

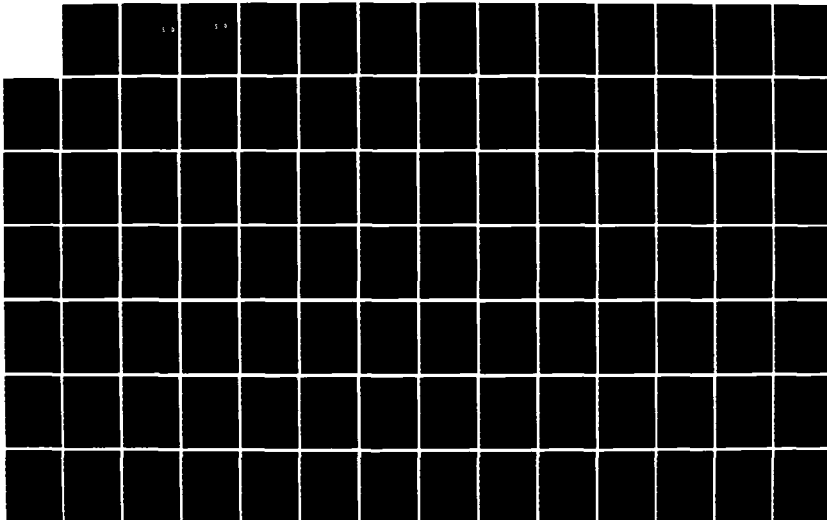
AD-A163 947

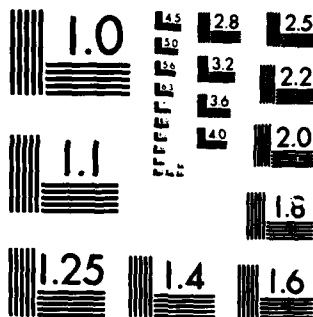
REASONING IN REAL-TIME FOR THE PILOT ASSOCIATE: AN
EXAMINATION OF A MODEL.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... D O NORMAN
DEC 85 AFIT/GCS/ENG/85D-12 F/G 6/4

1/2

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A163 947

DTIC FILE COPY



DTIC
ELECTE
FEB 12 1986

REASONING IN REAL-TIME FOR THE
PILOT ASSOCIATE:

AN EXAMINATION OF A MODEL BASED APPROACH TO
REASONING IN REAL-TIME FOR ARTIFICIAL
INTELLIGENCE SYSTEMS USING A
DISTRIBUTED ARCHITECTURE

THESIS

Douglas O. Norman, Captain, USAF
AFTT/GCS/ENG/85D-12

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

86 2 11 127

AFIT/GCS/ENG/85D-12

DTIC
ELECTE
FEB 12 1986
S D D

REASONING IN REAL-TIME FOR THE
PILOT ASSOCIATE:

AN EXAMINATION OF A MODEL BASED APPROACH TO
REASONING IN REAL-TIME FOR ARTIFICIAL
INTELLIGENCE SYSTEMS USING A
DISTRIBUTED ARCHITECTURE

THESIS

Douglas O. Norman, Captain, USAF
AFIT/GCS/ENG/85D-12

Approved for public release; distribution unlimited

AFIT/GCS/ENG/85D-12

REASONING IN REAL-TIME FOR THE PILOT ASSOCIATE:
AN EXAMINATION OF A MODEL BASED APPROACH TO
REASONING IN REAL-TIME FOR ARTIFICIAL
INTELLIGENCE SYSTEMS USING A
DISTRIBUTED ARCHITECTURE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Douglas O. Norman, B.S.

Captain, USAF

December 1985

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited



Preface

I have many people to thank for their support and help during this thesis effort. Most importantly, my wife Jan and my two children, Michael and Joseph. They all have suffered some from my lack of attention to them and their needs; yet despite this, they have understood and given me lots of love.

Steve Cross is a superlative advisor who has the nerve to allow his students to explore new areas (areas where he may even feel uncomfortable when the results start coming in).

Gary Lamont was always "ready, willing, and able" to discuss and argue. Difficult as his classes were (I'm probably one of the few in AFIT history to take four courses from him), I probably learned the most from these classes as they provided a firm computational theory foundation. He and his doctoral student, Jim McManama, helped me to understand some key issues which are at the heart of this thesis.

Rob Bahnij helped me to understand the tactical flying game. An F-16 instructor pilot, Rob introduced a part of flying to me for which (as a "lowly" private pilot) I had no real feeling. Without his introduction, I wouldn't understand the problems of the front-line single-seat pilot.

Since the area of this thesis is nominally the Pilot Associate, this would be less than ideal.

Why did I do this thesis? Today, many assert that they will build "real-time" Artificial Intelligence systems which will do all sorts of wonderful things. Building these systems seems to be an ad-hoc affair where one builds a system; runs it; discovers it won't work in "real-time" except on toy problems; then builds another system. The lessons learned tend to be of the type: "get a faster computer," or "it'll be alright when parallel computers are perfected." The algorithms used tend to go unexamined. I found my education in the theory-of-computation sequence prepared me to approach the real-time AI problems from a slightly different perspective than that which I found in the literature or as presented by many government contractors. This perspective seemed to highlight some errors in thought that they exhibited. This seemed to make "real-time AI" ideal for a thesis as it is both interesting and timely.

— Douglas O. Norman

Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
I. Introduction	1
The Pilot Associate	1
A Computation Issue	2
The Task of "Planning"	5
Real-Time: What Is It?	8
Problem Statement	10
Scope	11
Approach	12
Overview of the Thesis	12
II. Distributing a Task Among Processors: A Data-Flow Approach	14
Background	14
An Analysis	17
Summary	25
III. A Distributed Data-Flow Constraint Propagation Architecture	26
A Game Problem	26
Parallel Processing in DDAFCON	33
The "Strong Components" Method	35
The "Parallel Paths" Method	40
Execution Control	46
A Complexity Analysis	50
Why Not Rules?	60
Summary	62

	Page
IV. A DDAFCON Primer	65
Data Structure	65
Agent Structure	66
Blackboard Structure	67
Packet Structure	72
Agent-List	72
Data-Flow Network Structure	72
Data-Flow Description Method (DFDM)	74
Using DDAFCON	84
V. Flight Planning Application Using DDAFCON	89
Background	89
Results	92
Scenario 1	96
Scenario 2	102
Estimating the Overhead	104
Summary	107
VI. Conclusions and Recommendations	109
Discussion and Conclusions	109
Recommendations	113
Appendix A: An Abstract Model of Planning and a Proof of "Planning" as an NP-Complete Problem	117
Appendix B: Flight Planning Application Code	121
Bibliography	141
Vita	145

List of Figures

Figure	Page
1. The Data-Flow Network for the Example Problem .	32
2. "Strongly Connected Components" Network Partition and Schedule	39
3. The Parallel-Paths Algorithm	43
4. "Parallel-Paths" Network Partition and Schedule.	44
5. Schedule Produced on Altered Set of Agents . . .	45
6. A Macro View of Data-Flow Through the "Game" Problem	48
7. Diagrammatic Representation of Dependency-Set Interrelations	54
8. Data-Flow Network for $c = a/b$	56
9. Constraint-Set for $c = a/b$	56
10. Overlapping Constraint-Sets	57
11. Blackboard Segment Structure	68
12. Packet Structure	72
13. DFDM Declaration for Distance = Rate * Time . .	73
14. Graph Segment Produced for Figure 13	73
15. Major Constraint-Set for Flight Plan Application	94
16. Completion Times and Speedup for Scenario 1 Using SCM	97
17. Completion Times and Speedup for Scenario 1 Using PPM	98
18. Completion Times and Speedup for Scenario 2 Using SCM	99

Figure	Page
19. Completion Times and Speedup for Scenario 2 Using PPM	100
20. Processor Utilization for Both Scenarios	102
21. Overhead Estimates for Both Scenarios	106

List of Tables

Table	Page
1. Execution of the Example "Game"	49
2. Data-Flow and Message-Flow for Single Constraint-Set on $c = a/b$	57
3. Data-Flow and Message-Flow for Overlapping Constraint-Sets on $c = a/b$	58
4. Agent Placement for Flight Plan Application . .	93

Abstract

The use of artificial intelligence (AI) techniques for pilot aiding in real-time (DoD Pilot Associate--PA) introduces some seemingly intractable problems. Most algorithms which will be used in a PA are search intensive with exponential-order-time-complexities. The thesis outlines the problem, and explores the possibility of using parallel processing for minimizing the computational costs.

Methods for, and results from, distributing data and functions among many processors are presented. An extension to the blackboard structure used in many AI systems, called a "virtual blackboard" is introduced. The results and techniques are bundled into a prototype distributed data-flow constraint-network application system called DDAFCON. DDAFCON is discussed, and a flight-planning application is presented using DDAFCON. The results are discussed.

It is shown that the maximum speedup due to parallel processing is bounded by the number of processors which a problem solution can accept. An algorithm is presented for finding the maximum number of parallel pieces of a problem solution. In general, distributing an intractable problem over many processors is unlikely to result in a tractable problem. Reasons for this are explored.

The thesis concludes that generalized AI structures and reasoning paradigms (specifically, "deep models"), although seemingly required for the performance level desired for applications such as the PA, may not be realizable for many real-time applications. A change in emphasis for developing real-time AI systems is suggested.

REASONING IN REAL-TIME FOR THE PILOT ASSOCIATE:
AN EXAMINATION OF A MODEL BASED APPROACH TO REASONING
IN REAL-TIME FOR ARTIFICIAL INTELLIGENCE SYSTEMS
USING A DISTRIBUTED ARCHITECTURE

I. Introduction

The Pilot Associate

The Air Force recently announced plans to incorporate techniques from a sub-field of computer science called "artificial intelligence" into fighter aircraft to aid the pilot in performing his mission. The techniques include methods which automate reasoning and problem solving (31; 37).

The techniques are embodied in a set of computer programs called a "Pilot Associate" (PA), which is the Air Force's component of the DOD Strategic Computing Program (13). At this time, the precise definition of the PA remains undetermined; however, the goal of the PA is to raise and maintain the "situation awareness" of the single seat fighter pilot by reducing his work load and enabling him to concentrate on the global mission tasks. Many of the assorted independent details of operating the aircraft, which are currently integrated by the pilot, will be integrated by the PA. Examples of the types of tasks

which the PA would need to perform include: intelligent navigation, mission planning (and dynamic re-planning when unforeseen conditions dictate), intelligent internal system monitoring, emergency procedure aids, standard procedure monitoring, threat assessment, and tactical advisement.

A Computation Issue

Most of the AI programs which will be used to automate the high-level cognitive-type tasks in the cockpit are composed of search intensive algorithms which fall into the class of algorithms known as "hard" problems. Formally, this group is defined as "NP-Complete," and all are characterized as having exponential order time-complexity which, in the worst case, involves complete enumeration of all possible solutions (1). At this date, no algorithms faster than exponential time-complexity are known or anticipated.

The schema which can be used to develop a mental picture of the behavior of these algorithms is a decision search-tree (27). Every increment in depth in the tree gives an exponential increase in the number of tree nodes and a combinatorial increase in the number of potential paths to search. Clearly, the greater the branching factor (out-degree) of a node, the quicker the "exponential explosion." If the branching factor can be reduced to one,

the tree degenerates into a linear list; however, for any branching factor greater than one, the exponential nature of the algorithm remains, "tamed" somewhat but still present.

Search reduction techniques have been developed (A, A*, B, C). In essence, these techniques attempt to reduce the effective branching factor of the search. In AI, many researchers attempt to limit the problem's search-space by applying knowledge. Schank (33) developed the notion of a script to store sequential-pattern type knowledge in a data structure. Once a script is instantiated, the typical behavioral sequences are available for reasoning. Rouse's (22) observation that flying is very "scripty" allows the "usual" situations to be captured in a structure which can be accessed in time proportional to the number of scripts (17). What happens if a situation doesn't fit a script? If no other techniques are available, either the machine can give a "default" answer which may or may not be appropriate, or (if a general reasoner is available) it must reason about the situation from a model of the environment. Immediately, we're back into the exponential time complexity problem; and this is probably the time that help is most needed, when the situation is not nominal.

The necessity to use algorithms of this type raises the issue of whether real-time performance can be extracted reliably from these systems. From the work that has been

presented to date on the PA and other similar projects, no one has considered the ramifications of using NP-Complete algorithms in a real-time environment. Analyses found to date include a contractor's analysis of the relative speed of various expert system building tools; no thought was given to the class of algorithms which would be developed and used with the tool. This is potentially devastating.

Mission planning, replanning, and emergency procedure monitoring are a few of the areas in which AFIT has considered the needs of a PA and the methods by which the aid would be delivered (2; 10; 11; 23). All are computationally hard problems. Mission planning and replanning are shown in Appendix A to be NP-Complete while emergency procedures are a type of diagnosis problem (5; 35) which also have been demonstrated to have exponential time-complexities. This general area should be a good one in which to explore architectures to support the real-time computation of these difficult problems, and to examine the complexion of systems which must run under such demands.

The approach adopted in this thesis investigation is to use "planning" as a vehicle to investigate the suitability of a general model-based reasoning engine for real-time reasoning. The area addressed examines the type of reasoning required given a failure to instantiate a script or a "canned plan." It has been argued that rule bases of

knowledge are not sufficiently deep representations of the aerospace "world" to capture the types of knowledge needed to reason in novel situations (10; 11). Rather, a computational model of the environment is built which is rich enough to examine novel situations. This generality seems to be required of a PA which is to actually fly on missions rather than exist as a laboratory curiosity. The cost of the generality is shown to be quite high.

The Task of "Planning"

Planning, in the AI literature, is defined as ". . . deciding on the course of action before acting . . ." (8:515). In a programming sense this can be envisioned as a search through a decision tree to discover a successful path. The nodes in the tree represent the various operators available to bring the items being planned over into some acceptable tolerance. The path described by the excursion through the tree is the "plan." The order of the operators in the path string is the order in which the operators are applied to the domain items to effect the plan.

Planning need not be a blind search, as implied by the above description. Rather, many schemes exist which limit the branching in the tree, or reduce the effective depth. Besides the "scripts" method already mentioned, "hierarchical" planners have been developed [e.g. (36)].

