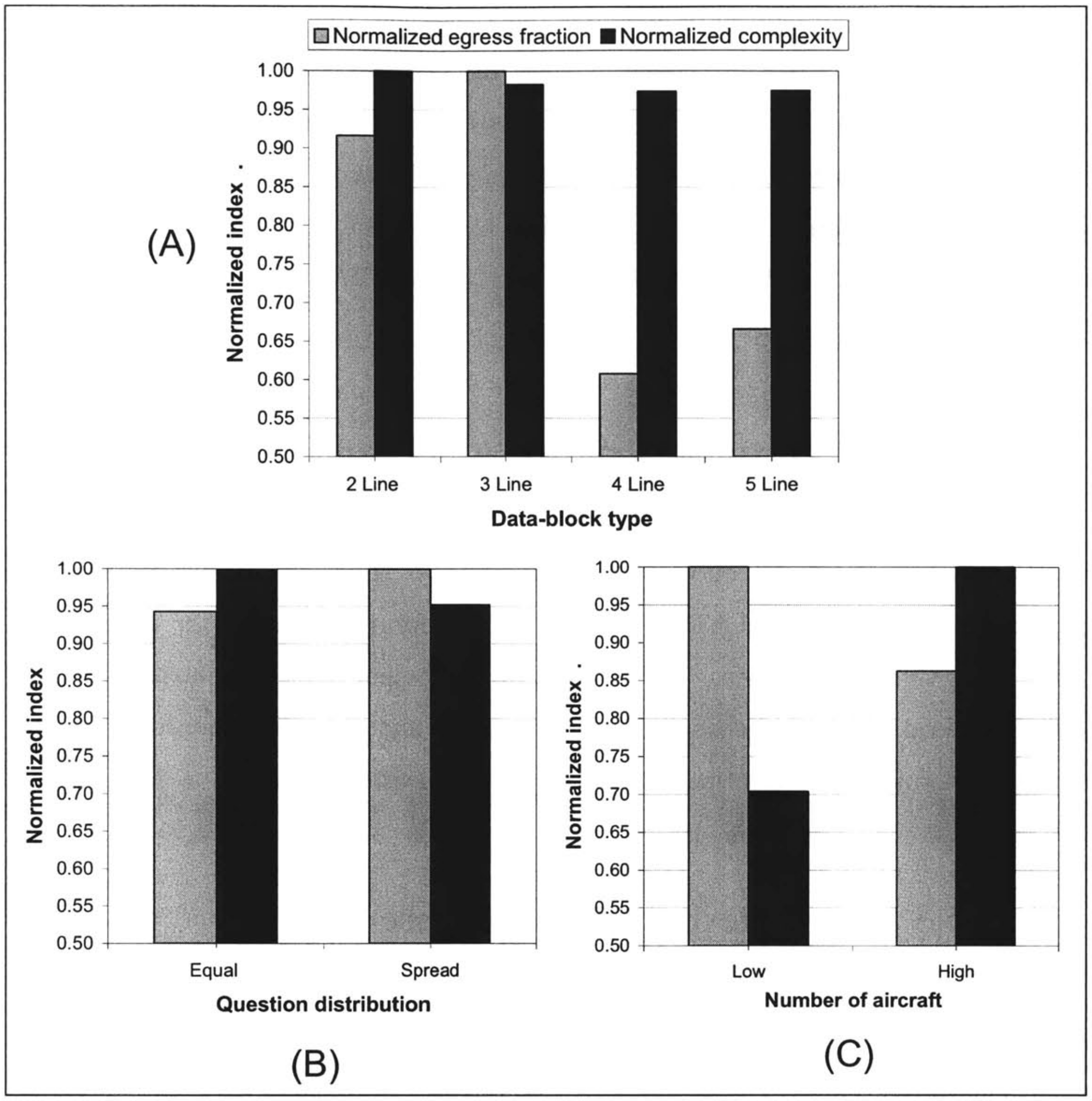


Figure 5-4. Normalized complexity and normalized egress fraction versus (A) data-block type, (B) question distribution and (C) number of aircraft.

The complexity values are also compared for the performance results across the other two factors of question distribution and number of aircraft. The graph comparing complexity and performance results for the different question distributions is shown in Figure 5-4B. The plot demonstrates the inverse relationship between complexity and performance. When estimated complexity is higher (i.e. in the equal distribution) the performance is measured to be lower and vice-versa.

The graph comparing complexity and performance results for different number of aircraft is shown in Figure 5-4C. This plot also clearly shows the inverse relationship between complexity and performance. The larger complexity estimate corresponds with lower experimentally measured performance.

In conclusion the MDL complexity measure generally follows all the statistically significant trends measured by the experimentally derived overall performance metric. The relationship between predicted complexity and measured performance is inversely proportional.

5.3.2. Complexity and workload

This section compares the complexity estimates to the normalized results of the secondary question response task. Secondary task measurements have been established in human factors as consistent estimates of cognitive workload (Vidulich, 2003). From Figure 5-5A it is observed that the performance result has a slight downward trend and then increases once again. These points are found not to be statistically different from each other in Chapter 4. This is again consistent with the complexity prediction.

As seen in Figure 5-5B, the complexity predicts a difference across question distribution, however experimental results show that the two levels do not differ. In this specific case the complexity prediction does not succeed. This is an example of a case where the overall complexity of a scenario may not always predict performance on a specific subtask, as operators can trade-off resources between subtasks to overcome the higher complexity. In Figure 5-5C the inverse relationship between complexity and response accuracy for the number of aircraft is also reflected. Therefore in terms of secondary task question response accuracy, the complexity measure successfully captures the effect of aircraft count and the non-effect data-block type.

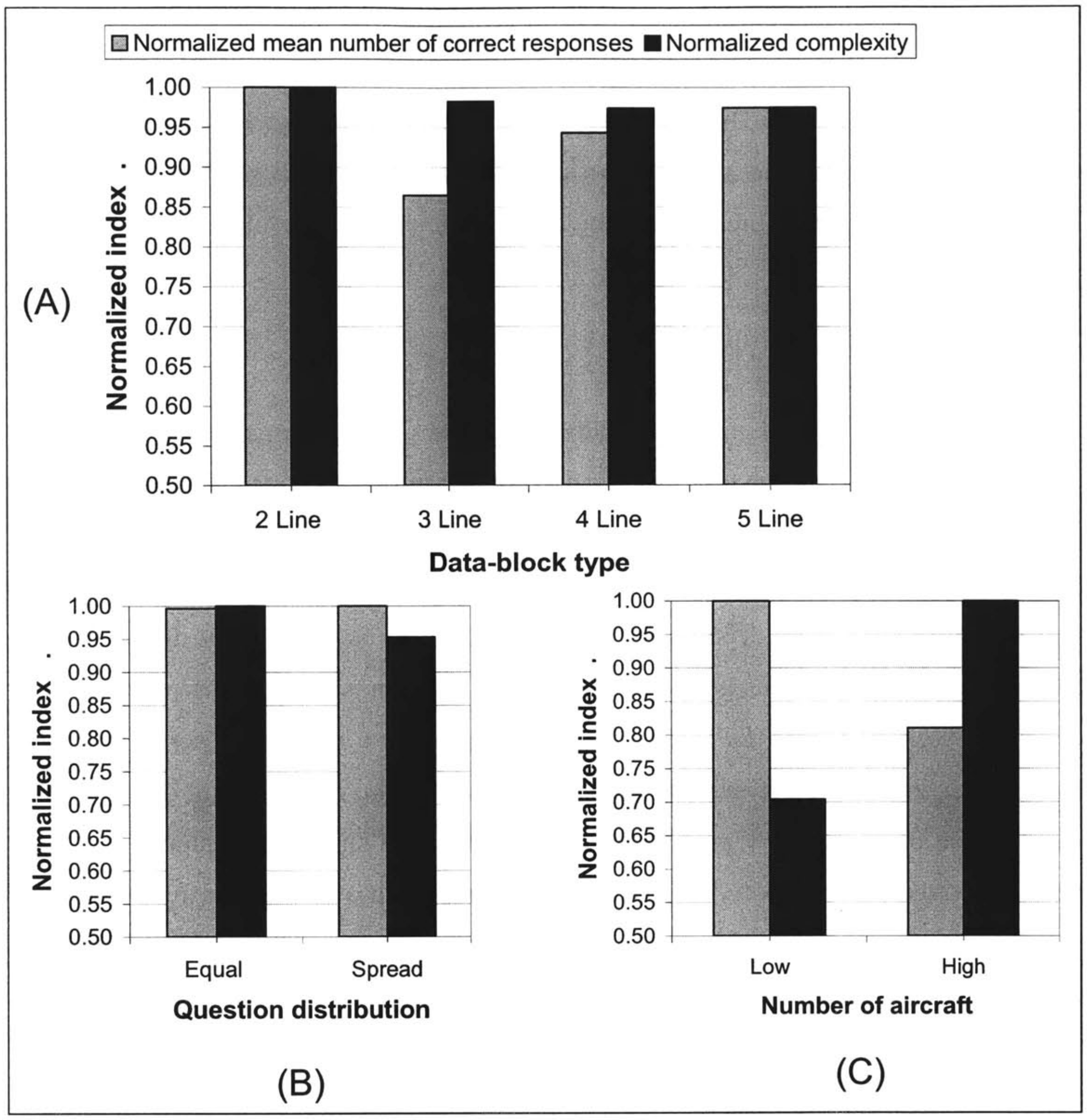


Figure 5-5. Normalized complexity and normalized correct responses versus (A) data-block type, (B) question distribution and (C) number of aircraft.

Question response time is an additional workload measure. The higher the measured secondary task response time is, the higher the level of cognitive workload. Therefore a direct relationship is expected between complexity and secondary task response time. Although not statistically significant, the question response times interestingly exhibit a similar trend to the complexity estimate as seen in Figure 5-6A. This is evidence suggesting that the complexity measure can be predictive of operator workload in human-machine interaction.

Figure 5-6B captures the direct relationship for the question distribution factor. The higher complexity predicts greater workload in the case of the equal distribution. Figure 5-6C shows the direct relationship between estimated complexity and performance for the number of aircraft. Across all the experimental factors, the complexity estimate is able to predict secondary task response time (i.e. workload).

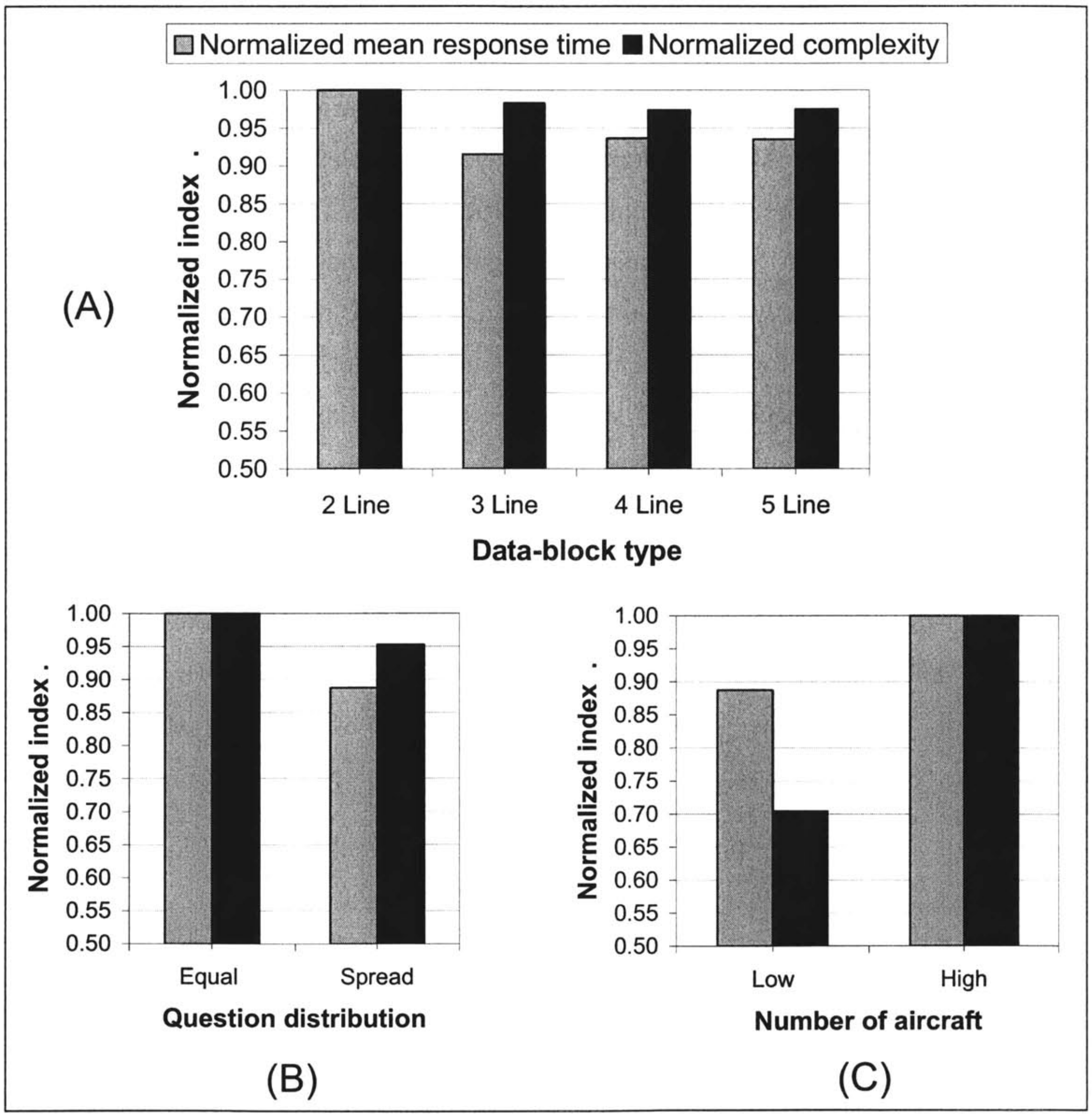


Figure 5-6. Normalized complexity and normalized question response time versus (A) data-block type, (B) question distribution and (C) number of aircraft.

5.4. Conclusions

In conclusion the complexity estimation from the MDL of the CIA consistently captures the relative experimental performance and workload differences between different human machine interaction configurations. The complexity analysis shows that the data-block type has a minimal contribution to complexity when considered within the broader context of the task. This is consistent with experimental findings. The relationship between complexity and question accuracy is also consistent for the number of aircraft factor. The results highlight that in ATC, the contribution of the complexity imparted by the number of aircraft dominates over the complexity contribution of other factors. The relationship between complexity and performance (egress fraction) is shown to be inversely proportional for all statistically significant trends. This evidence suggests that the complexity analysis methodology can be used to predict primary task performance. However for the question distribution factor, complexity predicts a difference when experimental results show none. The question response time trend is predicted by the complexity measure across all three experimental factors. The above evidence suggests that the proposed complexity methodology can predict relative workloads for different interface and task configurations. The methodology is thus validated for the representative ATC task, interfaces and CIAs. For a more general and conclusive validation, future research is suggested. The following chapter reviews the methodology, discusses its limitations, proposes future work, and includes final conclusions.

6. Conclusions

Complexity is an acute problem in air traffic control (ATC) and can ultimately limit the safety, capacity and efficiency of the system. The majority of research on ATC complexity has been concerned with examining the complexity imparted by the air traffic itself, and not on the overall complexity contributed by the human machine interaction (HMI) process.

In terms of HMI, inadequate interface and task designs may require a greater amount of operator training, likely have increased life-cycle costs, result in lower operator efficiencies and could cause a reduction in system safety. All these related issues are critical in air traffic control (ATC) and must be mitigated in order to be able to maximize traffic capacity. Methodologies that aid the development of improved technological interfaces are therefore of vast relevance, especially with the increased pervasiveness of computers in all facets of human life.