This approach limits the complexity by concentrating on developing plans by abstraction level refinement. In these methods, the first item planned is the planning itself (meta-planning). Thus, details are not considered at the highest level, but are considered once the high-level plan is in place. This is an instantiation of the well-known "divide-and-conquer" paradigm of software engineering. It eliminates time being spent on details which will later be discarded when the plan is changed at the higher levels.

Wilensky (41) recognized that planning about planning (meta-planning) could be guided (search space reduced) further by using appropriate "meta-themes." A "meta-theme" provides a context to work within. Wilensky maintains that meta-themes are important for common-sense reasoning. A meta-theme such as "satisfy hunger" might be important to a person who has gone without food for an extended period of time. This meta-theme would guide the planning that the person undertook.

Other schemes exist for controlling the complexity besides applying specific knowledge. Discrimination nets (3; 8) is an AI programming technique which has the effect of changing (for example) rule searches from depth-first to breadth-first. The success of this method is dependent on the breadth of the rule-tree.

A rich literature exists for "planning" in artificial intelligence, and many different metaphores have been used. Planning seems to be a generic process. The technique one uses to plan the activities in a day is the same technique used to plan the route of a car trip. Our current position is noted, where we wish to be is noted, then a set of operators is assembled which will get us to the goal. This capability is a crucial one for a PA to possess. Many times missions must be changed and re-planned during the course of the mission itself. Whether due to poor intelligence, late developments, or whatever, replanning demands that the pilot replan (at the very minimum) the routes he will fly to make the new time-on-target for the new target, as well as estimate whether he has sufficient resources aboard. And this task must be performed at the same time he is flying the aircraft (hardly an environment conducive to careful and thoughtful evaluation of the options).

As argued above, replanning is a task which must be incorporated into a PA; unfortunately, planning is an NP-Complete problem (see Appendix A) and likewise is replanning. Since the replanning tasks run in real-time, the total architecture of the system which will run the algorithm should be considered at the same time. A systems approach is required.

Real-Time: What Is It?

"Real-Time" is a much referenced but seldom defined entity. There is little in the AI literature concerning algorithm complexities, and virtually nothing about real-time performance. Some researchers consider these issues uninteresting and misdirected (4) since it tends to focus discussion away from the "pure" AI issues towards implementation issues. However, with the current interest in applying these techniques to domains which require adequate performance in time critical situations, the issue of "what is real-time" must be addressed.

In a recently published paper which addressed real-time in AI, O'Reilly and Cromarty (28) provide a precise definition of real-time: it is the time interval in which an answer must be delivered. That is, if

$$t_{r_i} \leq T_C - T_R \quad (1)$$

where

T_R is the time when the system requests a response

T_C is the time at which the response must be completed

t_{r_i} is the time it takes r_i to complete

then the system can be said to be "real-time." Their definition has been adopted.

Again, as pointed out by O'Reilly and Cromarty, the time interval in which the response is required is

generally not known a priori. In a theoretical sense, one must guarantee that the algorithm halts with an answer in time "t" given an arbitrary input and an arbitrary current state. Unfortunately, it is not possible to guarantee both an optimal answer and a halting in an arbitrary time interval. Given the non-deterministic nature of search algorithms (and hence of most AI algorithms), no nominal time-to-complete can be calculated. If a sub-optimal answer is acceptable, can an algorithm be found which can produce an answer in time "t" which is bounded by some relative epsilon? Many NP-Complete problems have known, epsilon-bounded approximation algorithms (20) which run in polynomial times; unfortunately, the set-covering problem does not. Since it has been shown that the planning problem is polynomially reducible to the set-covering problem (Appendix A), there probably does not exist an epsilon-bounded approximate algorithm for planning which is not NP-Complete itself. For "planning," probably the best that can be hoped for when using this algorithm in a real-time environment is an unbounded approximation if the time to converge on an answer is greater than the time available. Obviously, one prefers that the approximation move monotonically closer to the correct answer the longer the algorithm is allowed to work (i.e. the estimate is a function of "t"), but this can not be guaranteed.

Problem Statement

It has been argued by some researchers in the AI field that rule based systems are insufficient for a general reasoning system, and that a model-based approach must be adopted to get the generality needed for in-depth reasoning about a domain (10; 11).

Rule-based systems are known to degenerate rapidly at their knowledge borders. It is also difficult to predict the behavior of the system due to the interactions among the rules (43). Model-based reasoning systems incorporate mathematical and other symbolic representations of the system to be reasoned about. Pieces may easily be added or subtracted from the model and the effects predicted. A model permits unusual problems to be reasoned about since the effect of a condition may be propagated through the model and the results noted. As a general statement: rules tend to be "about" the system while models "represent" the system and simulate its properties.

This thesis effort examines a model-based general reasoning system using constraint satisfaction.¹

The system implements a distributed architecture to minimize the computational costs and to examine the real-time potential of such a system using a domain model. The

¹A common reasoning approach used in AI. For example, see Waltz (39) or Sussman (38).

research is driven by the fact that systems which can not provide real-time service can not be used in the cockpit.

Scope

Since the aim of this investigation is to explore possible software and hardware architectures for the distributed computation of search intensive algorithms, this effort examines the potential of parallel processing, describes a system with the appropriate characteristics and shows the results of using this novel architecture on an example application.

A research tool is developed for analyzing the parallelism in a problem which can be exploited. This tool also implements a parallel processing architecture and is used to investigate the performance of an AI type problem in a distributed environment.

The domain used for the example application is flight planning for general aviation. The aircraft used as a model is a Piper Warrior (PA-28-161). This is an area, and an aircraft, with which the author is familiar.

Many simplifying assumptions are made for the application, although none of these changes the character of the problem. These assumptions are detailed in the thesis section (Flight-planning Application) which discusses the specific application.

Approach

The specific approach involves an analysis of the nature of the environment in which AI systems perform, with an emphasis on real-time performance requirements. Current work and claims by investigators attempting to build real-time AI systems are examined. This analysis, detailed in the next chapter, suggested a data-flow approach to the architecture of the search-engine.

A model-based reasoning engine is examined which combines both mathematical and symbolic approaches. An application is presented which uses a planner which models an aircraft in its world. The structure of the planner, as a collection of small agents, has its genesis in Minsky's notion of the "society of mind" (26). The reasoning engine, built by the author, is an attempt to define an architecture that handles AI type problems while being designed to make maximum use of a multi-processor environment. A representation method, data-flow analyzer, and a network constructor are developed. As well, run-time support is provided for the environment and performance data is collected and displayed.

Overview of the Thesis

The thesis is divided into six chapters. Chapter one, of which this is part, introduces and motivates the subsequent chapters. It discusses the problems for an AI

system which must run in real-time and hints at the urgency for a rational evaluation of the capabilities of distributed AI. Chapter one closes with a brief introduction to the approach taken in this thesis for examining distributed AI. The second chapter discusses the problems and promise of distributing a task among a number of processors. Chapter three introduces DDAFCON (Distributed Data-Flow Constraint Network). DDAFCON was the author's trial structure for investigating problems in distributed AI. In the fourth chapter, the details of DDAFCON are explored. This chapter should be sufficient information for one to load and use the system developed. A flight planning application which uses the DDAFCON system is presented in chapter five. And, finally, chapter six contains conclusions and recommendations.

The system used for this development was a Symbolics 3670 running a version 5.3 operating system. All the code was interpreted, and dynamic binding of variables was assumed (this is the binding method used in version 5.3; version 6.0 uses a lexical binding).

II. Distributing a Task Among Processors:

A Data-Flow Approach

Parallel computing is recognized as a technique which offers (potentially) large performance gains given a hardware technology. This chapter explores the promise of parallel symbolic computation by reviewing the stated goals of many researchers in the area. This review serves to motivate the analysis which follows. The chapter closes by outlining some desirable properties for a tool which could be used to investigate the issues.

Background

At present, there is no known "accepted standard" for implementing parallel processing; in fact, little seems to be known about what tasks are appropriate to attempt on a parallel processing architecture. Flynn (15) coined some terms which have come to be accepted for categorizing processing schemes. These are the Single Instruction Single Data (SISD) scheme, the Single Instruction Multiple Data (SIMD) scheme, the Multiple Instruction Single Data (MISD) scheme, and the Multiple Instruction Multiple Data scheme. The SIMD scheme is useful for those situations where an operation is applied to a number of data items (which are essentially parallel by nature but have been

serialized by the single-instruction single-data (SISD) type machines normally used). This group is the most common parallel processing implementation and is chiefly used for array processing. Importantly, this class of parallel processor is not useful for general multiprocessing. The class of computer which is capable of multiprocessing is the MIMD computer.

Artificial intelligence applications may derive great computational benefits from parallel processing and thus increase what is their usual "dismal" time performance. Unfortunately, most "main stream" computer scientists studying parallel processing and distributed processing ignore the search intensive algorithms used in many AI applications. Some recent work (16) specifically ruled out NP-complete problems from consideration due to the potentially unbounded number of processors required and their non-deterministic nature. AI problems exhibit this non-deterministic nature; the algorithms used search for answers because no deterministic algorithms exist for calculating the answers. Some work exists for distributed processing in AI applications, most notably Lesser and colleagues at the University of Massachusetts at Amherst.

Lesser's work centers on synthesizing a system view of a problem given a number of separate pieces, rather than on distributing pieces of a single problem which can be processed in parallel for a performance increase. This is

a slight difference in emphasis from the approach taken in this effort. Lesser (24) explains that his approach is a "constructionist" one while breaking a problem up for distribution to a number of processors for parallel processing is, in his words, a "reductionist" approach. A reductionist approach is taken here since this is the method to use to increase the time performance of an algorithm, and this is what's required to make real-time applications possible.

Many ways are currently being explored for implementing parallel symbolic computations. Most ongoing approaches (see PARSYM Digest)¹ involve an MIMD scheme using a collection of "ready" processors which are to be given work as needed. Then, upon completion of their work, they go back into the "ready" bin until needed again. Suggestions for implementing these systems point to the "parallelism" present in the evaluation of interpreted Lisp code. By design, all arguments to functions in Lisp are evaluated prior to the operation being applied to the arguments. The idea is this: why not give each argument to a separate processor and process them at the same time? Thus to evaluate the list (CONS A B) (see PARSYM discussions Vol 1:1) A and B could be evaluated in parallel. Claims

¹The PARSYM Digest is an ARPANET mailing list originating at Stanford University for discussions about parallel symbolic computing.

abound for 2-3 orders of magnitude improvements in performance expected using such techniques. Special languages are being developed to exploit the parallel processing potential. Many approaches require the programmer to define explicitly the parallel pieces in the problem and to insure locality of interprocess communication (34). The failure to maximize the locality of reference in knowledge-based applications has been blamed for poor performance (40).

Larry Davies (from SUMEX-AIM), the editor of the PARSYM Digest, recently took a survey of current work being done on parallel symbolic computing. Some of the results were published in the PARSYM Digest (Vol 1:11). Examples of the results expected from the research to be (or being) performed were provided by the researchers. Jay Glickman (from AIDS) expects to get, at a minimum, 2-3 orders of magnitude performance improvement for the Autonomous Land Vehicle. Tony Li (USC) expects 2 orders of magnitude for handling a "blackboard" structure. Bert Halstead (MIT) feels an order of magnitude will be easy, even for small problems. Are the claims reasonable? An analysis can show what gains are possible, even for the previously discussed Lisp code above.

An Analysis

Consider (CONS A B), current single processor Lisp environments evaluate both A and B before performing the

CONS operation on the two. Therefore, if (to be as conservative as possible) one assumes that the CONS operation consumes an infinitesimal amount of time (say, epsilon), most of the time to evaluate this s-expression is taken in the argument evaluations. Let our parallel speedup be measured in relation to the linear time (18:3.4). Thus we will say that an order of magnitude speedup gives a 10x improvement. For this analysis one can immediately state that the time improvement for the parallel implementation of the above s-expression is

$$\frac{A_t + B_t + \epsilon}{\max(A_t, B_t) + \epsilon} \quad (2)$$

where

A_t is the time to evaluate A

B_t is the time to evaluate B

$A_t + B_t$ is the linear time

since epsilon approaches zero, its affect is negligible. This function has a maximum when $A_t = B_t$. Thus the maximum speedup for this expression is 2x over linear time. And this maximum may only be achieved when the arguments are closely matched in processing time. The greater the difference in processing time, the smaller the parallel advantage. For the general case

$$\frac{(\sum_{i=1}^n t_i) + \epsilon}{(\max_{i=1,n} t_i) + \epsilon} \quad (3)$$

and

$$\lim_{n \rightarrow \infty} \frac{(\sum_{i=1}^n t_i) + \epsilon}{(\max_{i=1,n} t_i) + \epsilon} = \infty \quad (4)$$

where

t_i is the time to evaluate the i th element

Since, in the limit, parallelism promises infinite advantage over linear processing (assuming no overhead), it is easy to assume that sterling performance will accrue to any problem where many processors are assigned.

The argument continues. Suppose one could achieve a reasonable advantage at each level of evaluation? Then the parallel benefit would rise exponentially. If one could achieve as little as 10 percent advantage, 10 percent compounded over, say, 100 levels is an advantage of 1.1^{100} or about 1370x (slightly better than 3 orders of magnitude). While this is true (mathematically) one must be realistic about how this could take place. Even if the division at each level were only into 2 parallel pieces, the number of processors required is appreciable. If one makes available every processor after it becomes blocked (can't continue

while waiting for a sub-piece to evaluate on another processor), $0.5 * 2^{100}$ processors are required; these are the number of items which are ready for evaluation in our example.¹ If less processors are used, say, $.25 * 2^{100}$, then each of the processors will have a queue of size 2 at the bottom of the evaluation tree. The queue size grows as an inverse function of the difference between the maximum number of processors which can be assigned and the number available. Thus if the number of processors is reduced by a factor of 2^x , the queue size grows by 2^x . The number is staggering. What has occurred in a swapping of space for time. But the space requirement is blowing up exponentially.