The problem of a human performing a highly procedural task using a set of tools is conceptually equivalent to the problem of the processing of a program or algorithm by an artificial computational device. HMI is primarily constrained due to information processing limits within the central nervous system (CNS) which correspond to complexity limits. The questions of how to quantify information complexity of HMI and human cognition in general have been the subject of much research and debate, yet few substantial breakthroughs have resulted.

In this thesis, the complexity of human machine interaction is defined as the *minimum amount of information required to describe the human machine interaction process in some fixed description language and chosen level of detail.* Quantifying complexity for competing interface variants and task situations can help the engineer to more objectively select between these configurations. In this thesis, a theoretically based method is developed that consists of a cognitive task analysis[18], from which the cognitive

[18] Note that the step of conducting the CTA could be bypassed if the practitioner has sufficient knowledge about the human interaction process. In that case the CIA functions could be written directly.

interaction algorithm (CIA) is derived. The complexity of the resulting algorithm is then found by the minimum description length (MDL) of the set of relevant CIA functions.

As shown in Chapter 5, the complexity estimation from the MDL of the CIA is able to consistently capture the relative performance and workload differences between different human machine interaction configurations (experimental scenarios). The relationship between complexity and primary task performance (egress fraction) is shown to be inversely proportional for all statistically significant trends. The relationship between complexity and question accuracy is also consistent for the number of aircraft factor, while the question response time trend is predicted by the complexity measure across all three experimental factors. The complexity analysis predicted a very small difference in complexity between data-block types (2.5%), thus matching the experimental results which show that the data-block type is not significant across any dependent measure. The above evidence strongly suggests that the proposed complexity analysis method can be applied to predict relative operator performance and workload for different interface and task configurations.

This method could apply to the analysis of any HMI process, including aircraft cockpits, UAV ground control stations, manufacturing, websites and personal electronics such as mobile telephones. If complexity of human machine interfaces can be effectively measured, current trends in the field of human factors could be changed by allowing a greater degree of preliminary quantitative analysis rather than extensive (and more costly) post-development testing[19]. This parallels what has occurred in the aircraft industry. In the early days of aviation aircraft design was mainly a trial and error process, requiring extensive testing. Today entire aircraft can be designed computationally and testing is minimized. The same could become true of human machine interfaces in the future.

While this complexity estimation methodology seems promising, several issues and limitations are identified. The first issue concerns the potential ambiguity of algorithms. One potential problem is that it is possible that the same interaction can be expressed with dissimilar algorithms of different complexity. Creating accurate

[19] A simple cost estimation predicts that the cost of running the experiment discussed in this thesis, to uncover differences between data-block types, was approximately six times as costly as carrying out the complexity analysis.

algorithms requires knowledge of how the human brain *actually* makes decisions and processes information. If the required level of detail about human information processing is yet unknown, and this detail is needed in order to ascertain key aspects of the interface or task configurations being compared, then this method may not be easily applicable. For example, can the algorithm truly capture the effect of clutter on a display or of slightly changing the color of an object? A second potential limitation is whether the information quantified for each CIA function accurately captures the information of the actual task. A further limitation is that while the results produce a complexity value and difference between configurations, the magnitude of this difference is difficult to interpret. To overcome this, it may be possible to scale the complexity based on the effect of a known complexity factor, or to measure performance for only a single configuration scenario and then scale that performance by the complexity fractions for every other scenario.

In future research it may be of interest to investigate whether any methods already established which capture human information processing and machine interaction, can be used to essentially formulate the equivalent of the CIA, and hence be used to estimate complexity. One method meriting further consideration is Natural GOMS Language (NGOMSL) developed by Kieras (1988; 1997) and described a structured natural language used to express the user's methods and selection rules. Complexity could be estimated by using the compressed information content of NGOMSL descriptions of HMI for different configurations. Other potential ways to estimate complexity in HMI could be attained by measuring compressed information content of sets of discrete event data gathered by screen and input device capture technology.

In subsequent iterations of this research, studies could focus solely on the expression of these algorithms in a more precise and standard way, or the use of existing algorithms from either artificial intelligence or computational cognitive science. More detailed algorithms representing the cognitive processes of controllers could be developed if this method is to be applied to actual ATC interfaces and experienced controllers. These algorithms can then be used to analyze current ATC systems, and help in the development of new ATC systems by allowing the more objective comparison of competing interface and task configurations.

Different approaches for estimating complexity could also be applied. For example one very promising yet simple metric could be attained by measuring the time required for a typical user to learn to operate a certain system (within some performance margin). The interface and/or task configuration with the lowest learning time would be the least complex. It would be of interest to compare the complexity analysis method proposed in this thesis to a learning time measure of complexity.

In conclusion this thesis attempts to connect cognitive complexity and algorithmic information theory while providing a practical measure of complexity in HMI. The thesis contributes to the literature by developing and applying the idea that human information processing can be expressed in a formal language (CIA) from which complexity estimates can then be computed by compression methods (MDL). Technological interfaces can then be optimized by selecting the design and task configurations that yield minimum complexity by the proposed method. As shown in this thesis, configurations with lower MDL complexity estimates resulted in higher performance and lower workload. This work can provide a reference for future research on measuring HMI complexity, as well as for research that attempts to express and analyze human information processing in a more formal manner.

7. References

The economic and social benefits of air transport. (2005). Geneva, Switzerland: Air Traffic Action Group.

Annett, J. (2004). Hierarchical task analysis. In D. Diaper & N. Stanton (Eds.), *The handbook of task analysis for human-computer interaction* (pp. 67-82). Mahwah, NJ: Lawrence Erlbaum Associates.

Annett, J., & Stanton, N. (2000). *Task analysis.* London ; New York: Taylor & Francis.

Arbib, M. A. (1992). Schema Theory. In S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence* (2nd ed., pp. 1427-1447): Wiley.

Arbib, M. A. (2003). *The handbook of brain theory and neural networks* (2nd ed.). Cambridge, Mass.: MIT Press.

Baddeley, A. D. (1986). *Working memory.* Oxford, UK Oxford University Press.

Bar-Yam, Y. (1997). *Dynamics of complex systems.* Reading, Mass.: Addison-Wesley.

Bar-Yam, Y. (1999). *Unifying themes in complex systems : proceedings from the International Conference on Complex Systems.* Cambridge, Mass.: Perseus Books.

Black, P. E. (Ed.). (1999). *Komogorov complexity*: CRC Press LLC.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). Occam's Razor. *Information Processing Letters, 24*(6), 377-380.

Braver, T. S., Cohen, J. D., Nystrom, L. E., Jonides, J., Smith, E. E., & Noll, D. C. (1997). A parametric study of prefrontal cortex involvement in human working memory. *Neuroimage, 5*, 49-62.

Bureau of Labor Statistics, U. S. D. o. L. (2006). *Occupational Outlook Handbook, 2006-07 Edition, Air Traffic Controllers.* Retrieved April 28, 2006, from http://www.bls.gov/oco/ocos108.htm

Burkhardt, R. (1967). *The Federal Aviation Administration.* New York: Frederik A. Praeger.

Card, D. N., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Cover, T. M., & Thomas, J. A. (1991). *Elements of Information Theory*: John Wiley & Sons, Inc.

Cowan, N. (2001). The magical number 4 in short-term memory: a reconsideration of mental storage capacity. *Behavioral and Brain Sciences, 24*(1), 87-185.

Craik, K. (1943). *The Nature of Explanation*. Cambridge, UK: Cambridge University Press.

Cummings, M. L., & Tsonis, C. G. (2005). Partitioning Complexity in Air Traffic Management Tasks. *International Journal of Aviation Psychology,* Accepted for publication.

Diaper, D., & Stanton, N. (2004). *The handbook of task analysis for human-computer interaction*. Mahwah, NJ: Lawrence Erlbaum.

Donderi, D. (2003). *A complexity measure for electronic displays: Final report on the experiments* (No. CR2003-046). Toronto, Ontario, Canada: Department of National Defence, Defense Research and Development Canada.

Donderi, D. (2006). Visual Complexity: A Review. *Psychological Bulletin, 132*(1), 73-97.

Edmonds, B. (1996). What is Complexity? In F. Heylighen & D. Aerts (Eds.), *The growth of structural and functional complexity during evolution*. Dortrecht: Kluwer.

Edmonds, B. (1999). *Syntactic Measures of Complexity*. Unpublished Doctoral Thesis, University of Manchester, Manchester, UK.

Edmonds, B. (1999). *What is Complexity? - The philosophy of complexity per se with application to some examples in evolution*. Dortrecht: Kluwer.

Fitts, P. M. (1954). The Information Capacity of the Human Motor System in Controlling the Amplitude. *Journal of Experimental Psychology, 47*(6), 381-391.

Gell-Mann, M., & Lloyd, S. (1996). Information measures, effective complexity, and total information. *Complexity, 2*(1), 44-52.

Godfrey-Smith, P. (1996). *Complexity and the function of mind in nature*. Cambridge ; New York: Cambridge University Press.

Grassberger, P. (1989). Problems in quantifying self-organized complexity. *Helvetica Physica Acta, 62*, 498-511.

Grünwald, P. D., Myung, I. J., & Pitt, M. A. (2005). *Advances in minimum description length : theory and applications*. Cambridge, Mass.: MIT Press.

Hackos, J. T. (1998). *User and task analysis for interface design*. New York: Wiley.

Heppenheimer, T. A. (1995). *Turbulent Skies*. New York: John Wiley & Sons, Inc.

Hilburn, B. (2004). *Cognitive Complexity in Air Traffic Control - A Literature Review* (No. EEC Note No. 04/04): Eurocontrol.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions of Computer-Human Interaction, 3*, 320-351.

John, B. E., & Kieras, D. E. (1996). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions of Computer-Human Interaction, 3*, 287-319.

Just, M., & Carpenter, P. (1992). A capacity theory of comprehension: individual differences in working memory. *Psychological Review, 103*(4), 761-772.

Kandel, E., R., Schwartz, J., H., & Jessel, T., M. (2000). *Principles of Neuroscience* (Fourth ed.): McGraw-Hill.

Kaye, S. M., & Martin, R. M. (2001). *On Ockham*. Belmont, CA, USA: Wadsworth/Thomson Learning.

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 135-158). Amsterdam: North-Holland.

Kieras, D. E. (1997). A Guide to GOMS model usability evaluation using NGOMSL. In M. Helander (Ed.), *Handbook of human-computer interaction* (2nd ed., pp. 733-766). Amsterdam: North-Holland.

Kieras, D. E. (2004). GOMS models for task analysis. In D. Diaper & N. Stanton (Eds.), *The handbook of task analysis for human-computer interaction*. Mahwah, NJ: Lawrence Erlbaum Associates.

Kopardekar, P. (2003). *Measurement and Prediction of Dynamic Density*. Paper presented at the ATM 2003 R&D Seminar, Budapest.

Laudeman, I. V., Shelden, S. G., Branstrom, R., & Brasil, C. L. (1998). *Dynamic density: An air traffic management metric* (No. NASA-TM).

Li, M., & Vitányi, P. M. B. (1997). *An introduction to Kolmogorov complexity and its applications* (2nd ed.). New York: Springer.

Mack, A., & Rock, I. (1998). *Inattentional blindness*. Cambridge, MA: MIT Press.

Magrab, E. B. (2000). *An engineer's guide to MATLAB*. Upper Saddle River, NJ: Prentice Hall.

Majumdar, A., & Ochieng, W. Y. (2002). *The factors affecting air traffic controller workload: a multivariate analysis based upon simulation modeling of controller workload.* Paper presented at the 81st Annual Meeting of the Transportation Research Board, Washington DC.

Masalonis, A. J., Callaham, M. B., & Wanke, C. R. (2003). *Dynamic Density and Complexity Metrics for Realtime Traffic Flow Management.* Paper presented at the 5th USA/Europe Air Traffic Management R&D Seminar, Budapest, Hungary.

Merriam-Webster, I. (2001). *Merriam-Webster's collegiate dictionary* (10th ed.). Springfield, Mass.: Merriam-Webster.

Militello, L., & Hutton, R. (2000). Applied cognitive task analysis (ACTA): a practitioner's toolkit for understanding cognitive task demands. In J. Annett & N. Stanton (Eds.), *Task Analysis* (pp. 90-113). London ; New York: Taylor & Francis.

Miller, C. (2000). The Human Factor in Complexity. In T. Samad & J. Weyrauch (Eds.), *Automation, Control and Complexity: An Integrated Approach.* Chichester: John Wiley & Sons Ltd.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review, 63*, 81-97.

Mogford, R. H. (1997). Mental models and situation awareness in air traffic control. *International Journal of Aviation Psychology, 7*(4), 331-341.

Nolan, M. S. (2004). *Fundamentals of air traffic control* (4th ed.). Belmont, CA: Thomson-Brooks/Cole.

Norman, D., & Bobrow, D. (1975). On data-limited and resource limited processing. *Cognitive psychology, 7*, 44-60.

Pinker, S. (1997). *How the mind works* (1st ed.). New York: Norton.

Rasmussen, J. (1986). *Information processing and human-machine interaction : an approach to cognitive engineering*. New York: North-Holland.

Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica, 14*, 465-471.

Schraagen, J. M., Chipman, S. F., & Shalin, V. L. (2000). *Cognitive Task Analysis.* Mahwah, NJ: Lawrence Erlbaum Associates

Seamster, T. L., Redding, R. E., Cannon, J. R., Ryder, J. M., & Purcell, J. A. (1993). Cognitive task analysis of expertise in air traffic control. *International Journal of Aviation Psychology, 3*, 257-283.

Seamster, T. L., Redding, R. E., & Kaempf, G. L. (1997). *Applied cognitive task analysis in aviation.* Aldershot, England ; Brookfield, Vt.: Avebury Aviation.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal, 27*, 379-423 and 623-656.

Shannon, C. E., & Weaver, W. (1964). *The mathematical theory of communication.* Urbana,: University of Illinois Press.

Simons, D. J. (2000). Attentional capture and inattentional blindness. *Trends in Cognitive Sciences, 4*(4), 147-155.

Sirdhar, B., Seth, K. S., & Grabbe, S. (1998). *Airspace complexity and its application in air traffic management.* Paper presented at the 2nd USA/Europe ATM R&D Seminar, Orlando, FL.

Skolnik, M. (2001). *Introduction to RADAR systems.* New York: McGraw-Hill.

Squire, L. (1992). Memory and the hippocampus: a synthesis from findings with rats, monkeys and humans. *Psychological Review, 99*(2), 195-231.

Vidulich, M. A. (2003). Mental workload and situation awareness: essential concepts for aviation psychology practice. In P. S. Tsang & M. A. Vidulich (Eds.), *Principles and practice of aviation psychology.* Mahwah, NJ: Lawrence Erlbaum Associates.

Wang, R. F., & Spelke, E. S. (2002). Human spatial representation: insights from animals. *Trends in Cognitive Sciences, 6*(9).

Wickens, C. D., & Hollands, J. G. (2000). *Engineering psychology and human performance* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Wickens, T. D. (2002). *Elementary signal detection theory.* Oxford ; New York: Oxford University Press.

Xing, J. (2004). *Measures of Information Complexity and the Implications for Automation Design* (No. DOT/FAA/AM-04/17).

Xing, J., & Manning, C. A. (2005). *Complexity and Automation Displays of Air Traffic Control: Literature Review and Analysis* (No. DOT/FAA/AM-05/4).

Appendix A. Table of Subtasks and Information Requirements

This appendix contains information regarding the breakdown of the various subtasks that were identified as part of the cognitive task analysis (CTA). These helped with the creation of the cognitive process flow charts that were presented in Chapter 3. They also include listings of the information requirements for each subtask.

Table A1. Task decomposition and information requirements

No.	Subtask	Description	Information requirements
1	Direct aircraft	Direct aircraft that appear on screen to the required egress and ensure exits at the correct velocity and altitude for the given egress.	Flight number, actual altitude, actual velocity, actual heading, desired altitude, desired velocity, desired heading, selected aircraft, egress specifications, control panel state, cursor position, aircraft/subtask priority, overall objectives.
1.1*	Scan display (version 1)	Scan eyes about the display in order to perceive information required for comprehension of the situation. Isolate an aircraft that must be directed.	Positions of aircraft, status of aircraft, flight numbers, egress locations and information
1.1.1	Choose aircraft	Mentally isolate an aircraft to direct based on a certain priority (eg: which is closest to the egress)	Position of selected aircraft, flight number of selected aircraft
1.1.1.1	Compute distance to exit line	Capture the distance of the aircraft to exit and hold for comparison	Position of selected aircraft, position of nearest exit line
1.1.2	Perceive destination	Perceive the egress destination of a certain aircraft	Egress information (Z1 - Z4), aircraft position
1.1.2.1	Check specific exit requirements	Check the exit requirements for a given exit from the chart	
1.2	Issue Command	Issue a command to a given aircraft	Aircraft, desired aircraft state, current aircraft state

1.2.1	Heading command	Change the heading of a given aircraft	Aircraft position, desired aircraft direction, actual aircraft direction
1.2.1.1	Click on aircraft	Select aircraft to which to command will be issued by clicking on the graphical symbol	Aircraft, cursor position, click/selection feedback
1.2.1.2	Click heading radio button	Select the radio button on the command panel for heading	Knowledge of which button is for heading (either by location or by reading label), feedback on click, cursor position
1.2.1.3	Direction to heading conversion	Convert a desired heading direction (visual) to a heading angle from 0-360 degrees (numeric)	Current direction, current heading, desired heading, mental conversion/computation
1.2.1.4	Click in text box	Click in text box adjacent to radio buttons	location of text box, position of cursor, click feedback
1.2.1.5	Erase text in text box	Erase the text by continuously pressing backspace or by selecting text and hitting backspace or delete	selection of text, keyboard button (eg: backspace)
1.2.1.6	Change to keyboard	Move hand from mouse to keyboard or use other hand	Location of the required keyboard keys
1.2.1.7	Type new text in text box	Type the desired heading (3 digits) in the text box	Keyboard numeric keys, desired heading
1.2.1.8	Change to mouse	Move hand from keyboard to mouse the same hand was used to type	Location of mouse
1.2.1.9	Click command	Click on the command button at which point the aircraft begins its maneuver and the control panel disappears	cursor position, command button location, click feedback
1.2.2	Altitude command	Change the altitude of a given aircraft	Aircraft, current altitude, desired altitude
1.2.2.1	Click on aircraft	Select aircraft to which to command will be issued by clicking on the graphical symbol	Aircraft, cursor position, click/selection feedback
1.2.2.2	Click altitude radio button	Select the radio button on the command panel for altitude	Knowledge of which button is for altutude (either by location or by reading label), feedback on click, cursor position

1.2.2.3	Click in text box	Click in text box adjacent to radio buttons	location of text box, position of cursor, click feedback
1.2.2.4	Erase text in text box	Erase the text by continuously pressing backspace or by selecting text and hitting backspace or delete	selection of text, keyboard button (eg: backspace)
1.2.2.5	Check desired altitude	Check the required exit altitude from sector map (note position in sequence may vary)	Location of map, selected aircraft required egress (Z1-Z4)
1.2.2.6	Change to keyboard	Move hand from mouse to keyboard or use other hand	Location of the required keyboard keys
1.2.2.7	Type new text in text box	Type the desired altitude (3 digits) in the text box	Keyboard numeric keys, desired altitude
1.2.2.8	Change to mouse	Move hand from keyboard to mouse the same hand was used to type	Location of mouse
1.2.2.9	Click command	Click on the command button at which point the aircraft begins its maneuver and the control panel disappears	cursor position, command button location, click feedback
1.2.3	Speed command	Change the speed of a given aircraft	Aircraft, desired aircraft speed, current aircraft speed
1.2.3.1	Click on aircraft	Select aircraft to which to command will be issued by clicking on the graphical symbol	Aircraft, cursor position, click/selection feedback
1.2.3.2	Click speed radio button	Select the radio button on the command panel for speed	Knowledge of which button is for speed (either by location or by reading label), feedback on click, cursor position
1.2.3.3	Click in text box	Click in text box adjacent to radio buttons	location of text box, position of cursor, click feedback
1.2.3.4	Erase text in text box	Erase the text by continuously pressing backspace or by selecting text and hitting backspace or delete	selection of text, keyboard button (eg: backspace)
1.2.3.5	Check desired altitude	Check the required exit altitude from sector map (note position in sequence may vary)	Location of map, selected aircraft required egress (Z1-Z4)

1.2.3.6	Change to keyboard	Move hand from mouse to keyboard or use other hand	Location of the required keyboard keys
1.2.3.7	Type new text in text box	Type the desired speed (2 digits) in the text box	Keyboard numeric keys, desired speed
1.2.2.8	Change to mouse	Move hand from keyboard to mouse the same hand was used to type	Location of mouse
1.2.3.9	Click command	Click on the command button at which point the aircraft begins its maneuver and the control panel disappears	cursor position, command button location, click feedback
2	Conflict detection and correction	Ensure that no aircraft are in conflict and prevent future conflicts by projecting future aircraft positions	Current aircraft positions, future aircraft positions, predicted trajectory courses.
2.1*	Scan display (version 2)	Scan eyes about the display in order to perceive information required for conflict detection	Current aircraft positions, future aircraft positions, predicted trajectory courses.
2.1.1	Check for common flight levels	Check if aircraft are at same altitude or may be in the near future	Current aircraft positions, current aircraft altitudes, rates of climb
2.1.2	Project future positions	Mentally estimate future trajectories of aircraft and whether they will intersect at the same point in time	Current aircraft positions, current aircraft speeds, future aircraft positions
2.1.3	Conflict uncertainty level	Based on the trajectories determine a level of uncertainty regarding a potential conflict	
2.2*	Identify conflicting aircraft	Identify two or more aircraft which are predicted to conflict	
2.2.1	Display and view conflict circles	Click a button on screen to view 3n.m. conflict circles around each aircraft	Location of button, status of circles (on/off)
2.3	Modify aircraft trajectory	Modify the trajectory of an aircraft which may conflict	
2.3.1	Speed command	Change the speed of a given aircraft	Aircraft, desired aircraft speed, current aircraft speed

2.3.2	Altitude command	Change the altitude of a given aircraft	Aircraft, current altitude, desired altitude
2.3.3	Heading command	Change the heading of a given aircraft	Aircraft position, desired aircraft direction, actual aircraft direction
3	Respond to questions	Respond to the questions requested by searching for the requested piece of information, typing it in, and submitting it	
3.1	Sense aural alert	Hear aural signal denoting a new request and decode its meaning	knowledge of audio signal meaning
3.2	See new request message	Notice the new request text has turned green and decode its meaning	knowledge of new request meaning
3.3	Focus eyes and attention to data-link	Once the alert has been heard, eyes and attention are pointed at the data-link interface in order to read the question	Location of data-link interface and question, knowledge that new question has appeared
3.3.1	Read question	Read the sentence containing the question that has appeared	Language knowledge to decode meaning of symbols and words
3.4*	Scan display (version 3)	Scan eyes about the display in order to find a specific flight number amidst the aircraft	aircraft positions, aircraft already scanned, target flight number, information type required
3.5*	Extract information from data-block	Once particular flight is found on screen, one	information type required, position of that information within data-block
3.5.1	Hold information	Once the piece of information has been extracted from the data-block must hold it in short term memory until it can be dumped	Piece of information from within data-block
3.6	Submit answer	Submit the answer	Piece of information from within data-block, location of data-link interface, submission feedback
3.6.1	Click in text box	Click inside the data-link text box	Position of data-link text box, feedback of click
3.6.2	Change to keyboard	Move hand from mouse to keyboard or use other hand	Location of the required keyboard keys

3.6.3	Type response	Type piece of information about flight inside data-link text box	Piece of information from within data-block
3.6.4	Change to mouse and move cursor	Move hand from keyboard to mouse the same hand was used to type	Location of mouse, location of cursor
3.6.5	Report response	Once the information has been typed inside the text box, it must be reported by clicking on the report button	cursor position, report button location, click feedback
4	Confirm alert	Confirm that a yellow dot beside each data-block has been seen by clicking a button	Knowledge of alert meaning, attentional (conscious) capture of presence of alert, cursor position, check confirm button location
4.1*	Scan display (version 4)	Scan eyes about the display in order to perceive yellow dot representing alert.	Knowledge of alert meaning, attentional (conscious) capture of presence of alert
4.1.1	Attentional capture of alert	Conscious capture	Alert status (on/off)
4.2	Press check confirm button	Move cursor towards check confirm button	Position of check confirm button, position of cursor
4.2.1	Confirm alert disappears	After pressing the button, ensure that the alert has disappeared from the data-block	Alert status (on/off)

Appendix B: Cognitive Interaction Algorithm Functions and Charts

This appendix contains the remaining cognitive interaction algorithm functions and charts that are not included in Chapter 3. Functions described in Chapter 3 are not described again and only the algorithms are shown.

selectAC

The first function presented in this appendix is the *selectAC* function which chooses and aircraft based on a certain priority. The cognitive task analysis showed that priority is based simply on the distance to the exit location. The CIA function for selecting an aircraft is shown below. This function returns the next aircraft that is to be commanded.

```
function prox = selectAC
prox = 1
for n = 1:nac
if ac(prox).pos >= ac(n).pos
for i = 1:length(complete)
if n ~= complete(i)
prox = n
end
end
end
end
```

The operator selects an aircraft based on proximity to the exit, but also one that has not been commanded previously. This algorithm assumes that the operators have complete recollection of the aircraft which have already been commanded. In reality this is not the case and is one of the reasons why controllers annotate and update flight strips as they go about their ATC task. It is also assumed that the scan pattern for this function begins by fixating upon the exit and then moving the eyes to the nearest aircraft, such that the effect of the number of aircraft is minimal.

command_alt, command_spd

The functions for commanding altitude and speed are similar to the function for commanding heading that was presented in Chapter 3. The charts for these two functions are shown in Figures B1 and B2 and show what other functions are called and how many times. The algorithms follow the charts

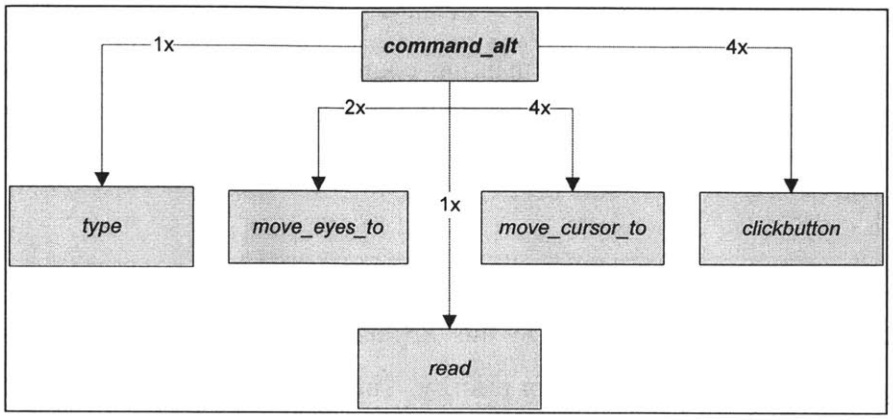


Figure B1. CIA functions called by altitude command

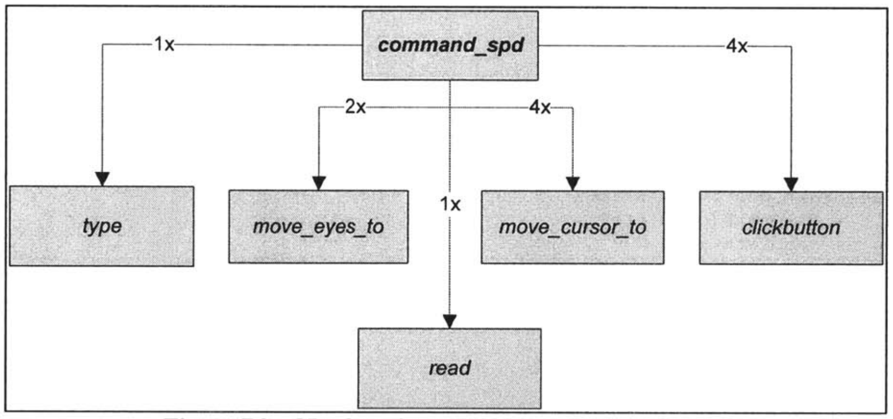


Figure B2. CIA functions called by speed command

```
function command_alt(p)
move_eyes_to(cursor.xy)
move_cursor_to(ac(p).symbol.xy)
clickbutton
ac(p).exit = read(ac(p).exit)
move_eyes_to(chart.xy)
if ac(p).exit == Z1
altrequired = 280;
elseif ac(p).exit == Z2
altrequired = 290;
elseif ac(p).exit == Z3
altrequired = 350;
elseif ac(p).exit == Z4
altrequired = 260;
end
move_cursor_to(ctrpanel.radio2.xy)
clickbutton
move_cursor_to(ctreditbox.xy)
```

```
clickbutton
type(altrequired)
move_cursor_to(cmndbutton.xy)
clickbutton
```

```
function command_spd(p)
move_eyes_to(cursor.xy)
move_cursor_to(ac(p).symbol.xy)
clickbutton
ac(p).exit = read(ac(p).exit)
move_eyes_to(chart.xy)
if ac(p).exit == Z1
spdrequired = 42;
elseif ac(p).exit == Z2
spdrequired = 49;
elseif ac(p).exit == Z3
spdrequired = 48;
elseif ac(p).exit == Z4
spdrequired = 41;
end
move_cursor_to(ctrpanel.radio3.xy)
clickbutton
move_cursor_to(ctreditbox.xy)
clickbutton
type(spdrequired)
move_cursor_to(cmndbutton.xy)
clickbutton
```

move_hand_to, move_eyes_to, move_cursor_to

```
function move_hand_to(xyz)
delta = hand.xyz - xyz
motor(delta)
```

```
function move_eyes_to(xy)
delta = fix.xy - xy
motor(delta)
```

```
function move_cursor_to(xy)
move_hand_to(K*xy)
move_eyes_to(xy)
```

type, clickbutton, press

The chart showing the calls by the *type* function is shown in Figure B3 followed by the simple algorithm.