Two other arguments work against the massively parallel architectures: side-effects, and order-relations. Parallel implementations must be strictly devoid of side-effects if a system is going to run correctly when run in parallel. No evaluation chain may have an effect on another separate evaluation chain. Each must be independent of all others once spawned and processing (locality of reference). If not, the results would be indeterminate (7) or a chain would need to wait for another chain; thus implying an order relation between the two. Order-relations are probably the biggest argument against massive parallelism. Most

¹This is the number of leaves at the bottom of the evaluation tree.

calculations are based on things that proceeded them in a chain. This order must be preserved if a problem is to be solved! Say $C = f(B)$ and $B = g(A)$, B can't be evaluated until A 's value is known and $g(A)$ has been calculated. Clearly $f(B)$ and $g(A)$ can't benefit from parallel calculations no matter how many processors are available.

The argument implies that for every application there is a maximum amount of parallelism that can be realized. Consider this "gedanken" experiment. Imagine a problem and a problem solution. As in all solutions, there is a decision tree structure which describes the solution (make it as fine grained as desired). Now, assign a processor to every node of the tree. This is the absolute maximum number of processors which the solution will hold without any redundancy (redundancy adds no speed). Start the system going. The completion time required will be the minimum time possible given the specific hardware technology; and the speedup over the same solution executed on a single processor will represent the maximum speedup possible due to parallel execution.

Upon closer examination, massively parallel schemes can be analyzed by examining the tree formed by the expansion of the (in Lisp) s-expressions. The minimum time to evaluate the s-expression is no less than the largest path from the initial node to a terminal node. This is

equivalent to the critical path of a critical path method (CPM) network (19).

The advantage of parallel processing is a strong function of the problem being solved. The only way to gauge whether a problem is amenable to parallel processing is to examine the DATA-FLOW in the system, and this is what this thesis effort discusses. Many of the claims for multi-orders of magnitude improvement in performance without regard to the problem are probably a bit optimistic. If an algorithm can be found which will find every piece of a problem where there is some parallelism with another piece of the problem, the maximum number of processors that a problem can use can be found.

This analysis shows some stark facts. The absolute maximum speedup a problem can achieve, assuming no overhead costs, is absolutely bounded from the top by the number of processors that can be assigned to the problem. A necessary (not sufficient) condition for an order-of-magnitude speedup over a single processor, for any algorithm and any problem, is the need to be able to assign at least ten processors to the problem. To gain a two order-of-magnitude speedup, at least 100 processors must be able to be assigned to the problem. To repeat: the PROBLEM must be such that it supports at least 100 parallel pieces to potentially realize a two order-of-magnitude speedup. For

a three order-of-magnitude speedup, at least 1000 processors must be able to be assigned, etc.

Consider a claim that real-time performance can be achieved on a butterfly architecture¹ with 128 processors. Assume that

1. The real-time requirement is convergence in 250 milliseconds.
2. The system always converges.
3. The problem structure allows a perfectly balanced parallel decomposition (i.e. we can achieve the maximum benefit from the processors available).
4. There are no added overhead costs associated with the parallel processing.
5. The hardware technologies are the same in the single and multi-processor environments.

If the time to calculate the result, on a single processor, is greater than $128 * 250 \text{ milliseconds} = 32 \text{ seconds}$, the claim can be shown to be false immediately; and there was much assumed about the optimality of the problem structure. Performance claims, like the one examined here, are often heard in the pilot-aiding area; yet the analyses are seldom found.

¹A method of interconnecting many processors to form a multiprocessing environment developed by the BBN corporation. The U.S. Government is stressing its use for SCI projects.

As argued previously, a way to reduce the computation time of a problem, given a specific algorithm, is to exploit any inherent parallelism and distribute the pieces among a number of processors. To exploit the inherent parallelism in a problem, the data-flow in the system which solves the problem may be examined. Ultimately it's the data-flow which determines the parallelism in a problem. By performing an analysis of the problem off-line, much of the overhead at run-time can be reduced. This is the approach taken here, and it results in both a distributed data-base and distributed functions which act on the data-base. This approach implies a system constructed of fairly small individual functional units (fine grained). Given the sensitivity of real-time performance to the problem structure, a tool should be developed which can be used to analyze various problem-types and determine the expected performance improvement from parallel processing.

Recognizing that a cornerstone of AI is heuristic search, this investigation attempts to develop a tool which will permit the exploration of the major issues involved in parallel processing of AI problems. The tool must:

1. Have the ability to examine and exploit any inherent parallelism in the problem.
2. Implement the notion of a general purpose constraint satisfaction system (in this sense the system must solve a class of problems which resemble the linear-

programming problems found in Operations Research. However, the types of problems in this system are not restricted to "calculable" mathematical functions, but may be symbolic expressions. To allow this, the solution is found through a search process which is a basic (implicit) operation in the system).

3. Maintain (to the degree possible) locality of interprocess communication.

The structure of a Distributed Data-Flow Constraint Network (DDAFCON) tool is explored in the following two chapters.

Summary

This chapter has attempted to outline some of the promises and problems inherent in the parallel computation of AI problems. Parallel computing is shown not to be the panacea hoped for by many. Rather, it is shown that the nature of the problem-structure must be considered foremost. This suggests that trouble is afoot for applications which attempt to "run in real-time" when an insufficient analysis has been performed of the real-time requirements and the affinity of the problem for parallel execution.

III. A Distributed Data-Flow Constraint Propagation Architecture

This effort examines the interaction of both artificial intelligence systems (and the predominately search intensive problems AI deals with), and the distributed architectures which may help to run these algorithms efficiently. DDAFCON is an attempt to integrate these two areas. Initially, this chapter introduces DDAFCON's AI side. It explains the way in which DDAFCON handles AI-type problems. Although many AI techniques were used "behind the scenes" for jobs such as parsing functions to determine the qualitative effect of each term, this chapter stresses the job DDAFCON performs from the view of the user, and then explains some of the unique ways in which the job is executed. The balance of the chapter introduces some of the ways in which the architecture is designed as well as an analysis of the complexity.

A Game Problem

A reasonable way to explain the key ideas and algorithms in this development is to approach it by using a game problem. Once the general ideas are understood, a closer, more analytical view may be taken. The problem proposed is the kind of game that those in organizations

such as Mensa pride themselves in solving. The problem is provided, you, the reader, think about it. Keep track of the "protocol" you went through to find the answer.

The game is this: given the equations--

$$\begin{array}{lll} B = 2 * A & C = 3 * A & D = B + 2 \\ E = B - 2 & F = C + 3 & G = C - 3 \\ H = D + E & I = F + G & J = H + I \end{array} \quad (5)$$

each of which is written on a separate piece of paper. You may hold onto only one piece of paper at a time. The problem: find the lowest integer value for A for which $J > 30$. Although the game is boring and most likely the reader won't bother actually completing it; the ideas on HOW you might solve it are important. An analytical person might try to simplify the equations to get $J = f(A)$. The restriction on seeing only one equation at a time was an attempt to deny that approach. The easiest way is to choose a value for A then propagate the results to equations which include A as a term. This propagation is continued until J is calculated. Once J has been calculated, the constraint predicate on J may be examined. If the predicate is "false" (i.e. the constraint has been violated) then another value must be chosen for A. Alternatively, one could choose a value for J, say 30, then back that value through the network formed by the equations to discover the value of A. This is a rather poor method as there are too many degrees of freedom, and this implies an immediate

combinatorial explosion. For example, in the equation $J = H + I$, all possible values for H and I which sum to 30 would need to be considered. Many would combine both methods, moving values back and forth through the network.

Suppose the first (allowed) method was chosen, and one picked a value for A which resulted in $J < 30$. The value for J needs to be increased. Would you increase J by increasing or decreasing A (this is a trivial game problem, not a trivial question)? How did you know? Trial and error is one way to test it. Another is to examine the qualitative effects of the network of equations. That is to say, a reasoning process such as this: ". . . since J is the addition of H and I, to increase the value of J one must increase the values of H, I, or both. Since H is . . ." etc. This is probably the method the reader used to determine the effect on J of increasing or decreasing A's value. One uses the knowledge of the effects of changes in the arguments on the result of the function when one knows the function behavior. The above problem uses only addition and multiplication. It is simple. If "intelligent" systems are to arise, I would think they would need to include this sort of deductive reasoning (for the record, $J = 10 * A$; thus $A = 4$).

The reasoning algorithm in DDAFCON is designed to solve this problem in precisely the way outlined above. A value for A would be guessed. A value for J would be

found by exercising the network, then the value for J would be examined to see if it was acceptable. If not, the direction of change required would be worked back through the network till reaching A, and A would be changed accordingly.

As an illustration, the problem is coded into a form for the DDAFCON system. DDAFCON uses many small pieces of code called "agents" to solve problems. These agents, as a minimum, consist of a name, a list of input and/or output variables, and (if both input and output variables are mentioned) a list of functions which calculate the output variable values from the input variable values.¹

This approach is patterned after Minsky's "society of mind" notion (26) where many small agents cooperate to form a system. There appear to be some practical advantages to this approach.

1. Each agent is independent of the others. Thus, incremental growth is possible (and the nominal situation).

2. The exact "fit" of an agent in a system is clear and unambiguous. Its inputs and outputs are precisely defined with no side effects.

3. The grain size of each agent is easily controlled. This is more of an advantage of model-based

¹Again, in this analysis, the data items are considered nodes in a graph, while the arcs are the functions which transform the input to the output.

reasoning than DDAFCON per se. A model built to reason about a real-world system can be constructed at any level of abstraction. This supports incremental growth as well. Prototypes may be developed at a high level and gradually refined.

4. Explicitly listing inputs and outputs allows an easy static analysis of the data-flow in the system. Different algorithms for partitioning the system into pieces for parallel execution can easily be tried.

The "game problem" posed above is illustrated next. It is coded using the data-flow description method provided by DDAFCON. Using this method, a "model" of the problem is constructed. The model is "compiled" into a distributed system which solves the problem. The exact meaning of each part is discussed in detail in the next chapter, and one may wish to consult it; however, intuition should be enough to work through the code.

```
(agent-frame    fooA      ; this agent produces A with a
                        ; value of 1.
  (output (A))
  (control (A))          ; for a description of all the
  (defaults ((A 1)))    ; details of the problem descrip-
                        ; tion language, see the users
                        ; manual

(agent-frame    fooA-B    ; this agent takes the current
                        ; value of A and produces B.
  (input (A))
  (output (B))
  (functions (
    (B (times 2A))))))
```

```

(agent-frame    fooA-C      ; this agent takes the current
                    ; value of A and produces C.
  (input (A))
  (output (C))
  (functions (
    (B (times 3 A))))))

(agent-frame    fooB-DE    ; this agent takes the current
                    ; value of B and produces D and E.
  (input (B))
  (output (D E))
  (functions (
    (D (plus B 2))
    (E (difference B 2 )))))

(agent-frame    fooC-FG    ; this agent takes the current
                    ; value of C and produces F and G.
  (input (C))
  (output (F G))
  (functions (
    (F (plus C 3))
    (G (difference C 3)))))

(agent-frame    fooDE-H    ; this agent takes the current
                    ; value of D and E and produces H.
  (input (D E))
  (output (H))
  (functions (
    (F (plus D E)))))

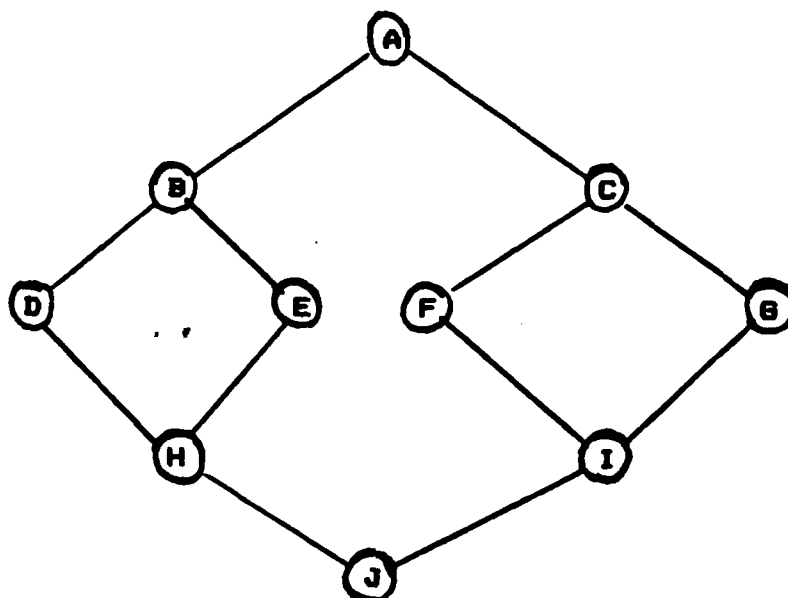
(agent-frame    fooFG-I    ; this agent takes the current
                    ; value of F and G and produces I.
  (input (F G))
  (output (I))
  (functions (
    (I (plus F G)))))

(agent-frame    fooHI-J    ; this agent takes the current
                    ; value of H and J and produces J.
  (input (H I))
  (output (J))
  (functions (
    (J (plus H I)))))

(constraints (
  ((greaterp J 30) "the object is to make
    J  $\geq$  30"))))

```

The data-flow network which this forms looks like that shown in Figure 1.



constraint: $J \geq 30$

Fig. 1. The Data-Flow Network for the Example Problem

The network has one constraint, that J must be > 30 . Initially, A is set equal to the value 1. This is propagated through the network until J is calculated (10) and then tested against the constraint ($J > 30$). Since the constraint is violated, the agent which calculates J 's values examines the function which produces the value for J and then sends messages back to its input variables to INCREASE their values since the aim is to INCREASE J 's

value and the operation performed on H and I to produce J is addition. These messages drift back through the network, each agent receiving them transforms them in the appropriate way, based on the calculation it made. Eventually a message reaches a terminal node in the graph. This node's (a data item) value is explicitly controlled by an agent rather than being the result of intermediate calculations. The agent can change the value of the variable, based on the messages received, and the new data is then propagated forward through the network once again. When all messages cease, the network is quiescent and the problem is solved. A detailed account of the solving of this problem is deferred until the algorithm for distributing the problem among a number of processors is discussed.

Parallel Processing in DDAFCON

Presently, DDAFCON has two different algorithms for partitioning and distributing the problem pieces over a number of processors. The first method uses a "strongly connected components" analysis of the full data-communication network. The second method uses an algorithm called "parallel paths" which examines only the forward data-flow graph and produces a partitioning of the graph which assigns all the nodes to a linear chain of data which is either parallel to, or independent of, all other linear

chains. The "parallel paths" algorithm is a new one developed during this thesis investigation. It seems to produce the maximum number of pieces a problem can be dissected into, and hence has a potential value beyond this thesis. In the following paragraphs, the "strong components" analysis will be addressed first, then the "parallel paths" method will be presented.

For both methods, the data are the focus of the analysis, not the code which manipulates the data items (agents). The focus on the data resulted in a slight change in "view" when examining the graph structures. Rather than the "usual" way of representing functions as nodes and data items as arcs flowing into the nodes, the data items are represented as the nodes and the arcs denote the transformation of one data item to another, i.e. the flow of data from one form to another. The "switch of focus" allowed standard graph-theoretical techniques to be used on the data items. This approach seems more natural than those techniques which attempt to find parallelism by using "task precedence graphs" (7). The "unnaturalness" of task precedence graphs is due to the nature of precedence and its determination. If one wishes to know whether one task precedes another, the data-flow must be examined; for it is only through an order relation on data-items that a task precedence has any meaning.

The "Strong Components" Method. Refer to the data-flow network sketched in Figure 1. There are obviously two parallel pieces to this problem. Common sense suggests that one could speed the calculation of this problem by about twofold if one could divide the problem in two just after the value of A is produced, then join again to calculate J. This is precisely the result of the network partitioning which DDAFCON produces when using "strongly-connected components."

"Strongly connected components" is a well known graph algorithm (1; 9; 14) which examines a graph and partitions it into subgraphs where, in each subgraph, each node is connected by a path to every other node. DDAFCON uses a strongly-connected components analysis of the graph to discover those data-flow sections which are interrelated, and thus can't profit from parallel processing. Each strong component of the graph contains a number of data items which, initially, are assigned together. They occupy a structure called a "blackboard segment." The notion of a blackboard is well described and documented in the AI literature and won't be discussed in detail, other than where the structure described here differs from the "standard." The graph of the data-flow structure which DDAFCON analyzes is actually two graphs superimposed and preprocessed. One graph describes the forward data-flow. This is the propagation of calculated data-values through the

network. The second graph describes the message flow through the network. This graph forms links between each pair of nodes which had data flow between them in the data-flow graph. More formally it can be stated that an arc exists in the message-flow graph from node alpha to beta if and only if an arc exists in the data-flow graph from node beta to alpha. Thus the message-flow graph is a symmetrical closure on the data-flow graph. One critical point must be borne in mind: the types of objects which flow in the two directions are completely different from one another. Data-objects are objects which flow in a forward direction and are arguments to functions and the result of functions. The message objects are requests which move in the reverse direction through the graph to request changes in the value of variables due solely to constraint violations. Messages represent a kind of feedback in the system.

It is because of this complication that some minor preprocessing of the graph is performed before the graph is examined for the strong components. All nonseparable components are first identified [see (4), Chapter 3]. Two specific conditions serve to identify separable components: if the data item originates a chain (i.e. has indegree = 0) or if a data item terminates a chain (i.e. has outdegree = 0). This permits a more natural dissection of the graph

since these nodes are the articulation points of the graph (14; 42).

Consider again the data-flow graph for the game found in Figure 1. One can consider it, keeping in mind that the reverse arcs have a different character than the forward arcs, as an undirected graph. Since node A originates the data-flow, all arcs from A are cut. Since node J terminates the data-flow (or, equivalently, starts the message flow) all arcs from J are cut. These two points formed a separation pair on the graph (42). A strong-components analysis shows that this graph contains two components. They are BDEH and CFGI. Nodes A and J are placed back into the most appropriate component (in this example, either one may be chosen). Thus the data-flow graph has been partitioned into two pieces each of which may go to a separate processor. All data items in the same component are put onto the same blackboard segment. Each blackboard segment is placed into a packet (initially empty). Each agent is then tested to see which packet contains the majority of the data items it manipulates or produces. It is placed in this packet. Ties are load-balanced. The result of all of this work is a set of packets, each of which is handed to a separate processor. Note that the form of the distributed data-base determines the placement of the functions. For this example, an optimal schedule is possible given the results, and the

maximum parallelism inherent in the problem is exploited (see Figure 2).

The results from running this problem on the Symbolics 3670 was a run time for the parallel system which took 58 percent of the time for the same algorithm processed in a linear fashion. This speedup shows the value in distributing a problem if it is possible to determine the pieces which can be processed in parallel.

Although apparently producing good results on various graphs, the strong components method fails on tree structured data-networks. This failure caused the author to search for an algorithm that could operate on tree structures and produce a partitioning that would exploit the maximum amount of parallelism found in the tree.

Consider a balanced binary tree. At every increment in depth, there are twice as many pieces which may be running in parallel. The strong components method, in this case, would identify only two components in a balanced tree, regardless of the depth. The root would (properly) be identified as the articulation point in the graph. Once the root's arcs were cut, each node in each of the sub-trees would be on a path with every other node in the same sub-tree. This was considered unacceptable. It was reasoned that an algorithm which could find all the parallel pieces in a tree structure might be useful in other graphs

	data	agents
Processor 1	G I F C A	fooA fooA-C fooC-FG fooFG-I
Processor 2	E H D B J	fooA-B fooB-DE fooDE-H fooHI-J

a) the partitioning produced

Processor 1	FooA	FooA-C	FooC-FG	FooFG-I	
Processor 2		FooA-B	FooB-DE	fooDE-H	fooHI-J

b) an optimal scheduling for the problem

Fig. 2. "Strongly Connected Components" Network Partition and Schedule

Note: The parsing shows that the maximum inherent parallelism was exploited and an optimal schedule is possible using the "Strongly-connected Components" graphical analysis.

as well. Since no suitable algorithm was found in the literature, one was devised.

The "Parallel Paths" Method. The "parallel paths" algorithm produces a different partitioning of the problem. The graph which is produced for this analysis is slightly different from the "strong components" graph as well. This algorithm operates on a spanning-tree representation (14) of the forward data-flow graph. Thus all cycles must be removed. Cycles which would be produced by the message "feedback" are eliminated by not including the feedback arcs explicitly in the graph. This is allowable since the existence of a data-flow arc immediately implies the existence of a message arc between the same nodes but in the reverse direction. Cycles of length zero are removed during graph construction (these are variables which appear as both input and output on the same agent). Other cycles are eliminated during the search for parallel paths. The exact method will be clear once the algorithm is presented.

The parallel paths algorithm uses a labeling scheme while performing a depth-first search of the graph. A node is chosen as a start point and arbitrarily called the source node for parallel path one. From this node a depth-first search is performed labeling each node found in the search with the path number (in this case "one"). During

the depth-first search, only one arc is taken out of any node, even if more are present. Each of the other arcs leading to the other nodes in the correspondence¹ is labeled as a "potential source node." They may be used as the source node for other parallel paths. This makes sense as all nodes which appear in another node's correspondence set are in parallel with each other. Once the end of the chain is found for the current path, a "potential source" node is found and a new path is started. If a "potential source" isn't found, any "new" node will do. This process is repeated until all "new" and "potential sources" are exhausted. At this point, all the nodes in the graph have been labeled with a path number. The nodes are then sorted into groups of like path numbers, and each collection with the same path number is defined as a black-board segment.

The parallel-paths method does not produce one unique labeling of the nodes. There are many equivalent partitions possible. This is due to the number of potential paths which can be taken from any node with an out-degree greater than one, as well as the choice of the initial source node. It seems the one constant among all the equivalent partitions, though, is the cardinality of

¹The correspondence set of a node are all those nodes which can be reached in a path length of one; thus they are "immediate neighbors."

the parallel paths set. In other words, the number of blackboard segments produced is the same. The cardinality of the parallel paths set is related to the maximum cardinality of the cohort sets.¹ Thus the number of processors which can potentially be assigned is the same.

A depth-first search of a graph with a cycle can lead to an infinite loop; therefore, if a node is found which already has a path number assigned, and the path number is the same as the current path number, a cycle has been found and the search of the current parallel path is halted.

The Parallel Paths algorithm is presented next (Figure 3). Following that, the result of the algorithm operating on the game example data-flow graph is shown.

Using the parallel-paths algorithm, the data-flow graph was partitioned into the pieces (ACF) (BD) (EH) and (GIJ). The schedule found in Figure 4 shows that four processors are not necessary for this problem; two would suffice. The discrepancy between the results of the graphical analysis and the number of processors used can be found in the discordance between the data-flow graph and the way the code is distributed among the agents. The branching from B to D and E is not done in parallel by two agents, but is done by one (as is the branching C to F and G).

¹A cohort set is the set of all nodes in a graph which are at the same level.

PARALLEL PATHS:

- 1) Mark all nodes' status as "new"
- 2) Initialize current path number to zero
- 3) Pick a node and call it "current source"
- 4) Call search-chain with parameters ("current source") and ("current path number" + 1)
- 5) If able, find a node whose status is "potential source," call it "current source"
- 6) If a failure on 5, then find a node whose status is "new" and call it "current source"
- 7) If either 5 or 6 were successful, go to 4
- 8) Finished, all nodes in the graph are labeled

SEARCH CHAIN: (input parameters: current-node, path number)

- 1) Mark current-node's status as "old"
- 2) Mark current-node's path number
- 3) Gather current-node's correspondence
If the correspondence set is null, this path is finished
- 4) For all the nodes in the correspondence do the following:
 - 4a) If the status is "new," mark it "potential source"
 - 4b) If the status is not "new" check the path number on the node against the current path number. If they are the same, we've found a cycle and this path is finished.
- 5) If the path is not finished, call Search Chain with a node in the correspondence and the current path number

Fig. 3. The Parallel-Paths Algorithm

	data	agents
Processor 1	A C F	fooA fooA-C fooC-FG
Processor 2	B D	fooA-B fooB-DE
Processor 3	E H	fooDE-H
Processor 4	G I J	fooFG-I fooHI-J

a) the partitioning produced

Processor 1	fooA	fooA-C	fooC-FG	
Processor 2		fooA-B	fooB-DE	
Processor 3			fooDE-H	
Processor 4			fooFG-I	fooHI-J

b) a scheduling for the problem

Fig. 4. "Parallel-Paths" Network Partition and Schedule

Note: The partitioning shows that the maximum inherent parallelism was exploited and the schedule produced using the "Parallel Paths" graphical analysis.

If D and E (F and G) were produced by separate agents, then the "width" of parallel operations would be four, and four processors could be used at that point of the processing. Figure 5 shows the results of altering the code to exploit the results found in the parallel-paths analysis.

Processor 1	fooA	fooA-C	fooC-F		
Processor 2		fooA-B	fooB-D		
Processor 3			fooB-E	fooDE-H	
Processor 4			fooC-G	fooFG-I	fooHI-J

Fig. 5. Schedule Produced on
Altered Set of Agents

Note: A schedule produced for the "game" using the parallel-paths method on an altered set of agents (see text).

A strong warning needs to be added at this time. The DDAFCON system, as discussed earlier, uses a search scheme which has a general applicability to most problems. This in no way implies that the DDAFCON algorithm is an optimal approach for a specific problem. It's probably suboptimal for every problem (a tough one to prove but certainly safe). For the game described, the simplest method to solve it would be to iterate the network over a monotonically increasing value for A and exit the loop when $J > 30$. For a parallel implementation, a separate processor could calculate the algorithm for different values of A, thus the answer could be found in time proportional

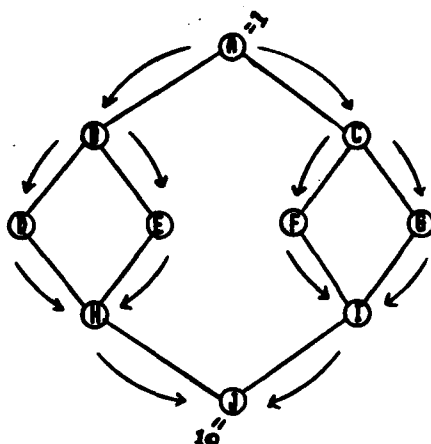
to a single running plus the overhead to hand the next value of A to a processor. Of course the example problem carries a single constraint and it is thus easy to see a "simple" algorithm which is tuned to the problem. These "tuned" algorithms lack the generality of DDAFCON. Nor are they easier to write. Neither do they approach the task in a manner with which a human can empathize. DDAFCON's manner of problem approach and its inclusion of a simple explanation capability make it eminently understandable.

Execution Control

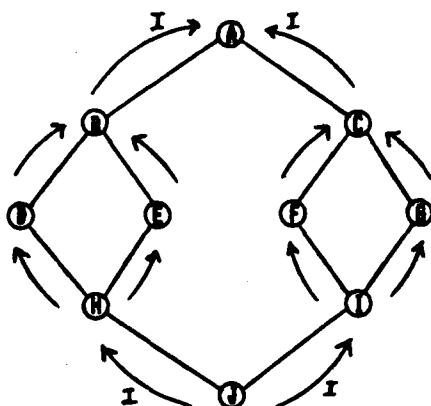
That DDAFCON partitions the data-flow network of the example into a form which can support an optimal schedule has been shown. Next to be described is the control scheme used to schedule the agents in each packet. Firstly, each packet runs on a separate processor. Not surprisingly, any agent on a packet can run concurrently with any other agent located on a different packet since, nominally, they are independent. Scheduling within a packet is also quite simple. Each packet has a simple controller which gives each agent a chance to run in turn. As in a data-flow computer (or a petri net), an agent can only run if its inputs are satisfied. Adopting this control convention results in an optimal schedule being realized for the problem presented. The difference between

DDAFCON's control strategy and a petri-net model (29) is subtle but profound. A petri-net node fires if all its input side is satisfied. For DDAFCON, its agents fire if no input values are missing (old or new) and any new data appears on the input side. This is justified since, with every change on the input side, the output values are changed. On a macro scale, data-flow appears to be a forward flow down a "data-channel"; followed by a reverse flow back through a "message-channel"; followed by a change in the value of an originating-variable; followed by a forward-flow (see Figure 6). These cycles continue until the network is quiescent; that is, there are no more messages moving, and consequently no new values to calculate.