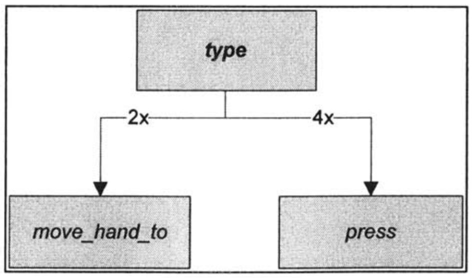


Figure B3. CIA functions called by the type function

```
function type(txt)
move_hand_to(keyboard.xyz)
if edit ~= []
press(bkspace)
end
for i=length(txt)
press(txt(i))
end
move_hand_to(mouse.xyz)
```

```
function clickbutton
motor(finger_down)
```

```
function press(key)
motor(key.xyz)
```

find_flight_high

```
function targetpos = find_flight_high(fl_no)
temp = 0;
while temp ~=target
j=1
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=2
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=3
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=4
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=5
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=6
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=7
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=8
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=9
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=10
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
j=11
move_eyes_to(ac(j).xy)
temp = read(ac(j).flno)
end
targetpos = ac(j).xy
```

findinDB3_X

The functions below describe the algorithmic representation of the extraction processes for each line (*X*) of the three line data-block

```
function item = findinDB3_2(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
else
item = read(ac(j).type)
end
```

```
function item = findinDB3_3(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
else
item = read(ac(j).type)
end
```

```
function item = findinDB3_4(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == CID
item = read(ac(j).CID)
else
item = read(ac(j).origin)
end
```

```
function item = findinDB3_5(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
```

findinDB4_X

The functions below describe the algorithmic representation of the extraction processes for each line (*X*) of the four line data-block

```
function item = findinDB4_2(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
```

```
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
else
item = read(ac(j).dest)
end
```

```
function item = findinDB4_3(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
else
item = read(ac(j).dest)
end
```

```
function item = findinDB4_4(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
else
item = read(ac(j).dest)
end
```

```
function item = findinDB4_5(j,info)
move_eyes_to(cursor.xy)
move_cursor_to(ac(j).downarrows.xy)
clickbutton
if info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
```

findinDB5_X

The functions below describe the algorithmic representation of the extraction processes for each line (*X*) of the five line data-block

```
function item = findinDB5_2(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
```

```
elseif info == dest
item = read(ac(j).dest)
elseif info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
end
```

```
function item = findinDB5_3(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
elseif info == dest
item = read(ac(j).dest)
elseif info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
end
```

```
function item = findinDB5_4(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
elseif info == dest
item = read(ac(j).dest)
elseif info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
end
```

```
function item = findinDB5_5(j,info)
if info == alt
item = read(ac(j).alt)
elseif info == spd
item = read(ac(j).spd)
elseif info == dest
item = read(ac(j).dest)
elseif info == actype
item = read(ac(j).type)
elseif info == CID
item = read(ac(j).CID)
elseif info == dest
item = read(ac(j).dest)
elseif info == pax
item = read(ac(j).pax)
else
item = read(ac(j).bag)
end
```

```
end
```

scanpath_low, scanpath_high

The two "scanpath" functions describe the process of scanning between targets on the display and their charts are shown in Figure B4. Both algorithms follow after the chart.

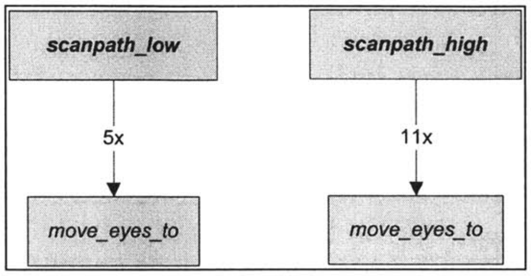


Figure B4. CIA functions called by the scan path functions

```
function scanpath_low
move_eyes_to(ac(1).xy)
move_eyes_to(ac(2).xy)
move_eyes_to(ac(3).xy)
move_eyes_to(ac(4).xy)
move_eyes_to(ac(5).xy)
```

```
function scanpath_high
move_eyes_to(ac(1).xy)
move_eyes_to(ac(2).xy)
move_eyes_to(ac(3).xy)
move_eyes_to(ac(4).xy)
move_eyes_to(ac(5).xy)
move_eyes_to(ac(6).xy)
move_eyes_to(ac(7).xy)
move_eyes_to(ac(8).xy)
move_eyes_to(ac(9).xy)
move_eyes_to(ac(10).xy)
move_eyes_to(ac(11).xy)
```

The above CIA functions simply consist of a set of eye movements between the various aircraft. It is assumed that a complete scan of all aircraft in the airspace occurs. This is usually not the case but balances out as most scans are partial even though they occur more frequently.

seedot

It is assumed that during each one of the previously described scans is when an alert is detected beside an aircraft data-block. This is described by the *seedot* function whose CIA is shown below.

```
function d = seedot
if dot == 'on'
d=1
end
```

report

The *report* function is the description of clicking the *Report* button on the display. The chart showing the functions called by report is shown in Figure B5 and is followed by the algorithm

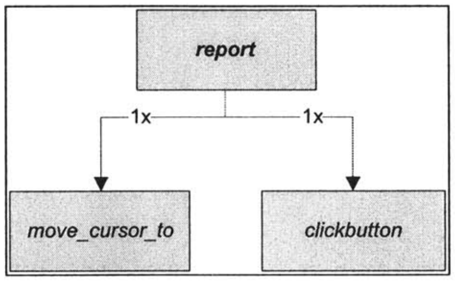


Figure B5. CIA functions called by the report function

```
function report
move_cursor_to(report.xy)
clickbutton
```

In the above function the cursor is moved to the known location on the display which is part of the human's spatial map of the interface. The mouse button is then clicked (*clickbutton)*.

Base CIAs for configurations of representative example in Chapter 3

The charts and algorithm below are for the base CIA of each data-block configuration for the representative task presented as an example in Section 3.4. The three remaining charts are shown followed by a single generic algorithm. The specific algorithms for each of the four variants are created by replacing the *J* in the *findinDB* function calls, with the data-block type number (2-5).

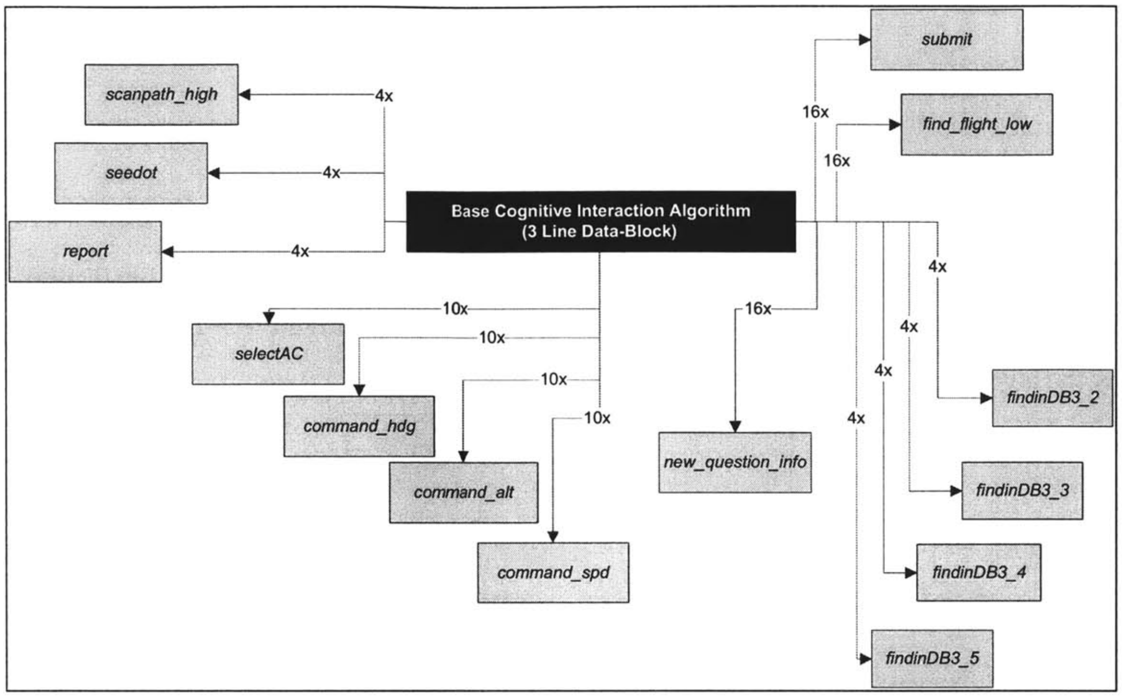


Figure B6. Structure of base CIA for the three-line data-block configuration. The numbers on the lines indicate the number of times each CIA function is called from the base algorithm.

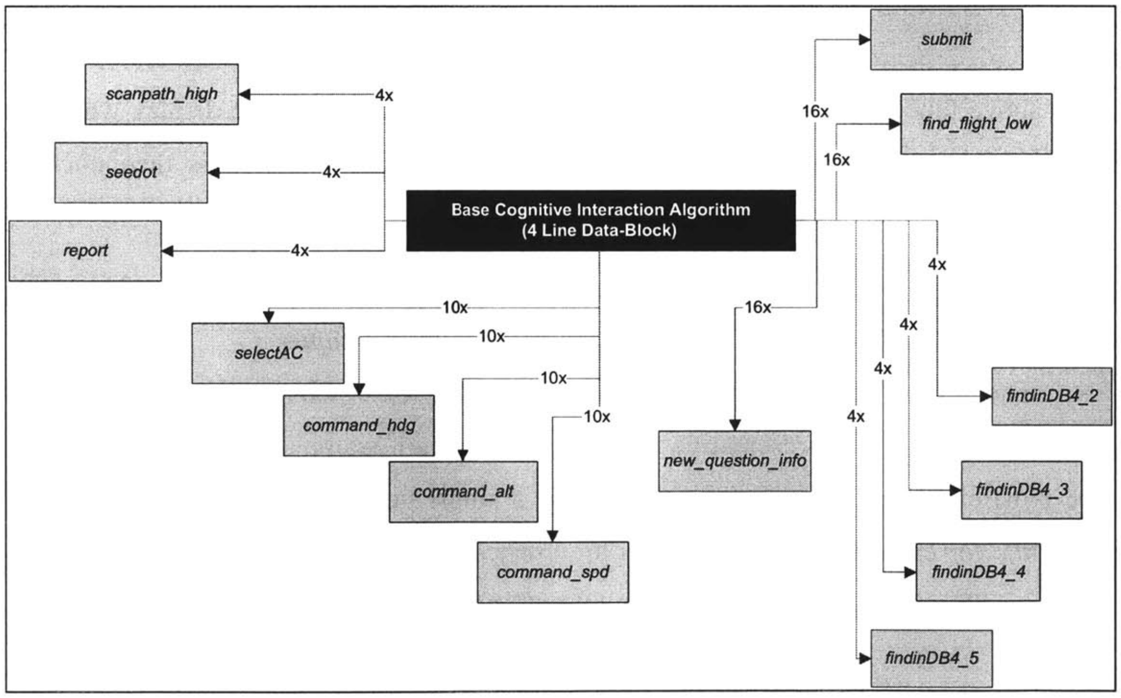


Figure B7. Structure of base CIA for the four-line data-block configuration. The numbers on the lines indicate the number of times each CIA function is called from the base algorithm.

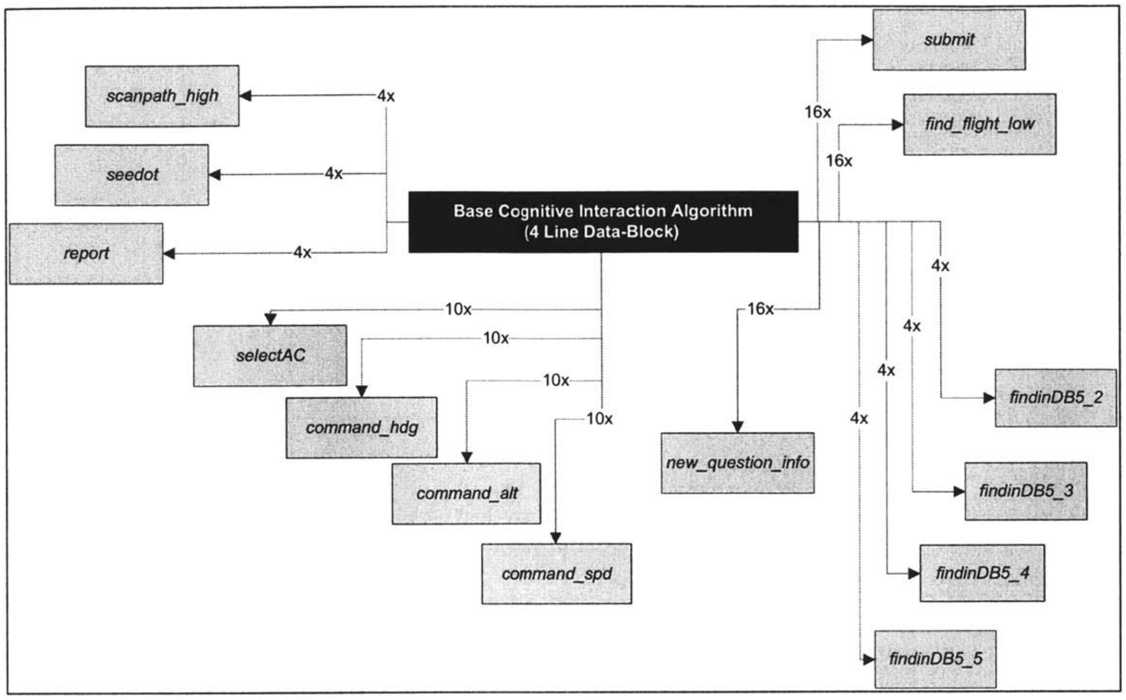


Figure B8. Structure of base CIA for the five-line data-block configuration. The numbers on the lines indicate the number of times each CIA function is called from the base algorithm.

```
ac1 = selectAC
command_hdg(a1)
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_hdg(a3)
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_hdg(a6)
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
ac8 = selectAC
command_hdg(a8)
command_alt(a8)
command_spd(a8)
ac9 = selectAC
command_hdg(a9)
```

```
command_alt(a9)
command_spd(a9)
ac10 = selectAC
command_hdg(a10)
command_alt(a10)
command_spd(a10)
q1 = new_question_info
targetAC = find_flight_high(q1(1))
ans = findinDBJ_5(targetAC,q1(2))
submit(ans)
q2 = new_question_info
targetAC = find_flight_high(q2(1))
ans = findinDBJ_3(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_high(q3(1))
ans = findinDBJ_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_high(q4(1))
ans = findinDBJ_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_high(q5(1))
ans = findinDBJ_2(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_high(q6(1))
ans = findinDBJ_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_high(q7(1))
ans = findinDBJ_4(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_high(q8(1))
ans = findinDBJ_4(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_high(q9(1))
ans = findinDBJ_5(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_high(q10(1))
ans = findinDBJ_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_high(q11(1))
ans = findinDBJ_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_high(q12(1))
ans = findinDBJ_2(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDBJ_3(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDBJ_4(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDBJ_5(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDBJ_4(targetAC,q16(2))
submit(ans)
scanpath_high
```

```
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

Page intentionally left blank

Appendix C: Base Cognitive Interaction Algorithms for Experimental Scenarios

This appendix contains the fifteen base cognitive interaction algorithms that are not included in Chapter 5, and which represent the other experimental scenarios. The remaining charts are also included here. Refer to the legend provided in Table 5-1 for a description of the scenario numbering scheme.

C.1 Scenario 111

The chart representing scenario 111 is shown in Figure C1. The algorithm follows after the chart.

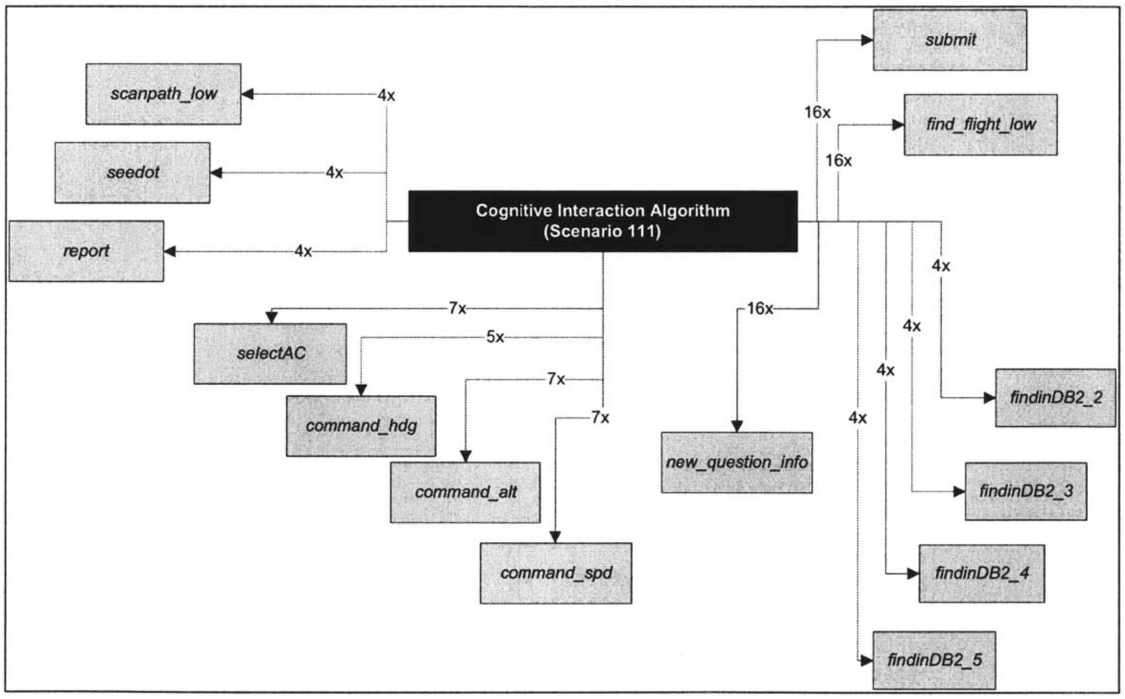


Figure C1. Scenario 111 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_hdg(a3)
command_alt(a3)
command_spd(a3)
```

```
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB2_2(targetAC,q1(2))
submit(ans)
q2 = new_question_info
targetAC = find_flight_low(q2(1))
ans = findinDB2_5(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB2_3(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB2_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB2_4(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB2_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB2_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB2_5(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB2_3(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB2_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB2_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB2_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB2_2(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB2_3(targetAC,q14(2))
submit(ans)
```

```
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB2_4(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB2_5(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
scanpath_low
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.2 Scenario 112

The algorithm (annotated) and chart for this scenario are presented in Chapter 5.

C.3 Scenario 121

The chart representing scenario 121 is shown in Figure C2. The algorithm follows after the chart.

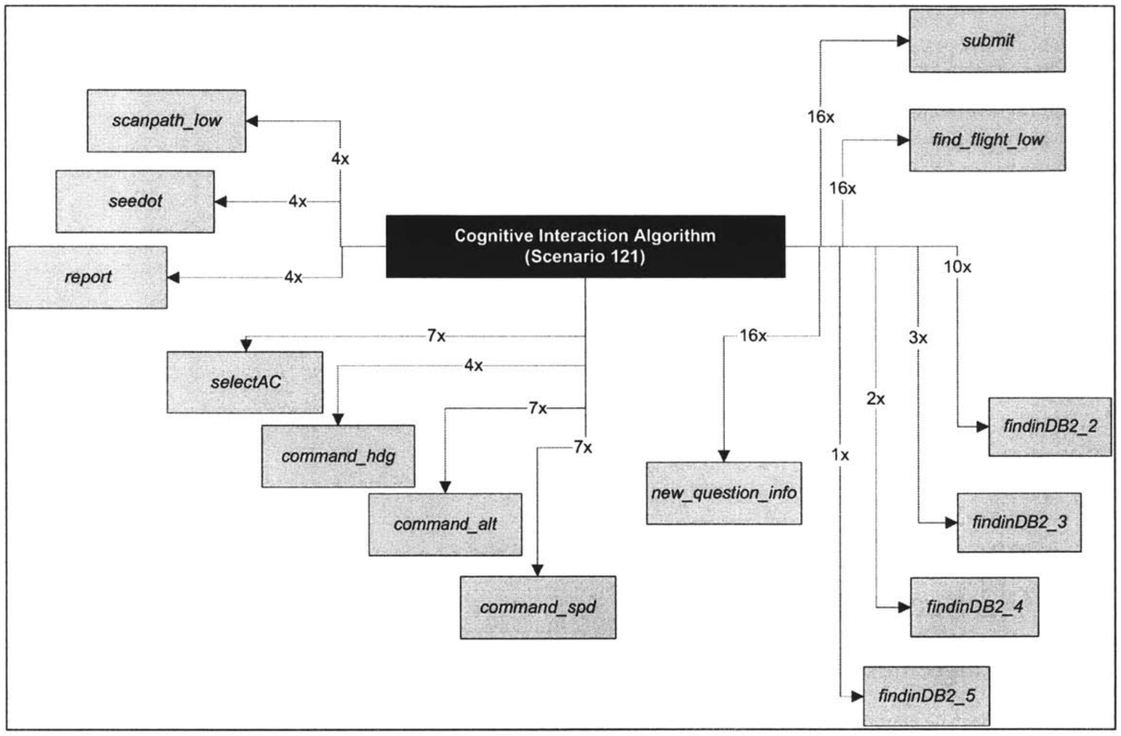


Figure C2. Scenario 121 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
```

```
        ans = findinDB2_2(targetAC,q1(2))
        submit(ans)
        q2 = new_question_info
        targetAC = find_flight_low(q2(1))
        ans = findinDB2_3(targetAC,q2(2))
        submit(ans)
        q3 = new_question_info
        targetAC = find_flight_low(q3(1))
        ans = findinDB2_2(targetAC,q3(2))
        submit(ans)
        q4 = new_question_info
        targetAC = find_flight_low(q4(1))
        ans = findinDB2_2(targetAC,q4(2))
        submit(ans)
        q5 = new_question_info
        targetAC = find_flight_low(q5(1))
        ans = findinDB2_3(targetAC,q5(2))
        submit(ans)
        q6 = new_question_info
        targetAC = find_flight_low(q6(1))
        ans = findinDB2_2(targetAC,q6(2))
        submit(ans)
        q7 = new_question_info
        targetAC = find_flight_low(q7(1))
        ans = findinDB2_2(targetAC,q7(2))
        submit(ans)
        q8 = new_question_info
        targetAC = find_flight_low(q8(1))
        ans = findinDB2_2(targetAC,q8(2))
        submit(ans)
        q9 = new_question_info
        targetAC = find_flight_low(q9(1))
        ans = findinDB2_4(targetAC,q9(2))
        submit(ans)
        q10 = new_question_info
        targetAC = find_flight_low(q10(1))
        ans = findinDB2_2(targetAC,q10(2))
        submit(ans)
        q11 = new_question_info
        targetAC = find_flight_low(q11(1))
        ans = findinDB2_2(targetAC,q11(2))
        submit(ans)
        q12 = new_question_info
        targetAC = find_flight_low(q12(1))
        ans = findinDB2_4(targetAC,q12(2))
        submit(ans)
        q13 = new_question_info
        targetAC = find_flight_low(q13(1))
        ans = findinDB2_5(targetAC,q13(2))
        submit(ans)
        q14 = new_question_info
        targetAC = find_flight_low(q14(1))
        ans = findinDB2_3(targetAC,q14(2))
        submit(ans)
        q15 = new_question_info
        targetAC = find_flight_low(q15(1))
        ans = findinDB2_2(targetAC,q15(2))
        submit(ans)
        q16 = new_question_info
        targetAC = find_flight_low(q16(1))
        ans = findinDB2_2(targetAC,q16(2))
        submit(ans)
        scanpath_low
        d1 = seedot
        if d1 == 1
        report
        end
        scanpath_low
        d2 = seedot
        if d2 == 1
        report
```

```
end
scanpath_low
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.4 Scenario 122

The chart representing scenario 122 is shown in Figure C3. The algorithm follows after the chart.

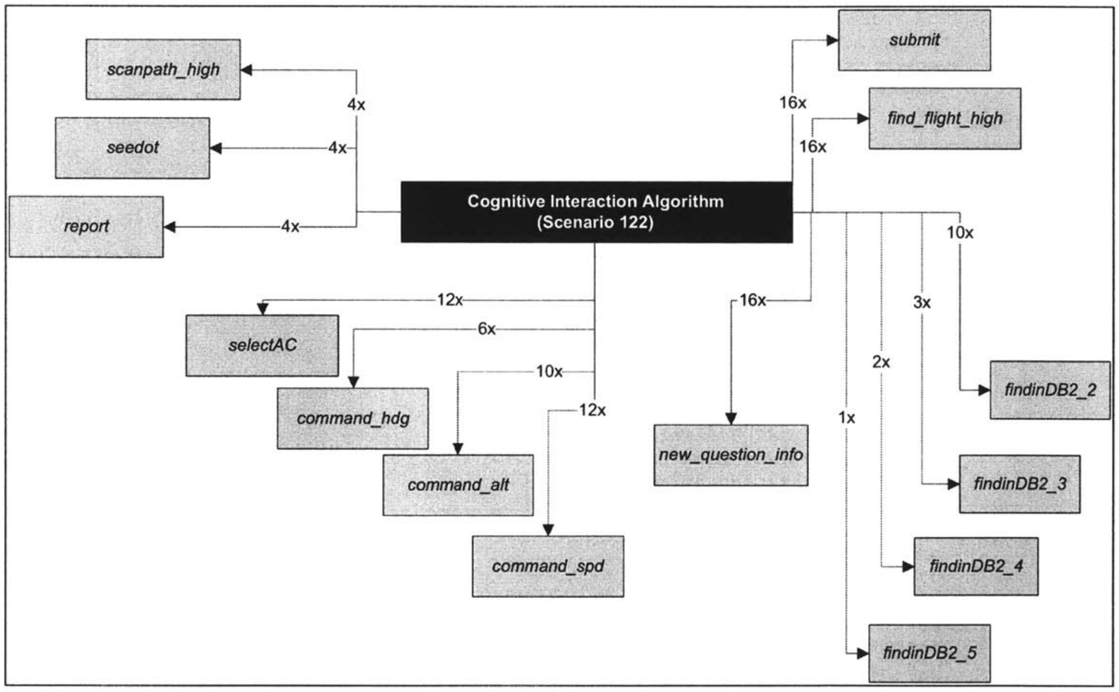


Figure C3. Scenario 122 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
```

```
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_hdg(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
ac8 = selectAC
command_alt(a8)
command_spd(a8)
ac9 = selectAC
command_alt(a9)
command_spd(a9)
ac10 = selectAC
command_hdg(a10)
command_alt(a10)
command_spd(a10)
ac11 = selectAC
command_alt(a11)
command_spd(a11)
ac12 = selectAC
command_hdg(a12)
command_spd(a12)
q1 = new_question_info
targetAC = find_flight_high(q1(1))
ans = findinDB2_3(targetAC,q1(2))
submit(ans)
q2 = new_question_info
targetAC = find_flight_high(q2(1))
ans = findinDB2_2(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_high(q3(1))
ans = findinDB2_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_high(q4(1))
ans = findinDB2_2(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_high(q5(1))
ans = findinDB2_2(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_high(q6(1))
ans = findinDB2_3(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_high(q7(1))
ans = findinDB2_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_high(q8(1))
ans = findinDB2_2(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_high(q9(1))
ans = findinDB2_2(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_high(q10(1))
ans = findinDB2_4(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_high(q11(1))
ans = findinDB2_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_high(q12(1))
```

```
ans = findinDB2_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB2_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB2_2(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB2_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB2_3(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.5 Scenario 211

The chart representing scenario 211 is shown in Figure C4. The algorithm follows after the chart.

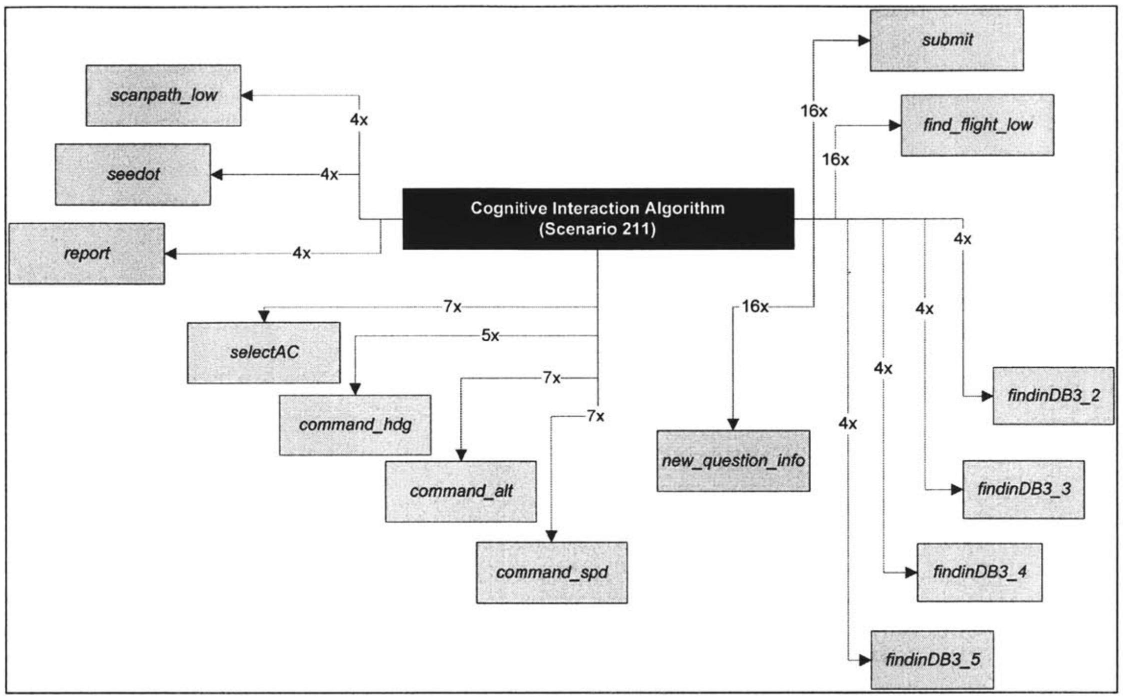


Figure C4. Scenario 211 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_hdg(a3)
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB3_2(targetAC,q1(2))
submit(ans)
```

```
q2 = new_question_info
targetAC = find_flight_low(q2(1))
ans = findinDB3_5(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB3_3(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB3_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB3_4(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB3_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB3_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB3_5(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB3_3(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB3_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB3_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB3_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB3_2(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB3_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB3_4(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB3_5(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
scanpath_low
```

```
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.6 Scenario 212

The chart representing scenario 212 is shown in Figure C5. The algorithm follows after the chart.

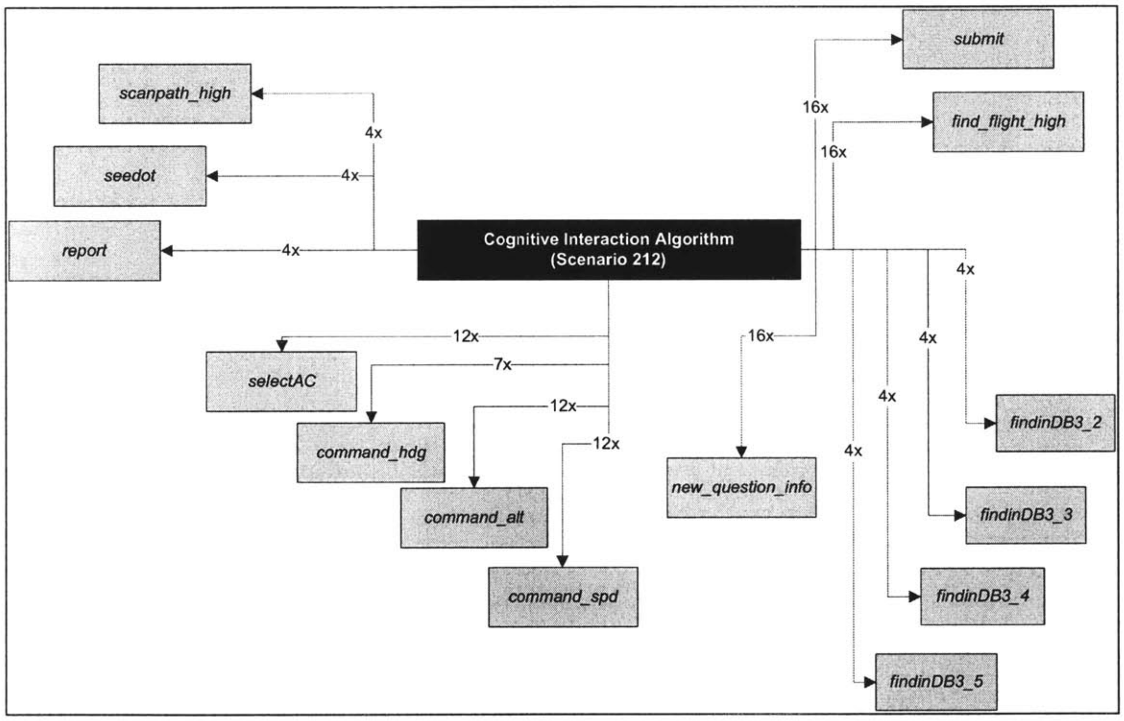


Figure C5. Scenario 212 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_alt(a4)
command_spd(a4)
```

```
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_hdg(a6)