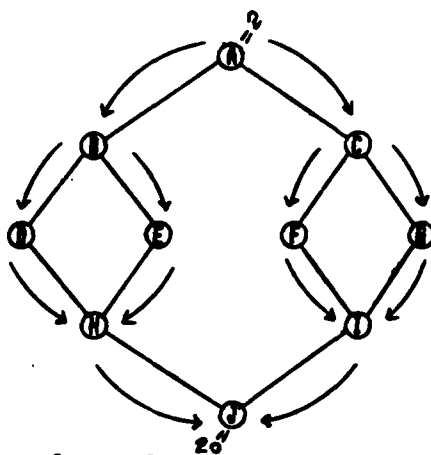
A detailed demonstration of this "game" follows, the points to notice are the forward propagation of data, the constraint violations, and backward propagation of messages, the value changes in A, and the forward propagation of A's new value. Table 1 shows a speedup of 1.9x. the actual measured speedup was 1.6x. The difference was the overhead due to the data collection, system management, and the inequality of the message-flow and data-flow sides (in Table 1 they are depicted as equal since they are both 0(1)).



a) initial value propagated through the network



b) constraint violation results in messages back



c) value altered, new value propagated

Fig. 6. A Macro View of Data-Flow Through the "Game" Problem

TABLE 1
EXECUTION OF THE EXAMPLE "GAME"
(70 Linear Steps Taking 36 Parallel Steps)

Variables (values and messages)										
	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	0	6	0	4	6	10
				(inc)	(inc)	(inc)	(inc)	(inc)	(inc)	(inc)
10	(inc) 2	(inc)	(inc)							
		4	6	6	2	9	3	8	12	20
				(inc)	(inc)	(inc)	(inc)	(inc)	(inc)	(inc)
20	(inc) 3	(inc)	(inc)							
		6	9	8	4	12	6	12	18	30
				(inc)	(inc)	(inc)	(inc)	(inc)	(inc)	(inc)
30	(inc) 4	(inc)	(inc)							
		8	12	10	6	15	9	16	24	40
36										
	A	B	C	D	E	F	G	H	I	J

A Complexity Analysis

DDAFCON implements many of the ideas outlined in appendix A. Given a data-flow network definition and constraints on variables, DDAFCON searches for a set of variable values which are "properly constrained" (see Appendix A for a definition). To find this set, DDAFCON performs a modified depth-first search (DFS). This, correctly, implies backtracking. Backtracking in DDAFCON may be thought of as dependency-directed and controlled through the constraints found on the agents which define a given system. This section introduces the basic unit of search called a "dependency-set" and shows how the complexity of the DDAFCON system is related to the structure of the dependency-sets.

The basic unit of search in DDAFCON is an object called the "dependency-set." A dependency-set includes exactly one stated constraint on a variable or variables which form the constrained variable set (CVS); all intermediate variables which were on the data-flow path to the CVS variables (these form the intermediate variable set (IVS)); and the controlled variables found at the beginning of the data-flow paths which form the controlled variable set. Although dependency-sets cover the environment, they do not partition the environment. Many dependency-sets may have elements in common. The example presented earlier in this chapter represents a single dependency-set; and

is the smallest unit upon which DDAFCON may be demonstrated.

To review, the example shown previously showed the forward movement of data through the data-flow network, a constraint test, a backward movement of messages through all the intermediate nodes to the controlled node where a new value was generated and propagated forward again. This is the basic (generate and test (43)) search on a dependency-set which DDAFCON implements. If any of the elements of the dependency-set were removed, search couldn't proceed. If there were no controlled variables, no new values could be pushed through the system.

Depth-first search (DFS) consists of three parts: moving down a path, backtracking along a path, and choosing a new path. The DFS in DDAFCON is modified from a blind search in a number of ways. First, "moving down paths" is associated with a forward movement of data along the data-flow network; and this operation happens in parallel along all the dependency-sets. Secondly, although backtracking occurs when a constraint is broken, the precise point in the search tree to which backtracking moves depends on the dynamic interaction of all the dependency sets. Backtracking may occur on many dependency-sets simultaneously. This results in an almost unanalyzably complex movement through the search tree. The example was simple and easy to follow; things are not so clear-cut with a more complex structure.

The complexity of DDAFCON can be found by examining the dependency-sets, and their interaction, in the problem environment. For each dependency-set, its complexity is dependent on the rate at which the controlled variables converge to acceptable values (become properly constrained). This speed depends solely on the algorithm used in choosing a new value for a controlled variable. For most controlled variables, a binary-search type algorithm may be used to choose the next value. This results in a complexity for the dependency-set which is

$$[\log x] (y + z) \quad (6)$$

where x is a function of the difference between the index of an acceptable answer in a variable's value list and the index of the starting value; and the size of the acceptable-answer set with respect to the value list size. This, of course, assumes a monotonic ordering of some sort on the values (integers are self-indexed, for example, by size). Note that the magnitude of this term may be reduced by using a good starting value, implying that good guesses are useful. On the other hand, bad guesses can hurt.

The y is the length of the dependency-set's path from controlled variable to constrained variable. The z term is the summation of all the complexities of all the data transformations along the data path. If the dependency-set contains more than one controlled variable, the x would

be the maximum of all x's, and y would be the maximum of all y's, and z would be the maximum of all z's.

If all dependency-sets in a problem are independent, the complexity of the system would be the maximum of all the dependency sets' complexities (since they would all be running parallel). This is the simplest condition to analyze; however, the real situation may have contained and overlapping dependency sets which are determined by the problem structure itself. These two categories, contained and overlapping, form the other class of dependency-sets whose complexity needs to be analyzed.

Dependency-sets which overlap can vary in overlap from a single overlapping controlled variable to a completely contained dependency-set (see Figure 7). For overlap conditions, the point of dependency-set interaction is the critical point for analysis. The point of interaction is the set of controlled variables which receive messages from all the constraints which have the controlled variables in common. For compatible constraints, there is no problem (compatible constraints are those which are properly constrained for some values of the controlled variables they have in common); however, if messages are received by a controlled variable from at least two different constraints, and each constraint requests a value change in a different direction, then that controlled variable's value can not be changed (a value can not grow simultaneously larger and

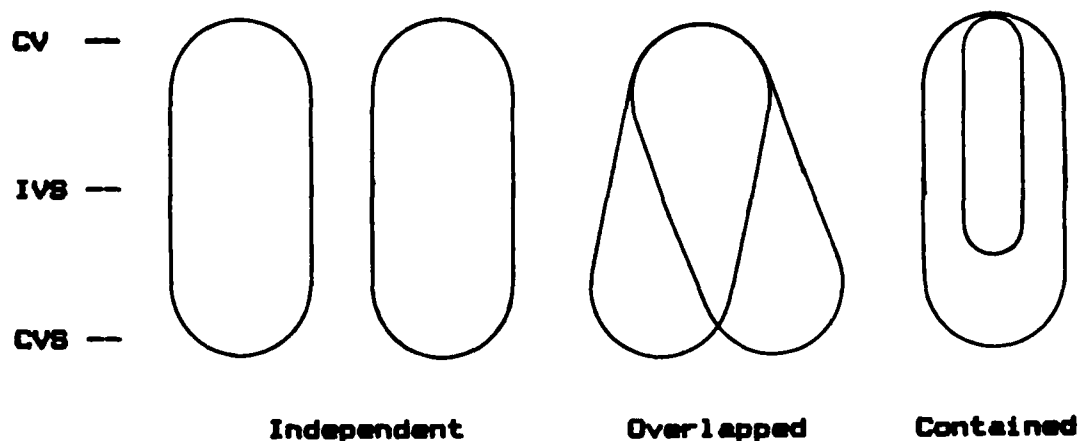


Fig. 7. Diagrammatic Representation of
Dependency-Set Interrelations

smaller). The controlled variable, in effect, becomes "fixed" at its present value for the current cycle of the network. If, under all conditions, no value can be found for a controlled variable which satisfies all the constraints on that variable, it may be said that the constraints are incompatible. This condition can result in DDAFCON never finding an answer. This is a direct analogue of the "halting problem" studied in Turing machines (25). Put another way: if (at least) two dependency-sets share

a controlled variable, it is possible that a valid condition for one may be an invalid condition for the second. And that the attempt to find a valid condition for the second may result in an invalid condition for the first. This is clearly an unstable condition. It is the equivalent of a depth-first search on an infinitely deep search tree.

Under certain conditions, it may be possible to determine whether a problem is unsolvable. If, for any dependency-set in a problem, all the controlled variables are fixed and there is still a constraint violation (i.e. not properly constrained), then the problem is "unsolvable" (25). Currently, dependency-sets are not explicitly mapped and tracked in DDAFCON. This would need to be implemented.

The overall complexity of a DDAFCON problem ($O(D)$), given compatible constraint-sets, is a function of the amount of overlap among constraint-sets and the complexity of each constraint-set. Let q be any controlled variable and $C(q)$ be the set of all constraint-sets of which q is a member. The complexity of $C(q)$, written as $O(C(q))$, which is the complexity of a composite of constraints due to an overlap through q , is proportional to the product of the complexities of each of the constraint-sets found in $C(q)$. Thus, for every controlled variable q

$$O(D) = \max O(C(q)) \quad (7)$$

The complexity is controlled by the problem structure. With all dependency-sets independent, the network itself is polynomial in complexity. As overlap increases, the permutations of the interactions on the network can become combinatorially complex. Since the system modeled many of the ideas shown in Appendix A, and that was shown to be NP-Complete; this result is not surprising. The following example illustrates the affect of overlapping constraint-sets. Consider a simple network formed by the equation $z = x/y$. The network looks like this (Figure 8):

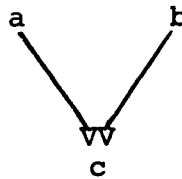


Fig. 8. Data-Flow Network for $c = a/b$

If the single constraint $c \geq 5$ is added, the result is a single constraint-set (Figure 9).

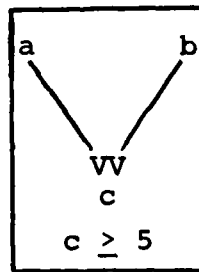


Fig. 9. Constraint-Set for $c = a/b$

If the initial values for a and y are both one, this problem would be solved in three cycles. Table 2 shows the

TABLE 2

DATA-FLOW AND MESSAGE-FLOW FOR SINGLE
CONSTRAINT-SET ON $c = a/b$

Values			Messages	
a	b	c	a	b
1	1	1	inc	dec
2	1/2	4	inc	dec
4	1/4	16	-	-
Network Quiescent				

Note: This table shows the data-flow and message flow of the single constraint-set presented in the text.

results. If a second constraint, $b \geq 1$, is added there would be two constraint-sets present which would overlap at b. This situation is shown in Figure 10.

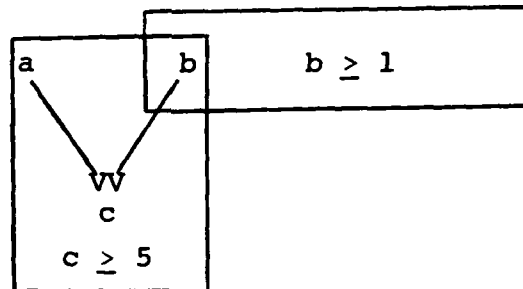


Fig. 10. Overlapping Constraint-Sets

With two overlapping constraint-sets, the number of network cycles to converge onto a properly constrained solution doubles given the same network and the same initial conditions. This is shown in Table 3. One can conclude that specific instances of certain problem-solutions rendered in this system can be of polynomial complexity

TABLE 3
DATA-FLOW AND MESSAGE-FLOW FOR OVERLAPPING
CONSTRAINT-SETS ON $c = a/b$

Values			Messages	
a	b	c	a	b
1	1	1	inc	dec
2	1/2	4	inc	(inc dec)
4	1/2	8	-	inc
4	1	4	inc	dec
8	1/2	16	-	inc
8	1	8	-	-
Network Quiescent				

Note: This table shows the data-flow and message-flow of the overlapping constraint-sets presented in the text.

provided that constraint-sets don't overlap; however, the general case must be considered to be of exponential complexity. A clear affirmation of a well-known assertion of computational theory. To wit, it is conjectured that no NP-Complete problem, when stated for the general case, has an algorithm which will run in polynomial time (25).

Other undesirable effects from the interactions of overlapping dependency-sets may occur as well. The relative lengths of the paths of overlapping dependency-sets may cause two messages to be received at a controlled variable when only one might suffice. Suppose there exists two dependency-sets, A and B (call the constraints defined

for each A_c and B_c respectively), each of which had a controlled variable x in common. Suppose further that the path in dependency-set A which leads from x to A_c is significantly longer than the path in dependency-set B which leads from x to B_c . This implies that messages, originating in A_c and B_c at the same time, could reach the agent which controlled x at different times. If the value in x was initially y_1 , and both A_c and B_c would be satisfied if x was y_2 , (the value of x takes the values $y_1, y_2, y_3 \dots$ as messages of type m are received), x may take on the value y_3 if two messages were received rather than one.

A common, and difficult, problem for AI systems is explaining their behavior in a manner understandable to a user. Explanation capabilities are usually found in rule-based systems and normally consist of a recital of the rules which led to a conclusion. Since DDAFCON operates on a model than a rule base, a different approach was developed. Since actions are only taken when constraints are broken, and the actions taken consist of altering values of controlled variables in the dependency-set of the broken constraint, DDAFCON provides explanations of why the variable's value is being changed, and how it will be changed. This is useful information as many dependency-sets have controlled variables not envisioned by the user. (This is due to the complex interactions of the dependency-sets. Remember, dependency-sets are a unit of action, not

construction. The notion of dependency-sets was developed to help analyze the system behavior.)