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
ac8 = selectAC
command_hdg(a8)
command_alt(a8)
command_spd(a8)
ac9 = selectAC
command_alt(a9)
command_spd(a9)
ac10 = selectAC
command_hdg(a10)
command_alt(a10)
command_spd(a10)
ac11 = selectAC
command_alt(a11)
command_spd(a11)
ac12 = selectAC
command_alt(a12)
command_hdg(a12)
command_spd(a12)
q1 = new_question_info
targetAC = find_flight_high(q1(1))
ans = findinDB3_5(targetAC,q1(2))
submit(ans)
q2 = new_question_info
targetAC = find_flight_high(q2(1))
ans = findinDB3_3(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_high(q3(1))
ans = findinDB3_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_high(q4(1))
ans = findinDB3_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_high(q5(1))
ans = findinDB3_2(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_high(q6(1))
ans = findinDB3_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_high(q7(1))
ans = findinDB3_4(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_high(q8(1))
ans = findinDB3_4(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_high(q9(1))
ans = findinDB3_5(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_high(q10(1))
ans = findinDB3_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
```

```
targetAC = find_flight_high(q11(1))
ans = findinDB3_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_high(q12(1))
ans = findinDB3_2(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB3_3(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB3_4(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB3_5(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB3_4(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.7 Scenario 221

The chart representing scenario 221 is shown in Figure C6. The algorithm follows after the chart.

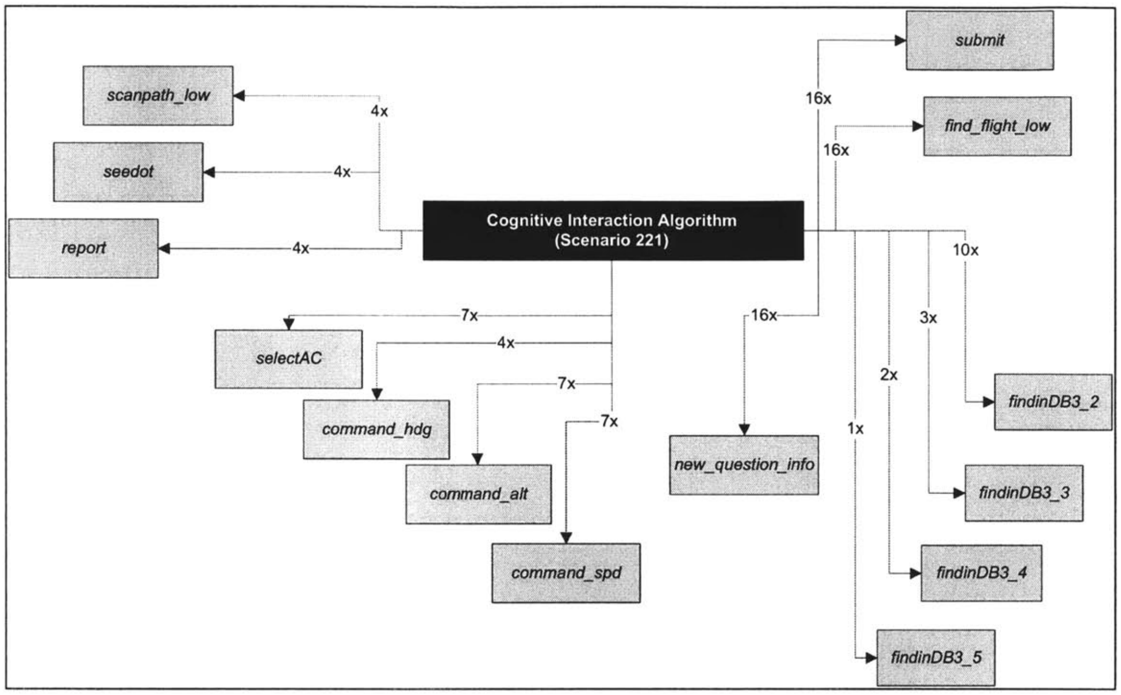


Figure C6. Scenario 221 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
```

```
targetAC = find_flight_low(q1(1))
ans = findinDB3_2(targetAC,q1(2))
submit(ans)
q2 = new_question_info
targetAC = find_flight_low(q2(1))
ans = findinDB3_3(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB3_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB3_2(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB3_3(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB3_2(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB3_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB3_2(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB3_4(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB3_2(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB3_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB3_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB3_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB3_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB3_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB3_2(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
```

```
report
end
scanpath_low
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.8 Scenario 222

The chart representing scenario 222 is shown in Figure C7. The algorithm follows after the chart.

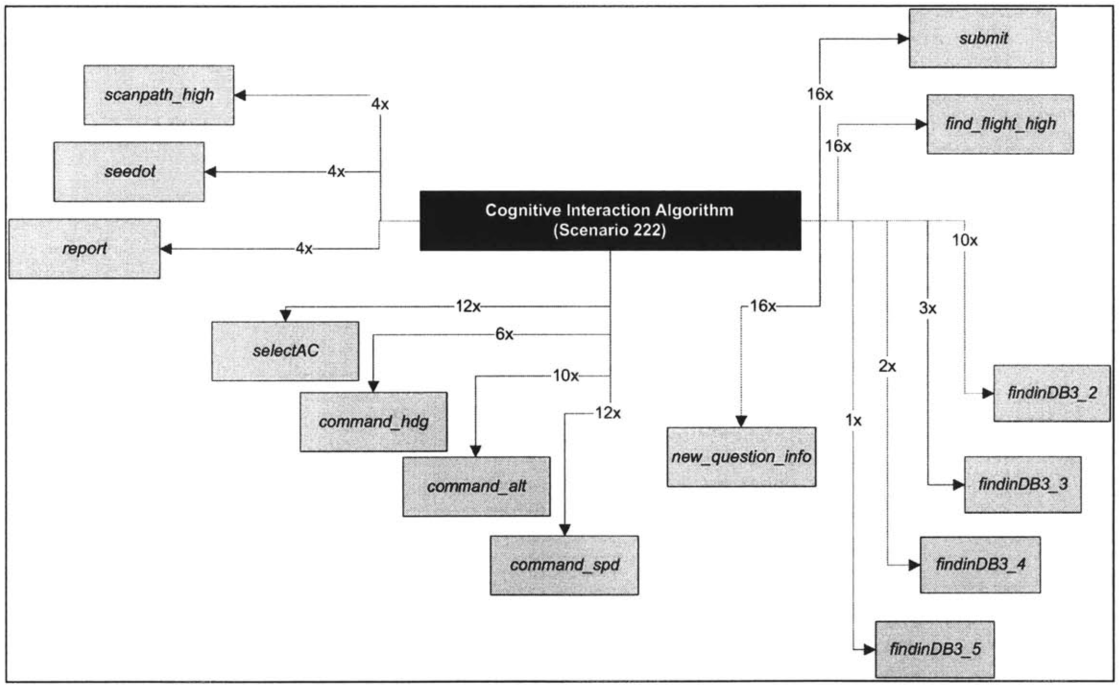


Figure C7. Scenario 222 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
```

```
    command_spd(a5)
    ac6 = selectAC
    command_hdg(a6)
    command_spd(a6)
    ac7 = selectAC
    command_hdg(a7)
    command_alt(a7)
    command_spd(a7)
    ac8 = selectAC
    command_alt(a8)
    command_spd(a8)
    ac9 = selectAC
    command_alt(a9)
    command_spd(a9)
    ac10 = selectAC
    command_hdg(a10)
    command_alt(a10)
    command_spd(a10)
    ac11 = selectAC
    command_alt(a11)
    command_spd(a11)
    ac12 = selectAC
    command_hdg(a12)
    command_spd(a12)
    q1 = new_question_info
    targetAC = find_flight_high(q1(1))
    ans = findinDB3_3(targetAC,q1(2))
    submit(ans)
    q2 = new_question_info
    targetAC = find_flight_high(q2(1))
    ans = findinDB3_2(targetAC,q2(2))
    submit(ans)
    q3 = new_question_info
    targetAC = find_flight_high(q3(1))
    ans = findinDB3_2(targetAC,q3(2))
    submit(ans)
    q4 = new_question_info
    targetAC = find_flight_high(q4(1))
    ans = findinDB3_2(targetAC,q4(2))
    submit(ans)
    q5 = new_question_info
    targetAC = find_flight_high(q5(1))
    ans = findinDB3_2(targetAC,q5(2))
    submit(ans)
    q6 = new_question_info
    targetAC = find_flight_high(q6(1))
    ans = findinDB3_3(targetAC,q6(2))
    submit(ans)
    q7 = new_question_info
    targetAC = find_flight_high(q7(1))
    ans = findinDB3_2(targetAC,q7(2))
    submit(ans)
    q8 = new_question_info
    targetAC = find_flight_high(q8(1))
    ans = findinDB3_2(targetAC,q8(2))
    submit(ans)
    q9 = new_question_info
    targetAC = find_flight_high(q9(1))
    ans = findinDB3_2(targetAC,q9(2))
    submit(ans)
    q10 = new_question_info
    targetAC = find_flight_high(q10(1))
    ans = findinDB3_4(targetAC,q10(2))
    submit(ans)
    q11 = new_question_info
    targetAC = find_flight_high(q11(1))
    ans = findinDB3_2(targetAC,q11(2))
    submit(ans)
    q12 = new_question_info
    targetAC = find_flight_high(q12(1))
    ans = findinDB3_4(targetAC,q12(2))
```

```
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB3_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB3_2(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB3_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB3_3(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.9 Scenario 311

The chart representing scenario 311 is shown in Figure C8. The algorithm follows after the chart.

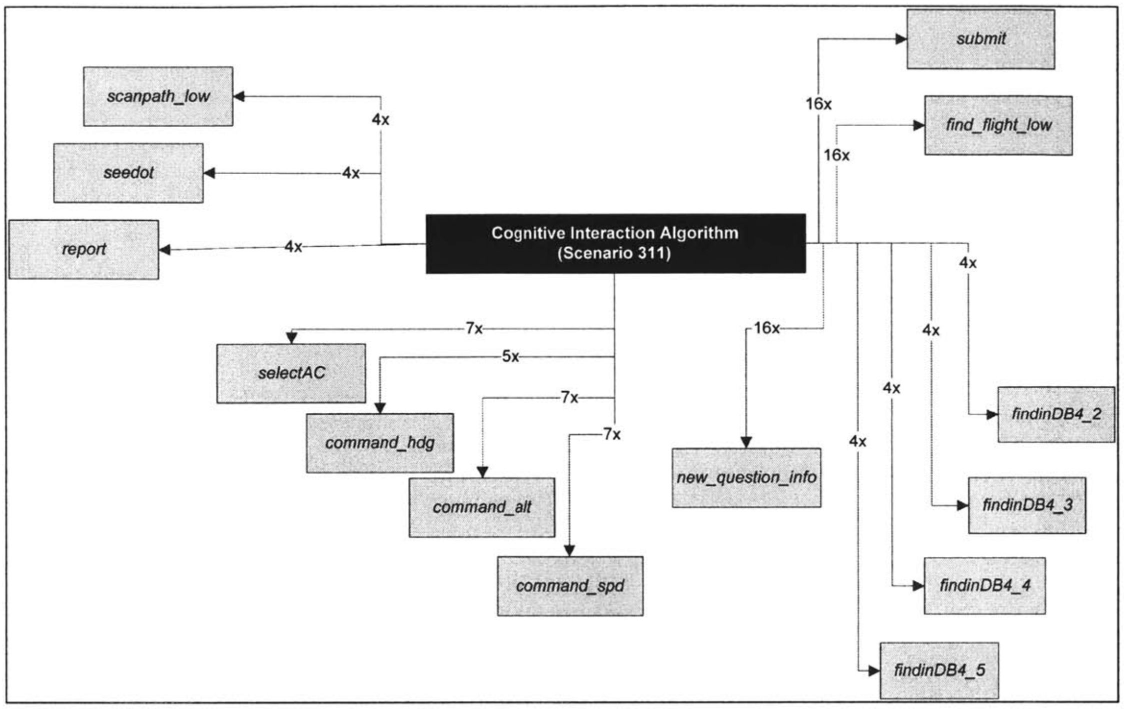


Figure C8. Scenario 311 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_hdg(a3)
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB4_2(targetAC,q1(2))
submit(ans)
q2 = new_question_info
```

```
targetAC = find_flight_low(q2(1))
ans = findinDB4_5(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB4_3(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB4_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB4_4(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB4_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB4_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB4_5(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB4_3(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB4_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB4_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB4_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB4_2(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB4_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB4_4(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB4_5(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
scanpath_low
d3 = seedot
```

```
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.10 Scenario 312

The chart representing scenario 312 is shown in Figure C9. The algorithm follows after the chart.

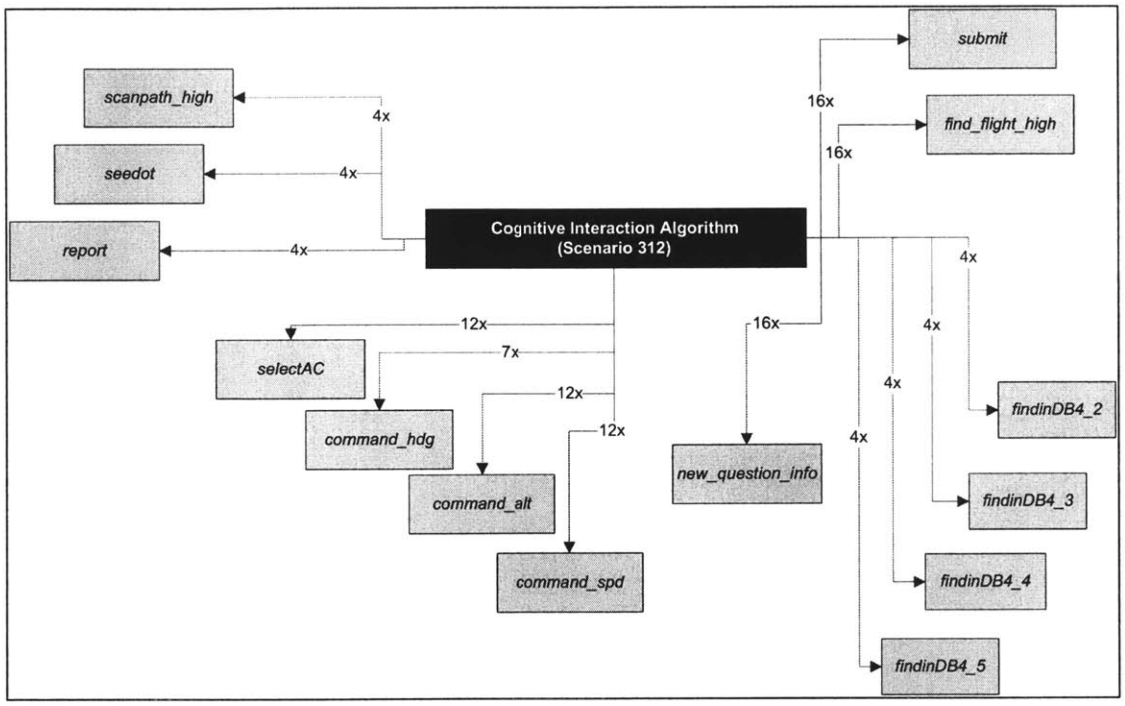


Figure C9. Scenario 312 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
```

```
        command_alt(a5)
        command_spd(a5)
        ac6 = selectAC
        command_hdg(a6)
        command_alt(a6)
        command_spd(a6)
        ac7 = selectAC
        command_hdg(a7)
        command_alt(a7)
        command_spd(a7)
        ac8 = selectAC
        command_hdg(a8)
        command_alt(a8)
        command_spd(a8)
        ac9 = selectAC
        command_alt(a9)
        command_spd(a9)
        ac10 = selectAC
        command_hdg(a10)
        command_alt(a10)
        command_spd(a10)
        ac11 = selectAC
        command_alt(a11)
        command_spd(a11)
        ac12 = selectAC
        command_alt(a12)
        command_hdg(a12)
        command_spd(a12)
        q1 = new_question_info
        targetAC = find_flight_high(q1(1))
        ans = findinDB4_5(targetAC,q1(2))
        submit(ans)
        q2 = new_question_info
        targetAC = find_flight_high(q2(1))
        ans = findinDB4_3(targetAC,q2(2))
        submit(ans)
        q3 = new_question_info
        targetAC = find_flight_high(q3(1))
        ans = findinDB4_2(targetAC,q3(2))
        submit(ans)
        q4 = new_question_info
        targetAC = find_flight_high(q4(1))
        ans = findinDB4_3(targetAC,q4(2))
        submit(ans)
        q5 = new_question_info
        targetAC = find_flight_high(q5(1))
        ans = findinDB4_2(targetAC,q5(2))
        submit(ans)
        q6 = new_question_info
        targetAC = find_flight_high(q6(1))
        ans = findinDB4_4(targetAC,q6(2))
        submit(ans)
        q7 = new_question_info
        targetAC = find_flight_high(q7(1))
        ans = findinDB4_4(targetAC,q7(2))
        submit(ans)
        q8 = new_question_info
        targetAC = find_flight_high(q8(1))
        ans = findinDB4_4(targetAC,q8(2))
        submit(ans)
        q9 = new_question_info
        targetAC = find_flight_high(q9(1))
        ans = findinDB4_5(targetAC,q9(2))
        submit(ans)
        q10 = new_question_info
        targetAC = find_flight_high(q10(1))
        ans = findinDB4_5(targetAC,q10(2))
        submit(ans)
        q11 = new_question_info
        targetAC = find_flight_high(q11(1))
        ans = findinDB4_2(targetAC,q11(2))
```

```
submit(ans)
q12 = new_question_info
targetAC = find_flight_high(q12(1))
ans = findinDB4_2(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB4_3(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB4_4(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB4_5(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB4_4(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.11 Scenario 321

The chart representing scenario 321 is shown in Figure C10. The algorithm follows after the chart.

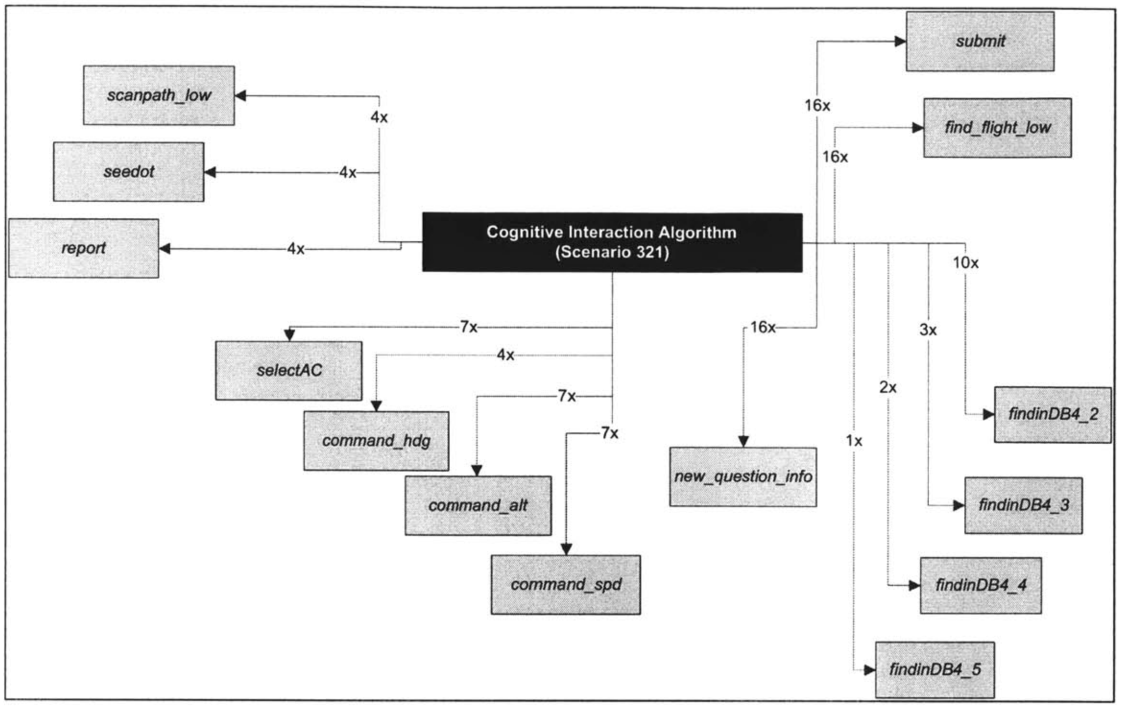


Figure C10. Scenario 321 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB4_2(targetAC,q1(2))
```

```
submit(ans)
q2 = new_question_info
targetAC = find_flight_low(q2(1))