This approach subsumes both dependency-directed backtracking and spreading activation. Both notions are incorporated into the basic primitives of the DDAFCON structure. Dependency-directed backtracking allows backtracking only along paths which are directly involved in the condition which gave rise to the backtrack need. This is realized through the use of certain message types which travel in the reverse direction over data-paths (see Chapter II). Spreading activation is a physiological metaphor born from observations of neural activity. As a neuron is excited, its excitement is spread throughout all the neurons which have connections to it (excitatory connections) in a characteristic way, thus resulting in the notion of a "spreading activation." The system presented uses the flow of data to "spread" its "activation" to all agents along the data-flow path. The notion of dependency-sets encompasses both of these ideas.

Why Not Rules?

In the development of any system, decisions and compromises must be made. The system presented, DDAFCON, is no exception. The common perception of AI systems is dominated by the rule-based system, and since DDAFCON does not use rules as its primary method of inference and

control, the question that is most often raised is: why not? This section will address some of the shortcomings of rule-based systems, not from the perspective of reasoning and knowledge usage, as these have been discussed elsewhere (10; 11); but from the perspective of software engineering principles and building a system whose purpose is to attempt parallel processing of a problem solution.

Rules have two parts: a left-hand side (LHS) which contains a predicate which is evaluated, and a right-hand side (RHS) which are the actions taken if the predicate evaluates to "true." Superficially, DDAFCON's agents appear to be rule-like. They have a "left-hand side" which is composed of an input-vector predicate (invisible to the agent, the agent is scheduled when the input vector is satisfied), and a "right-hand side" which is the running of the agent's code. This similarity ends at this high level.

Rules are fairly unstructured affairs, that which is evaluated on the LHS is not necessarily related in any way to the actions on the RHS. As well, the actions performed on the RHS are not limited in any way. Although the idea, in a "pure" system (43) is to add an inference to the data-base through an inference made in the LHS, any side effect desired is allowable. Usually "pure" rule-based systems are anything but pure. The result is unclear control flow and indeterminate coupling between rules. If a

list is not maintained of each step (rule firing), it is difficult to determine how one arrived in the current state. This potential anarchy, among other things, renders the code 1) difficult if not impossible, to validate; and 2) makes it difficult to partition for parallel processing.

To eliminate these problems, DDAFCON's agents are highly constrained. All external variables used by the agent must be explicitly declared as "INPUT." Without this declaration, the variables don't exist. The variable values used are kept locally, all variables are "pass by value." No side effects are possible since each agent is constrained in its own environment, and the only output an agent can perform is limited to those it specifically declared as "OUTPUT," and messages which can be sent to immediate predecessors on the data network. Agents function solely to transform input values to output values.

The clear and unambiguous data network derived from the interaction of DDAFCON agents makes constructing and analyzing a distributed system surprisingly easy. The lack of side effects makes the incremental development and debugging less painful.

Summary

Some of the major points discussed in this chapter are listed below:

1. A model-based approach allows reasoning about a system in an orderly way using a simulation which models the actual behavior and constraints of the system modeled.

2. Since the upper limit to performance is imposed by the number of processors which can be assigned to a problem, a need exists for an algorithm which can find the maximum number of parallel pieces in the problem. This is supplied by "parallel-paths."

3. Constraint satisfaction/propagation is used for "reasoning"; this is a common approach to reasoning with models.

4. DDAFCON's topology consists of small, distributed agents, each of which is responsible for a specific (quantitative or qualitative) calculation. Each agent specifies the data which it needs as input, and the data it supplies as output. All the data items required by the agents are located on a "virtual blackboard" (see Chapter IV). The intercommunication network implied by the set of all agents is partitioned and distributed among a number of processors which run simultaneously.

5. The partitioning of the network creates sets of data-items and assigns each set (called a packet) to a processor. Agents are then assigned to packets in a manner which maximizes the locality of reference. The co-location of agents usually implies a partial order relation on their calculations, thus each member of a set will not directly

benefit from parallel processing with other members of the same set. However, agents assigned to separate sets and placed on separate processors are probably independent from one another (no partial order relation) and thus benefit from the separate processors. Partitioning the network frees the programmer from explicitly examining the intercommunication network and "tuning" the system. Hudlicka and Lesser (21) report the tuning as a major effort in designing and altering their distributed systems. Splitting all major calculations into separate agents, then partitioning the network, is an attempt to exploit the maximum degree of parallelism inherent in the problem.

6. The search technique used rests on a notion of "dependency-sets," a structure which embodies the techniques of dependency-directed backtracking and spreading activation. It can be shown that the complexity of DDAFCON is exponential in time (in the limit); but that some problems can be solved in polynomial time if their structure allows.

IV. A DDAFCON Primer

This section further develops DDAFCON. It is divided into two parts which discuss separate (though inter-related) aspects: the data structure, and the Data-Flow Description Method (DFDM) declaration structures. The declaration language (DFDM) can be used without any other system specific knowledge; but probably not as well as if the overall structure is understood.

Data Structure

The data structure for DDAFCON will be presented in this section. A basic knowledge of Lisp data structures is assumed.

DDAFCON can be understood, in its simplest terms, as the interaction of two data structures: agents and a blackboard. The agents are named structures which hold a collection of data, procedures, functions, and constraints which are logically related to one another in some fashion. In one sense it may be thought of as a "procedure" in an ALGOL/Pascal/Ada sense in that all the data and procedures are locally defined with a narrow i/o channel that is strictly defined and enforced. In another sense it may be thought of as a "record structure," again in an ALGOL/Pascal/Ada sense because all the information contained in

the agent is available by accessing the structure using field names. In fact, the agent is a repository for all the procedures and local data used to calculate certain data-items defined by the agent and which are made available to other agents. Thus, when an agent is "scheduled to run" the run-time system obtains the appropriate code from the agent and runs it. Any references to variables used in the agent's code refer strictly to local variables defined and held by the agent.

Agent Structure. Specifically, an AGENT consists of a name (an atom) and a list of properties attached to the atom which include one or more of the following:

<u>Property Name</u>	<u>Items Kept by Under the Property</u>
INPUT	a list of input variables;
OUTPUT	a list of output variables;
DEFAULTS	a list of default values for input and/or output variables if any are undefined;
CONTROL	a list of variables "controlled" by this agent; (the notion of "controlled variables" will be dealt with later.)
CONSTANTS	a list of constant values;
SAVED-VALUES	a list of saved values; (those values which remain instantiated at all times)
FUNCTIONS	a list of functions; (a method for transforming input variable values to output variable values)

CONSTRAINTS	a list of constraints;
AUX-FUNCTIONS	a list of any auxillary functions needed;
CHANGE-FUNCS	a list of functions which are used to change the values of controlled variables in answer to request messages;
QUAL-FUNCS	a list of qualitative relationships between arguments to a function and the resultant
(local variable)	the value of a local variable kept in a blackboard-item structure (see blackboards). There are as many of these items as there are local variables. Each property name for a local variable is the name of the local variable.
LAST-SEQUENCE-NUMBERS	a list of the last sequence numbers for the input data (see blackboard-item under the blackboards section)

Blackboard Structure. The blackboard consists of a number of blackboard segments each of which contains one or more data items which were aggregated during the data-flow network partitioning. Each blackboard segment represents either a strongly connected component of the data-flow network, or a partial processing chain as found using the parallel paths method. One or more blackboard segments is included in a packet (remember the packet is the unit of assignment to a processor).

Each blackboard is an atom whose name is the name of the blackboard; and each variable which the blackboard

contains is a property of the atom accessed through the variable's name. For easier reference, the value bound to a blackboard is a list of all the variables contained in the blackboard. This functions as an index to the blackboard. Figure 11 shows the structure of a blackboard segment.

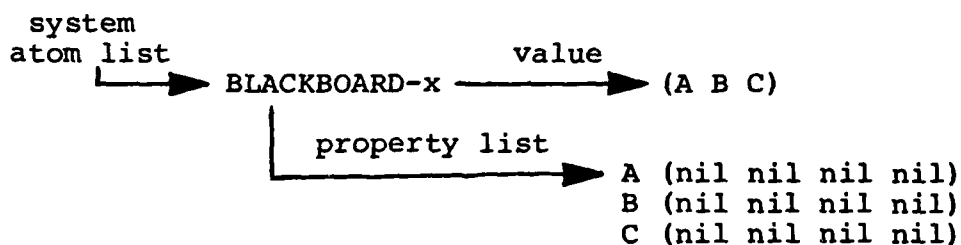


Fig. 11. Blackboard Segment Structure

Note: A blackboard segment called BLACKBOARD-x, showing the index and the variables contained (initialized condition).

Each variable on a blackboard is represented in a blackboard-item structure. This is a list with four elements: the current value of the variable, the sequence number for the current value, any current messages attached to the variable, and a slot for process flags.

Current Value. The current value of a variable is the quantity usually associated with the bound value of an atom in the Lisp world, or the contents of an address in the ALGOL/Pascal/FORTRAN/BASIC... world. The blackboard-items are untyped, thus the value may be anything, nothing is assumed, and consequently nothing is

enforced. It is possible to include typing and type checking in the code which is written using DDAFCON, but this is external to the system itself.

Sequence Number. At initialization, all variables have their sequence numbers initialized to zero. As new values are written as output to the blackboards, the sequence numbers are incremented. When an agent uses an input datum, it stores the sequence number for that datum. This allows each agent to keep track of new data as it appears, and consequently allows the scheduling of agents to run.

Current Messages. When an agent discovers that a constraint has been violated, it posts a message to the concerned input data items to request appropriate value changes which would relieve the constraint violation. Each message consists of an action requested (INCREASE, DECREASE) and a reason for the action. The reason is a string which is supplied by the coder (see DFDM "constraints").

Process Flags. Process flags are flags which control certain processing options. Currently there are two process flags, SHOW and FIXED. The flag SHOW, if attached to a variable, is used to signal that the variable's value is to be displayed to the user. The FIXED flag attached to a variable "fixes" the value of a variable so that any messages to change the value are ignored.

Note that the system's variables do not exist in the normal sense. For example, if there is a variable call "bar," which is contained by blackboard "blackboard-x" and used by agent "foo," there is no atom named "bar" which has a value bound to it. Rather, "bar" is the name of a property which both the blackboard and the agent share. This technique enforces local scoping of variables even though many agents may reference the "same" variable. For the agents, when they are scheduled to run, copies of the variables found on the appropriate blackboards are made for the agent and ALL references to these variables use the local copies. Only the "output" variables for an agent (those declared as output), at the termination of the agent's processing, are written out to the appropriate blackboards.

Messages are also posted to a data-item (see "messages" description above). This way no agent needs to know the identity of any other agents; nor which agents handle (produce or use) the variables which they have as input or produce as output. As well, this structure has messages appended only to variables which are known to the agent. This locality of reference and scope for both messages and data eliminates two major problems with the unlimited message passing found in the object-oriented approach: that of global access to all agents (actors, flavors, etc.) and the consequent tangled web of message

channels; and the need for each agent to have knowledge of both 1) who to send a message to, and 2) message to send. Using a locality of reference and scope scheme as found in DDAFCON, an agent needs to know only of its immediate environment. Message channels are limited to those explicitly defined through the input and output declarations of each agent.

Also note that both reading and writing to the blackboard segments is performed with accessor macros and "specific location code" which is written at the time of network compilation and data partitioning. The "specific location code" is placed on a disembodied property list and is specific by data-item (variable) location by blackboard and packet. Each packet's "specific location code" is appropriately different to reflect the local environment. If variable "A" is located on blackboard "B-1" which is on processor 2, and access requests are made for "A" from an agent on processor 2 and an agent on processor 1, the code executed to get the data will be different. One is a local access, the other is a network access. The code written for the two agents, though, is exactly the same. The agents have no knowledge where the data is kept. This makes writing code fairly easy as all the details are kept behind the scenes.

Packet Structure. The packet is the basic unit of processor assignment. Each packet, nominally, runs on a separate processor to achieve the potential parallelism. A packet exists as an atom with two properties, AGENTS and BLACKBOARDS. The AGENTS property contains a list of all agents assigned to the packet, the the BLACKBOARDS property contains a list of all the blackboard segments assigned (see Figure 12).

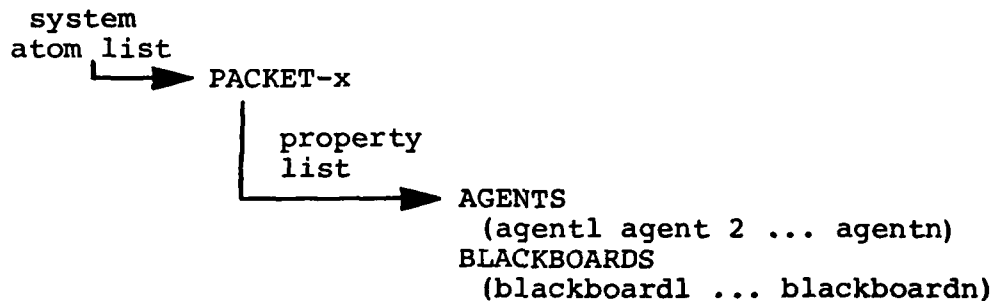


Fig. 12. Packet Structure

Note: A packet called PACKET-x showing its property list. The property list contains the names of the agents and blackboards which are to be assigned together to a processor.

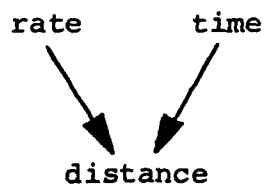
Agent-List. This is simply a list of agents, supplied by the user of all agents which should be included for structuring a DDAFCON environment.

Data-Flow Network Structure. The data-flow network is a graph assembled from the INPUT and OUTPUT declarations found in the agents listed in the agent-list. For the data-flow (as opposed to message flow; see Chapter IV)

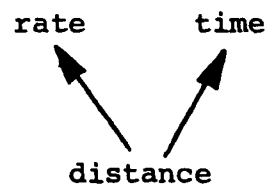
graph, each input element is taken as a node in the graph and the set of output elements is taken as the correspondence (connected by an arc path of length one) of the node. For the message flow graph, each output element is used as a node and the set of input elements is taken as its correspondence. For example, if an agent calculated "distance" given "rate" and "time" (the familiar $D=RT$), the declarations for the input and output would look like that shown in Figure 13, and the graph segment would appear as in Figure 14.

```
(INPUT (rate time))
(OUTPUT (distance))
```

Fig. 13. DFDM Declaration for
Distance = Rate * Time



a) data-flow



b) message-flow

```
(rate distance))
(time distance))
(distance (rate time))
```

c) internal representation of the graph

Fig. 14. Graph Segment Produced for Figure 13

The complete graph is a composite of all the agents appearing in the agent list. An important note: nodes found in different agents are considered identical if, and only if, they are spelled the same.