ans = findinDB4_3(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB4_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB4_2(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB4_3(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB4_2(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB4_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB4_2(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB4_4(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB4_2(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB4_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB4_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB4_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB4_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB4_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB4_2(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
```

```
scanpath_low
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.12 Scenario 322

The chart representing scenario 322 is shown in Figure C11. The algorithm follows after the chart.

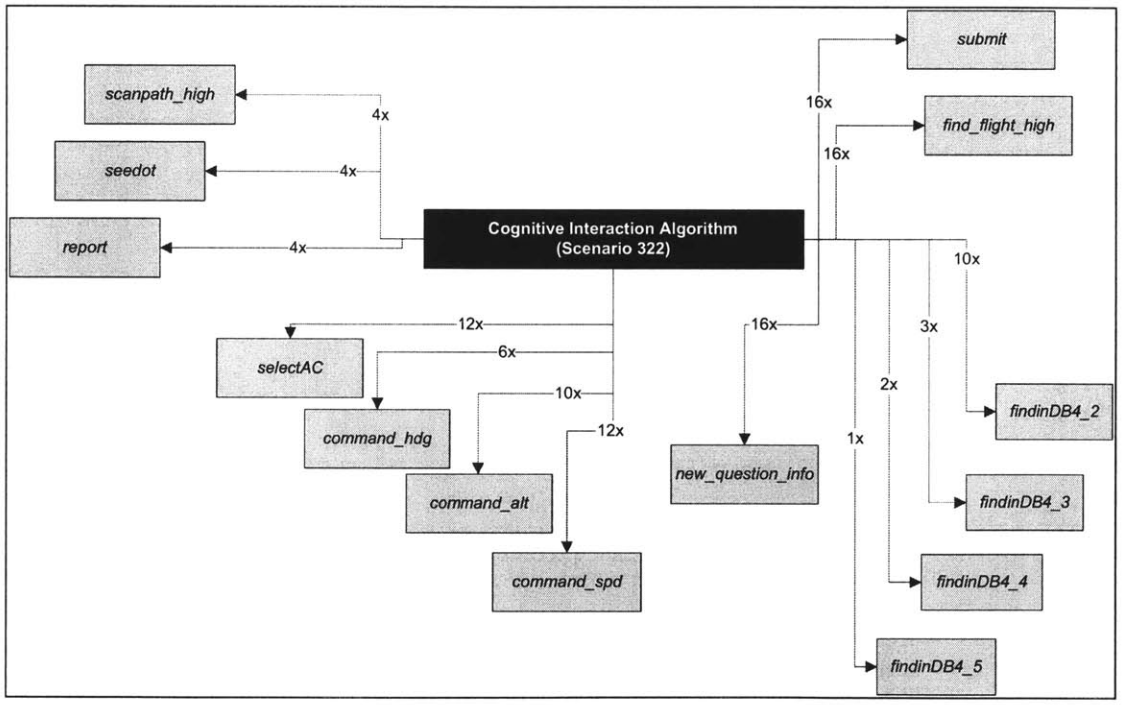


Figure C11. Scenario 322 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
```

```
          command_spd(a4)
          ac5 = selectAC
          command_hdg(a5)
          command_alt(a5)
          command_spd(a5)
          ac6 = selectAC
          command_hdg(a6)
          command_spd(a6)
          ac7 = selectAC
          command_hdg(a7)
          command_alt(a7)
          command_spd(a7)
          ac8 = selectAC
          command_alt(a8)
          command_spd(a8)
          ac9 = selectAC
          command_alt(a9)
          command_spd(a9)
          ac10 = selectAC
          command_hdg(a10)
          command_alt(a10)
          command_spd(a10)
          ac11 = selectAC
          command_alt(a11)
          command_spd(a11)
          ac12 = selectAC
          command_hdg(a12)
          command_spd(a12)
          q1 = new_question_info
          targetAC = find_flight_high(q1(1))
          ans = findinDB4_3(targetAC,q1(2))
          submit(ans)
          q2 = new_question_info
          targetAC = find_flight_high(q2(1))
          ans = findinDB4_2(targetAC,q2(2))
          submit(ans)
          q3 = new_question_info
          targetAC = find_flight_high(q3(1))
          ans = findinDB4_2(targetAC,q3(2))
          submit(ans)
          q4 = new_question_info
          targetAC = find_flight_high(q4(1))
          ans = findinDB4_2(targetAC,q4(2))
          submit(ans)
          q5 = new_question_info
          targetAC = find_flight_high(q5(1))
          ans = findinDB4_2(targetAC,q5(2))
          submit(ans)
          q6 = new_question_info
          targetAC = find_flight_high(q6(1))
          ans = findinDB4_3(targetAC,q6(2))
          submit(ans)
          q7 = new_question_info
          targetAC = find_flight_high(q7(1))
          ans = findinDB4_2(targetAC,q7(2))
          submit(ans)
          q8 = new_question_info
          targetAC = find_flight_high(q8(1))
          ans = findinDB4_2(targetAC,q8(2))
          submit(ans)
          q9 = new_question_info
          targetAC = find_flight_high(q9(1))
          ans = findinDB4_2(targetAC,q9(2))
          submit(ans)
          q10 = new_question_info
          targetAC = find_flight_high(q10(1))
          ans = findinDB4_4(targetAC,q10(2))
          submit(ans)
          q11 = new_question_info
          targetAC = find_flight_high(q11(1))
          ans = findinDB4_2(targetAC,q11(2))
```

```
submit(ans)
q12 = new_question_info
targetAC = find_flight_high(q12(1))
ans = findinDB4_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB4_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB4_2(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB4_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB4_3(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.13 Scenario 411

The chart representing scenario 411 is shown in Figure C12. The algorithm follows after the chart.

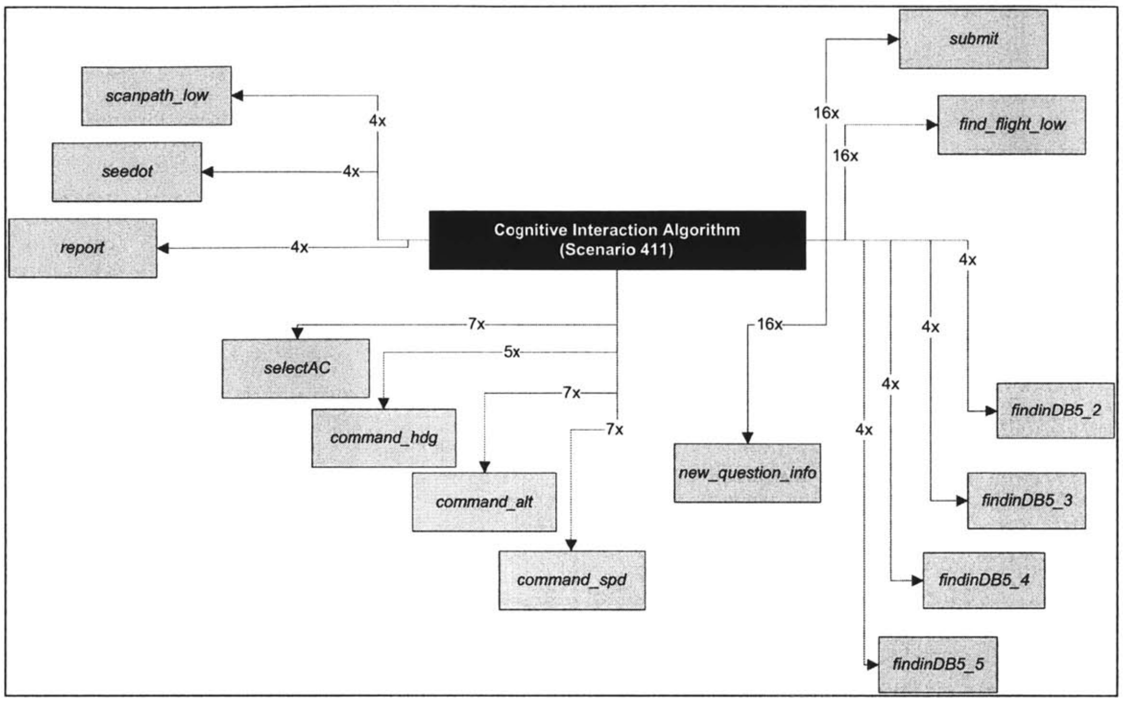


Figure C12. Scenario 411 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_hdg(a3)
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB5_2(targetAC,q1(2))
submit(ans)
q2 = new_question_info
```

```
targetAC = find_flight_low(q2(1))
ans = findinDB5_5(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB5_3(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB5_3(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB5_4(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB5_4(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB5_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB5_5(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB5_3(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB5_5(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB5_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB5_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB5_2(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB5_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB5_4(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB5_5(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
scanpath_low
d3 = seedot
```

```
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.14 Scenario 412

The chart representing scenario 412 is shown in Figure C13. The algorithm follows after the chart.

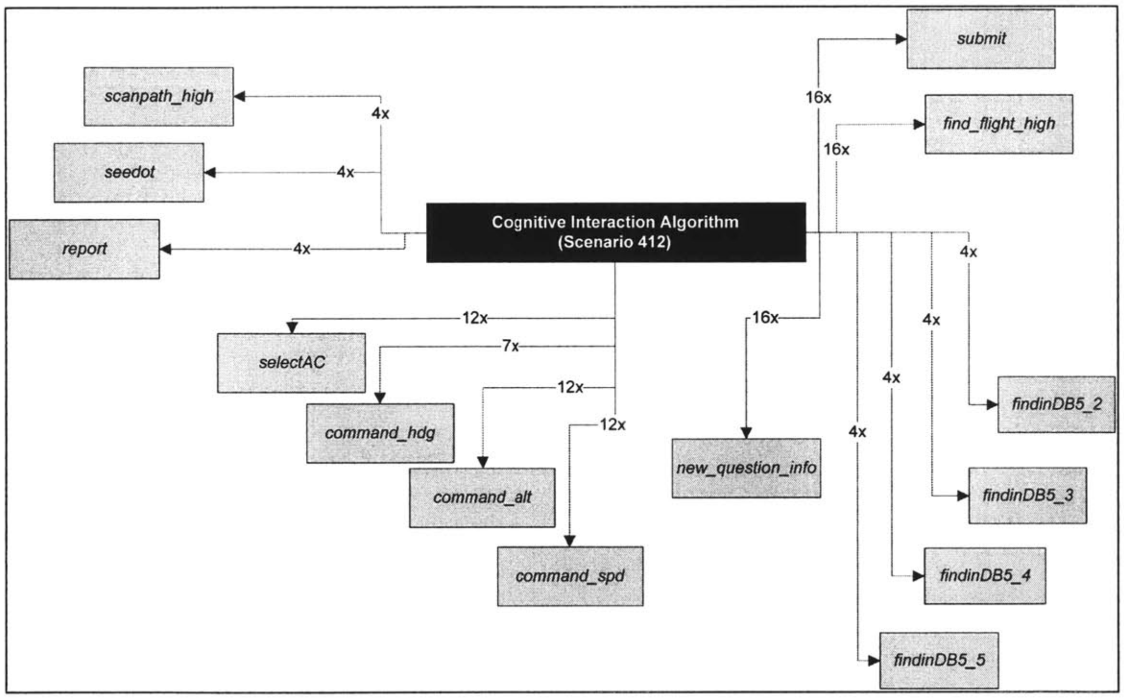


Figure C13. Scenario 412 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
```

```
        command_spd(a5)
        ac6 = selectAC
        command_hdg(a6)
        command_alt(a6)
        command_spd(a6)
        ac7 = selectAC
        command_hdg(a7)
        command_alt(a7)
        command_spd(a7)
        ac8 = selectAC
        command_hdg(a8)
        command_alt(a8)
        command_spd(a8)
        ac9 = selectAC
        command_alt(a9)
        command_spd(a9)
        ac10 = selectAC
        command_hdg(a10)
        command_alt(a10)
        command_spd(a10)
        ac11 = selectAC
        command_alt(a11)
        command_spd(a11)
        ac12 = selectAC
        command_alt(a12)
        command_hdg(a12)
        command_spd(a12)
        q1 = new_question_info
        targetAC = find_flight_high(q1(1))
        ans = findinDB5_5(targetAC,q1(2))
        submit(ans)
        q2 = new_question_info
        targetAC = find_flight_high(q2(1))
        ans = findinDB5_3(targetAC,q2(2))
        submit(ans)
        q3 = new_question_info
        targetAC = find_flight_high(q3(1))
        ans = findinDB5_2(targetAC,q3(2))
        submit(ans)
        q4 = new_question_info
        targetAC = find_flight_high(q4(1))
        ans = findinDB5_3(targetAC,q4(2))
        submit(ans)
        q5 = new_question_info
        targetAC = find_flight_high(q5(1))
        ans = findinDB5_2(targetAC,q5(2))
        submit(ans)
        q6 = new_question_info
        targetAC = find_flight_high(q6(1))
        ans = findinDB5_4(targetAC,q6(2))
        submit(ans)
        q7 = new_question_info
        targetAC = find_flight_high(q7(1))
        ans = findinDB5_4(targetAC,q7(2))
        submit(ans)
        q8 = new_question_info
        targetAC = find_flight_high(q8(1))
        ans = findinDB5_4(targetAC,q8(2))
        submit(ans)
        q9 = new_question_info
        targetAC = find_flight_high(q9(1))
        ans = findinDB5_5(targetAC,q9(2))
        submit(ans)
        q10 = new_question_info
        targetAC = find_flight_high(q10(1))
        ans = findinDB5_5(targetAC,q10(2))
        submit(ans)
        q11 = new_question_info
        targetAC = find_flight_high(q11(1))
        ans = findinDB5_2(targetAC,q11(2))
        submit(ans)
```

```
q12 = new_question_info
targetAC = find_flight_high(q12(1))
ans = findinDB5_2(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB5_3(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB5_4(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB5_5(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB5_4(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

C.15 Scenario 421

The chart representing scenario 421 is shown in Figure C14. The algorithm follows after the chart.

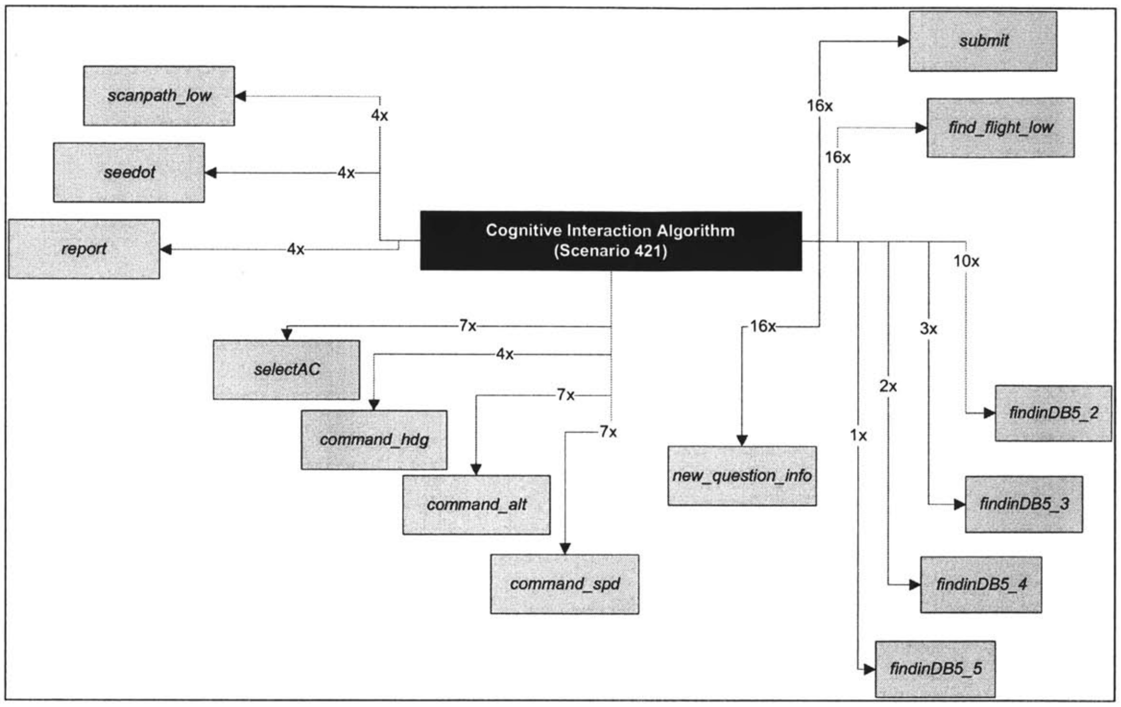


Figure C14. Scenario 421 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_hdg(a2)
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
command_alt(a5)
command_spd(a5)
ac6 = selectAC
command_alt(a6)
command_spd(a6)
ac7 = selectAC
command_hdg(a7)
command_alt(a7)
command_spd(a7)
q1 = new_question_info
targetAC = find_flight_low(q1(1))
ans = findinDB5_2(targetAC,q1(2))
```

```
submit(ans)
q2 = new_question_info
targetAC = find_flight_low(q2(1))
ans = findinDB5_3(targetAC,q2(2))
submit(ans)
q3 = new_question_info
targetAC = find_flight_low(q3(1))
ans = findinDB5_2(targetAC,q3(2))
submit(ans)
q4 = new_question_info
targetAC = find_flight_low(q4(1))
ans = findinDB5_2(targetAC,q4(2))
submit(ans)
q5 = new_question_info
targetAC = find_flight_low(q5(1))
ans = findinDB5_3(targetAC,q5(2))
submit(ans)
q6 = new_question_info
targetAC = find_flight_low(q6(1))
ans = findinDB5_2(targetAC,q6(2))
submit(ans)
q7 = new_question_info
targetAC = find_flight_low(q7(1))
ans = findinDB5_2(targetAC,q7(2))
submit(ans)
q8 = new_question_info
targetAC = find_flight_low(q8(1))
ans = findinDB5_2(targetAC,q8(2))
submit(ans)
q9 = new_question_info
targetAC = find_flight_low(q9(1))
ans = findinDB5_4(targetAC,q9(2))
submit(ans)
q10 = new_question_info
targetAC = find_flight_low(q10(1))
ans = findinDB5_2(targetAC,q10(2))
submit(ans)
q11 = new_question_info
targetAC = find_flight_low(q11(1))
ans = findinDB5_2(targetAC,q11(2))
submit(ans)
q12 = new_question_info
targetAC = find_flight_low(q12(1))
ans = findinDB5_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_low(q13(1))
ans = findinDB5_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_low(q14(1))
ans = findinDB5_3(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_low(q15(1))
ans = findinDB5_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_low(q16(1))
ans = findinDB5_2(targetAC,q16(2))
submit(ans)
scanpath_low
d1 = seedot
if d1 == 1
report
end
scanpath_low
d2 = seedot
if d2 == 1
report
end
```

```
scanpath_low
d3 = seedot
if d3 == 1
report
end
scanpath_low
d4 = seedot
if d4 == 1
report
end
```