Data-Flow Description Method (DFDM)

The DDAFCON DFDL consists of a number of key-words followed by various fields. Each key-word is processed in a specific way in order to provide the necessary data and code for the agent. Much of the code is altered during the "compiling" of the agents. Since no error messages have been included, an understanding of the process will help a user track down any bugs which may appear in their agent-code.

An example of an agent-frame declaration will be shown. This example is used to demonstrate the DFDM. The example includes all the implemented key-words.

```
(agent-frame    an-example

  (INPUT ( input-item-1 input-item-2 ... input-item-n ))
  (OUTPUT ( output-item-1 ... output-item-m ))
  (CONTROL ( output-item-c ))

  (DEFAULTS (
    ( input-item-f 1.0 )
    ( input-item-g (ask-for-item) )
    ( output-item-d d-const )))

  (CONSTANTS (
    ( d-const 1.2 )
    ( pi 3.14 )
    ( minimum-value-for-k 22.4 )))

  (SAVED-VALUES ( number-times-run ))
```

```

(FUNCTIONS (
  ( output-item-1 ( quotient input-item-1
                           input-item-2 ))

  ( output-item-m
    ( IFD* (an-aux-func (input-item-4 P)
                        (input-item-5 IP)
                        (input-item-6 I))))))

(CONSTRAINTS (
  ((LESSP input-item-1 input-item-2)
    "here state why item-1 must be less
    than item-2 ")
  ((GREATERP output-item-k minimum-value-for-k)
    "item-k must be greater than the
    minimum allowed" )))

(CHANGE-FUNCS (
  (output-item-c
    ( lambda (x)
      (progn ()
        (cond
          ((equal x 'increase)
            (SET F output-item-c
              (add1 output-item-c)))
          ((equal x 'decrease)
            (SETF output-item-c
              (sub1 output-item-c)))
          (WHY))))))

(AUX-FUNCTIONS (
  (ask-for-item ()
    (with editor-functions do
      (send 'terminal-io ':get-line)))

  (an-aux-func (arg1 arg2 arg3 )
    (foo arg1 arg2 (bar arg3))))))

```

1. AUX-FUNCTIONS

Form:

```
(AUX-FUNCTIONS {list of functions} )
```

Processing:

Each function is translated. 'DE is consed to the front of the function, and the form is evaluated. This

results in a correctly translated function available in the correct packet.

Auxillary functions are used to help process data. It is not necessary to include all functions which are called by code in the "functions" or "defaults" section. However, there are some benefits to doing so. Firstly, it provides a way to ensure that any functions required will find their way to the packet where they are needed. Secondly, any references to variables declared in the "input" or "output" sections will not be treated correctly if the function isn't defined as an auxillary function. Recall that the variables don't exist in the usual way (see data structures section), unless the user codes the variable accesses himself, the result will be an "unbound atom" error message at run time.

2. CHANGE-FUNCS

Form:

```
(CHANGE-FUNCS ({nameofitemto-change}{function-def})...)
```

Processing:

At agent definition time, these functions are taken, as received, and put as a property value onto the agent.

When used, change functions are retrieved from the agent, translated, then passed a single argument which describes the action requested (i.e., 'INCREASE or 'DECREASE is passed). It is the responsibility of the change function

to assign the new value to the variable. Since the code is translated before being evaluated, it is easy to make the correct assignments. The only "trick" is to use a SETF rather than a SETQ for the assignment. This is needed since the variable doesn't really exist as an atom, but as a property name. The SETF allows the correct code to be substituted for the variable name and executed correctly. Failure to use the SETF results in "argument not an atom or a locative" type errors.

Limited explanations are available which give the reasons why the change message was produced. These reasons are displayed by calling a function called (WHY). This capability makes a trace of the system logic easy to follow.

If no change functions are defined for an agent which has controlled variables, the system will attempt to change the value of the variable (i.e. there is a default change function). This is an exceedingly stupid function which increments or decrements the value by one. Note that system performance is a partial function of the speed at which the variables converge to acceptable values. The change functions directly influence the rate of convergence (see section on complexity).

3. CONSTANTS

Form:

(CONSTANT (({constantname}{constantvalue})...))

Processing:

Each constantname /constantvalue pair found is rendered into a DEFCONSTANT list and evaluated. Note that the value becomes global on the packet where it is located. This forces constants which have the same name to have the same value. If there is an attempt to change the value of a constant, a message will be presented the user asking whether this is desired.

4. CONSTRAINTS

Form:

(CONSTRAINTS (({constrainstatement} "reason string")...))

Processing:

When the CONSTRAINT statement is encountered, each constraint is treated in the following way. First, each constraint is negated to provide trigger conditions. Note that a constraint is a positive statement about the state of the system. Thus, for example, if in a flight planning system the weight of the aircraft should never exceed the max-gross-weight (defined as a constant perhaps), that is how the constraint is written. (i.e.: (LESSP weight max-gross-weight)). The DDAFCON writes the code which is actually executed. The second element of a constraint

statement is the reason. This is used for the system's explanation capabilities. These explanations follow the messages through the system and are provided to the user if he asks. To be useful the messages should explain, tersely, the reason for the constraint. For the above example an explanation could be included like this:

```
( (LESSP weight max-gross-weight)
  " the weight must be less than the max gross weight" )
```

In the flight-planning application appearing in another section of this thesis, this kind of information was useful when wondering why, the system was trying to remove fuel from the aircraft which was cutting into the reserve fuel on a long trip leg. When asked why, the reason was "the weight must be kept below the max-gross-weight." Since I had "FIXED" the payload (see sections "data-structures blackboard items," and "Using DDAFCON"), this was the only alternative to reduce the weight. (The system also attempted to stretch the range by power reductions in order to bring the reserves back to the FAA mandated 45 minutes.)

5. CONTROL

Form:

```
(CONTROL {list of controlled variables} )
```

Processing:

When encountered, the list is put onto the property "CONTROL" of the agent currently being processed.

When messages appear for a controlled variable, the agent determines what action is being requested, then the variable is altered. (See CHANGE-FUNCS for more information.)

6. DEFAULTS

Form:

(DEFAULTS (({variablename}{defaultvalue})...))

Processing:

When encountered during agent-frame construction the default name/value pairs are placed on the DEFAULTS property of the agent.

If a declared input item is gotten from the blackboard and is found to be un-valued (nil); a default value is sought. If no default is defined for the un-valued variable, the current agent's processing can't continue.

Defaults can be defined for the output variables as well. If, due to an undefined input variable, an output function can't be calculated, the output variable may be given a default value.

Default values may be any evaluatable form. Thus functions may be used to supply default values as well as constants. The default forms which are evaluated are also translated, thus any function which is used may also use blackboard variables if declared as input or output by the agent.

7. FUNCTIONS

Form:

(FUNCTIONS ((output-var-name1 (function1))...))

Processing:

When encountered during agent-frame construction, two properties are loaded onto the agent. The first is the FUNCTIONS property and it contains an association list indexed by output variable name. The second property is the QUAL-FUNCS property. This consists of an association list, indexed by output variable name, which describes the qualitative relationship between the output variable and the input variables. Each function is examined by the qualitative-effect parser to determine whether the input variables which are arguments to the function are qualitatively proportional, inversely proportional, or independent of the output variable (an independent finding is also used when the effect can't be determined). This information is used when moving messages through the network.

Consider an example. Suppose an agent calculated $D = A/B$ (D is an output variable. Both A and B are input variables). During agent-frame construction this function would be examined and it would be determined that A is qualitatively proportional to D while B was inversely proportional to D. If a message were received which requested that the value of D be increased, the agent would continue to pass the messages up the network altered according to

the qualitative relationships between D, A, and B. Since in this example D is to be increased, those input variables which are qualitatively proportional are sent a message to increase, while those which are inversely proportional are sent a message to decrease (along with any explanation string sent when the constraint was broken and the message originated).

Not all functions can be parsed without help. Only common mathematical expressions are successful. To allow other types of functions to be used, a special function called an "informal function description" (IDF*) may be used as a wrapper. This function has as its argument a list composed of the name of the function which should be called, followed by the arguments to the function. Each argument is expressed as a list with the argument as the first element and the qualitative relationship as the second. IFD* is fully recursive. Arguments may be atoms or functions. Suppose one defines division functions which handle integer division separately from real division and complex division (call them DIVI, DIVR, DIVC). The parser doesn't know about these functions and would assume, lacking any other knowledge, that all the arguments are qualitatively proportional to the result. This is clearly a false assumption. However, by using an IDF* wrapper, the parser is explicitly told the relationships. They would look like this:

(IFD* (DIVI (numerator P) (denominator IP)))
(IFD* (DIVR (numerator P) (denominator IP)))
(IFD* (DIVC (numerator P) (denominator IP))).

8. INPUT

Form:

(INPUT {list of input variables})

Processing:

When encountered during agent-frame construction the list of input variables is placed on the property INPUT. This list is used to fetch the needed variables for an agent's processing. It is also used to construct the data-flow network (see "Distributing tasks to processors: A data-flow approach").

9. OUTPUT

Form:

(OUTPUT {list of output variables})

Processing

Similar to INPUT.

10. SAVED-VALUES

Form:

(SAVED-VALUES {list of object names to save})

Processing:

When encountered during agent-frame construction, each object found on the list is declared special. This ensures its longevity.

Using DDAFCON

This section briefly outlines the steps which are necessary to use DDAFCON and the options available for output.

DDAFCON is written to run on a symbolics 36xx processor with assumed dynamic-scoping of variables. Although an effort was made to retain a standard Lisp environment, including extensive use of the "AFIT-Lisp" standard, some use of the Zeta-Lisp/Lisp machine environment was made, and this use compromises the portability somewhat. The non-standard features are in the user i/o. DDAFCON makes use of Z-macs editor functions for user input as well as the windows/window-panes provided on the Symbolics machine. "Flavors" was not used as it wasn't compatible with the aim for a distributed architecture. That is not to say that "flavors" can't be used for an application using DDAFCON, just that the "guts" of DDAFCON couldn't use it successfully. Changing the user i/o should allow DDAFCON to port to other machines.

Using DDAFCON consists of five steps:

1. Loading the DDAFCON code
2. Loading the agents needed
3. Constructing the environment
4. Choosing the processing/ i/o options and running the system
5. Interpreting the output

AD-A163 947

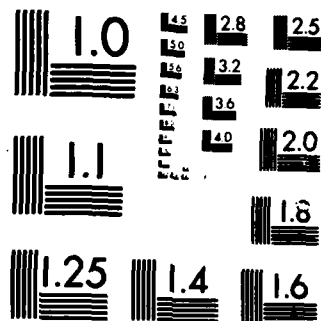
REASONING IN REAL-TIME FOR THE PILOT ASSOCIATE: AN
EXAMINATION OF A MODEL.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. D O NORMAN
DEC 85 AFIT/GCS/ENG/85D-12 F/G 6/4

2/2

UNCLASSIFIED

NL

									END				
									FINED				
									DTIC				



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Loading the DDAFCON code consists of finding it, and executing a load. The file name is DDAFCON.lisp.

Assuming that the agents have been written and stored in a file, that file should be loaded next. This step will construct the agent-frames which were defined in the file.

The environment is formed (i.e. the networks analyzed and packets built and assigned, etc.) by calling "construct-environment" and passing a list of the participating agents as the first argument and the method to use for partitioning the graph as the second argument. For example, if a system is to be defined from three agents names first-agent, second-agent, and third-agent; then the function to construct the agent interaction system is:

```
(CONSTRUCT-ENVIRONMENT '(first-agent second-agent
third-agent) '(parallel-paths))
```

or, equivalently:

```
(SETQ my-agents '(first-agent second-agent
third-agent))
```

```
(CONSTRUCT-ENVIRONMENT my-agents '(parallel-paths))
```

The second form is easier to use since the setq form may be defined in the file which holds the agent definitions ("strong-components" is the other partition method available).

When the packets are defined they will be displayed on the CRT as square areas with three sections. The top

section will have the packet name, the second section, titled "AGENTS," will have a list of the agents assigned to the packet, and the third section, titled "DATA" is reserved for any data in that packet which the user wishes to have displayed.

Data is displayed in the packet windows by telling the system to show the items desired. This is done by using a function called "SHOW." "SHOW" takes as an argument the name (or a list of names) of those variables which are to be displayed. Thus, if one wishes to see the values of A and B as processing proceeds, the following forms are evaluated:

```
(SHOW 'A) (SHOW 'B)
```

or

```
(SHOW '(A B) ).
```

Variables that are being shown may be turned off as well using "UN-SHOW."

Any variable which is to have a fixed value must be made "FIXED." The function "FIXED" does this. Use the syntax shown for "SHOW" substituting the word FIXED for SHOW.

RUN-SYSTEM with no options shows only the variables which were marked for display using the (SHOW ...) function. Other options add to this. One other option is "REASONS." This option provides reasons for alterations in controlled variables during run time. Another option

is "VALUES." This option prints a continuous stream of information on how the processing is moving. The final option is "NO-OVERHEAD." This option causes the performance statistics displayed to be calculated without figuring the overhead for the parallel processing. To run the system with all options, the function would look like this:

```
( RUN-SYSTEM '(reasons values no-overhead));
```

to run the system with the defaults:

```
( RUN-SYSTEM nil ).
```

Periodically during the run, and when the network is quiescent (no messages moving through the system, and all new values carried to terminal nodes in the network), the system displays some run statistics. The statistics include the number of blackboard accesses to foreign blackboard segments, the number of local blackboard accesses, the time the processing took if run as a sequential process on a single processor, the processing time if run in a multi-processor environment, and the parallel advantage realized for the run. The parallel advantage may be calculated with or without the parallel overhead costs.

The overhead for parallel processing, since most of the work was performed during the network partitioning, is limited to the cost of the network data transfers.