C.16 Scenario 422

The chart representing scenario 422 is shown in Figure C14. The algorithm follows after the chart.

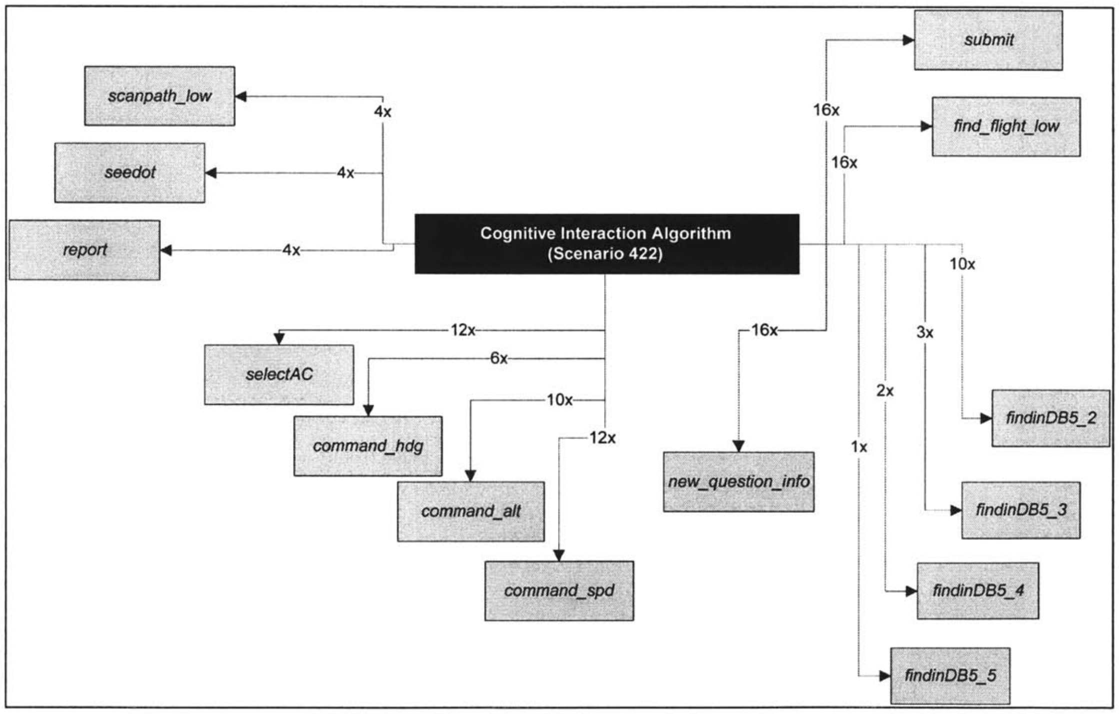


Figure C15. Scenario 422 algorithm function calls

```
ac1 = selectAC
command_alt(a1)
command_spd(a1)
ac2 = selectAC
command_alt(a2)
command_spd(a2)
ac3 = selectAC
command_alt(a3)
command_spd(a3)
ac4 = selectAC
command_hdg(a4)
command_alt(a4)
command_spd(a4)
ac5 = selectAC
command_hdg(a5)
```

```
    command_alt(a5)
    command_spd(a5)
    ac6 = selectAC
    command_hdg(a6)
    command_spd(a6)
    ac7 = selectAC
    command_hdg(a7)
    command_alt(a7)
    command_spd(a7)
    ac8 = selectAC
    command_alt(a8)
    command_spd(a8)
    ac9 = selectAC
    command_alt(a9)
    command_spd(a9)
    ac10 = selectAC
    command_hdg(a10)
    command_alt(a10)
    command_spd(a10)
    ac11 = selectAC
    command_alt(a11)
    command_spd(a11)
    ac12 = selectAC
    command_hdg(a12)
    command_spd(a12)
    q1 = new_question_info
    targetAC = find_flight_high(q1(1))
    ans = findinDB5_3(targetAC,q1(2))
    submit(ans)
    q2 = new_question_info
    targetAC = find_flight_high(q2(1))
    ans = findinDB5_2(targetAC,q2(2))
    submit(ans)
    q3 = new_question_info
    targetAC = find_flight_high(q3(1))
    ans = findinDB5_2(targetAC,q3(2))
    submit(ans)
    q4 = new_question_info
    targetAC = find_flight_high(q4(1))
    ans = findinDB5_2(targetAC,q4(2))
    submit(ans)
    q5 = new_question_info
    targetAC = find_flight_high(q5(1))
    ans = findinDB5_2(targetAC,q5(2))
    submit(ans)
    q6 = new_question_info
    targetAC = find_flight_high(q6(1))
    ans = findinDB5_3(targetAC,q6(2))
    submit(ans)
    q7 = new_question_info
    targetAC = find_flight_high(q7(1))
    ans = findinDB5_2(targetAC,q7(2))
    submit(ans)
    q8 = new_question_info
    targetAC = find_flight_high(q8(1))
    ans = findinDB5_2(targetAC,q8(2))
    submit(ans)
    q9 = new_question_info
    targetAC = find_flight_high(q9(1))
    ans = findinDB5_2(targetAC,q9(2))
    submit(ans)
    q10 = new_question_info
    targetAC = find_flight_high(q10(1))
    ans = findinDB5_4(targetAC,q10(2))
    submit(ans)
    q11 = new_question_info
    targetAC = find_flight_high(q11(1))
    ans = findinDB5_2(targetAC,q11(2))
    submit(ans)
    q12 = new_question_info
    targetAC = find_flight_high(q12(1))
```

```
ans = findinDB5_4(targetAC,q12(2))
submit(ans)
q13 = new_question_info
targetAC = find_flight_high(q13(1))
ans = findinDB5_5(targetAC,q13(2))
submit(ans)
q14 = new_question_info
targetAC = find_flight_high(q14(1))
ans = findinDB5_2(targetAC,q14(2))
submit(ans)
q15 = new_question_info
targetAC = find_flight_high(q15(1))
ans = findinDB5_2(targetAC,q15(2))
submit(ans)
q16 = new_question_info
targetAC = find_flight_high(q16(1))
ans = findinDB5_3(targetAC,q16(2))
submit(ans)
scanpath_high
d1 = seedot
if d1 == 1
report
end
scanpath_high
d2 = seedot
if d2 == 1
report
end
scanpath_high
d3 = seedot
if d3 == 1
report
end
scanpath_high
d4 = seedot
if d4 == 1
report
end
```

7958-89