The user may access any element on any blackboard without regard for its location, just as the system does. This is done using the ACCESS and DEPOSIT macros. (ACCESS

'{variable-name})) returns the four field data item
(DEPOSIT {value}{variable-name})) replaces the values found
in {variable-name} with the value supplied. This is a
dangerous macro. Be sure you know what you're doing before
using DEPOSIT.

Summary

This chapter presented the basic structure, data-flow description method, and user instructions for loading and using DDAFCON. The basic structure consists of a virtual blackboard which is synthesized from blackboard segments distributed among a number of processors, and the agents which use the items found on the blackboard. The data-flow description method is a small collection of keywords which allow the salient properties of an agent to be described in a manner which is convenient for distribution. The user instructions include the sequences required and options available for loading and running a DDAFCON system.

V. Flight Planning Application Using DDAFCON

Background

This chapter presents the results of writing an application using DDAFCON. The application is a flight planner for general aviation. Since this application is intended to demonstrate the characteristics of DDAFCON and the general structure of a model-based planner which might be used in an airborne environment, a simplified "world model" is developed. Specifically, the flight planner plans for single-leg trips in a typical four passenger general aviation aircraft. The aircraft modeled for this application is a Piper Warrior (PA-28-161). This aircraft was chosen because the author is familiar with it, and because it is a popular airplane, typical of a popular class of airplanes (four-passenger, single-engine, fixed-gear). The performance data for this aircraft was extracted from the Piper company's owner's handbook (30). Since planning and replanning are essentially the same activity, a planner is a reasonable place to examine the complexity and results of implementing an AI type system which would be flown. DDAFCON provides the environment in which the application is realized. Since DDAFCON provides a parallel implementation of the algorithms which a planner would use, it allows the development of a real-time performance strawman.

A "model," as used here, refers to the collection of agents which, together, describe the data-flow through the problem. A model for flight planning must consider items such as airports, winds, directions of flight, airspeeds, power settings, etc. These items are coded into the agents. The agents are listed in Appendix B.

Some simplifying assumptions have been made in this application; however, none of the assumptions affect the complexity of the model. First, for all the scenarios, the major planning meta-theme used is "get to the destination as fast as possible." Second, the aircraft is assumed to have an area navigation (RNAV) capability. Thus it can fly directly from one point to another. This eliminates the added coding required to compute and manage a list of turn points. Third, only trips which can be made non-stop (no landing for fuel) are considered. If the aircraft can't be configured for a non-stop trip (the model's job to determine), the trip is not considered possible. The model can be extended to provide more capability. The simplification still allows many scenarios to be tested.

The model can reason about weight and balance, altitude to fly (as a function of: winds aloft, direction of flight (FAR part 91 VFR considerations), and distance to destination), aircraft performance (considering density altitude), power settings to use, fuel management, true

headings, and courses made good. All the resources are manipulated to find a solution to the problem presented.

Once started, the application prompts for any required information which can not be inferred such as a departure airport and a destination airport, expected loading, and weather. When finished, a trip profile is presented. The information found in the trip profile includes: final loading information including passenger seating, baggage location and initial fuel load; a climb profile which has the time to climb to cruise altitude, the fuel used, and the distance over the ground covered once reaching the cruise altitude; a cruise profile which lists the power setting to use, the airspeed to maintain and the time and location at which to initiate the descent to the destination airport, and the fuel used; a descent profile which shows the power setting to use, the fuel consumed (and reserves available), and the time of arrival at the destination at pattern altitude.

Although some simplifying assumptions are made as mentioned earlier, the result is a reasonable approximation of the flight planning activities performed by a general-aviation pilot. Accident reports show that poor fuel management is a major problem among pilots in general, and general-aviation pilots in particular, so it's quite possible that a simple application, as presented here, may do more flight planning than most pilots.

DDAFCON collects various statistics about the run (see Chapter IV, Using DDAFCON). These results are shown and discussed in the next section.

Results

The agents developed for this application are presented in Appendix B. Shown here is the result of the network structure analysis (see Table 4). Using the strong-components method, the variables were placed onto two blackboards which were assigned to two packets for placement on two processors. Using the parallel-paths method, the variables were placed on 18 blackboards which were ultimately assigned to eight packets. The discrepancy between the number of blackboards and the number of packets lies in the manner in which agents (and thus the code which is executed) is assigned to packets. Variables are assigned to blackboards based solely on the graphical analysis of the data-flow structure. Packets are formed when agents (which contain the code) are assigned to a packet which contains a blackboard which has most of its data on it (to maximize locality of reference). If, after all agents are assigned, a packet contains only a blackboard and no agents, the blackboard is moved to another packet and the empty packet is discarded.

Figure 15 shows the "main" constraint-set found in this application.

TABLE 4

AGENT PLACEMENT FOR FLIGHT PLAN APPLICATION

a) Strong-Components Method		
Packet		
Packet-1	Trip-profile Trip-legs Passengers Plan1	Weather Weight-and-balance Best-altitude
Packet-2	Power-setting GS-and-CMG Initial-fuel Range-demon Flight-phase	Fuel-burn Cruise-oat True-air-speed Range-agent Trip-winds
b) Parallel-Paths Method		
Packet		
Packet-1	Best-altitude	
Packet-2	Power-setting Trip-profile Plan1	Fuel-burn Range-Demon
Packet-3	Trip-legs	
Packet-4	Weight-and-balance Initial-fuel	
Packet-5	Passengers	
Packet-6	GS-and-CMG Range-agent	True-air-speed
Packet-7	Flight-phase	
Packet-8	Cruise-OAT Trip-wind	Weather

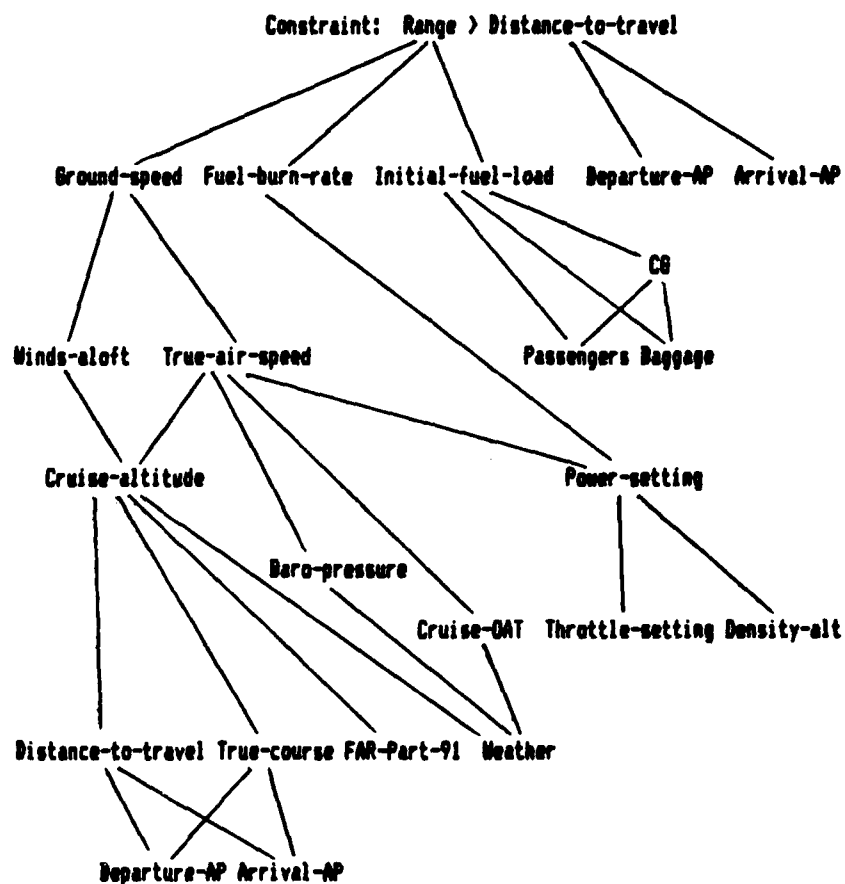


Fig. 15. Major Constraint-Set for Flight Plan Application

Note: The structure of the largest constraint-set found in this application is shown.

A number of scenarios were presented to this model. For this thesis, the results of two representative scenarios will be discussed as well as the performance of the DDAFCON system. The scenarios presented represent two trips between the Dayton International Airport in Vandalia, Ohio, and the Frederick Municipal Airport in Frederick, Maryland. The Frederick location was chosen for two reasons: 1) the author used to live there, and 2) it is a large enough distance (317 nm) that one must compromise between passenger load and fuel load when making a trip in a Piper Warrior.

Scenario 1 deals with a nominal flight which requires no conflict resolution (i.e. no constraints are broken). This provides a "best case" view of the planning paradigm. Scenario 2 presents some problems to the system. The plane is carrying too much payload to accommodate the fuel required to get to the destination at a high power setting (i.e. fast). The system must manipulate power settings, winds aloft, and altitudes to fly to solve this problem. Although perhaps not a "worst case," this scenario gave the system a good workout.

The results from each scenario will be analyzed and contrasted. From the performance data collected, some conclusions can be drawn about DDAFCON, the parallel processing it performs, the overhead the parallel processing requires, and the utilization of the processors.

The problem was presented to DDAFCON four times for each scenario, twice using the strong-components method, and twice using the parallel-paths method. For each of the methods, one presentation would include the parallel overhead, and one presentation excluded the parallel overhead. The speedup achieved is shown in Figures 16 through 19. These data also were used to estimate the overhead of parallel execution for this system.

In each figure, two sets of data are presented. Each set shows the time to complete the problem for the parallel execution of the problem, the time to complete for the linear execution of the system, and the speedup ratio for the presentation.

Scenario 1.

Setting: Take-off is from Dayton International Airport (DAY). The destination is Frederick Municipal Airport (FDK).

Load: One person (pilot) who weights 170 lbs., and 35 lbs. of baggage.

Weather: Clear, BP: 30.02 in Hg

winds aloft:

altitude (MSL):	3500	6500	9500
direction (true):	300	320	320
velocity (knots):	5	15	20

Plan:

True course:	095 degrees
Distance:	317 nautical miles
Cruise Alt.:	9500 ft. msl
Power:	90%
True Air Speed:	127 knots
Ground Speed:	140 knots
Fuel burn:	11.0 gph
Fuel Load:	48 g

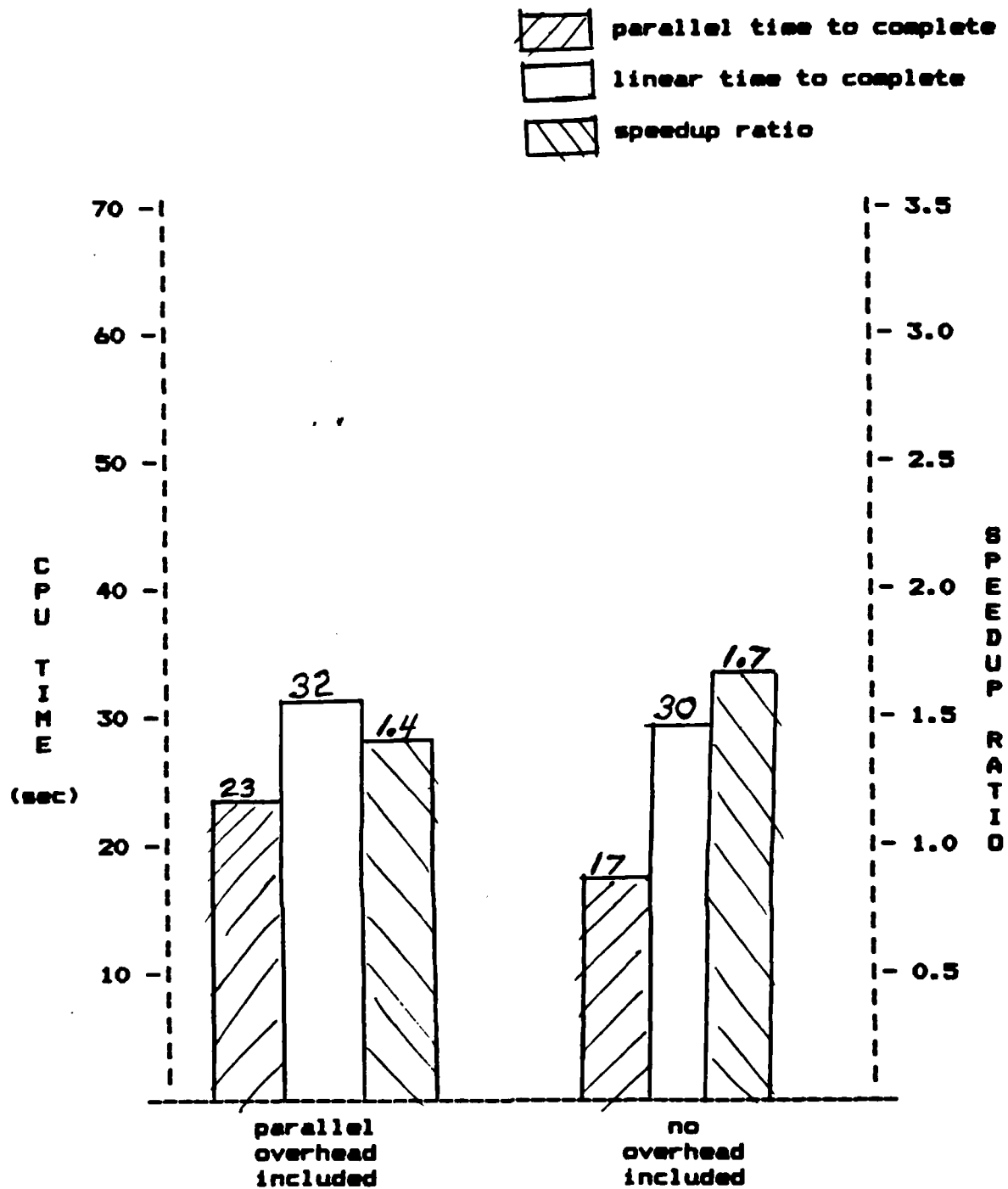


Fig. 16. Completion Times and Speedup for Scenario 1 Using SCM

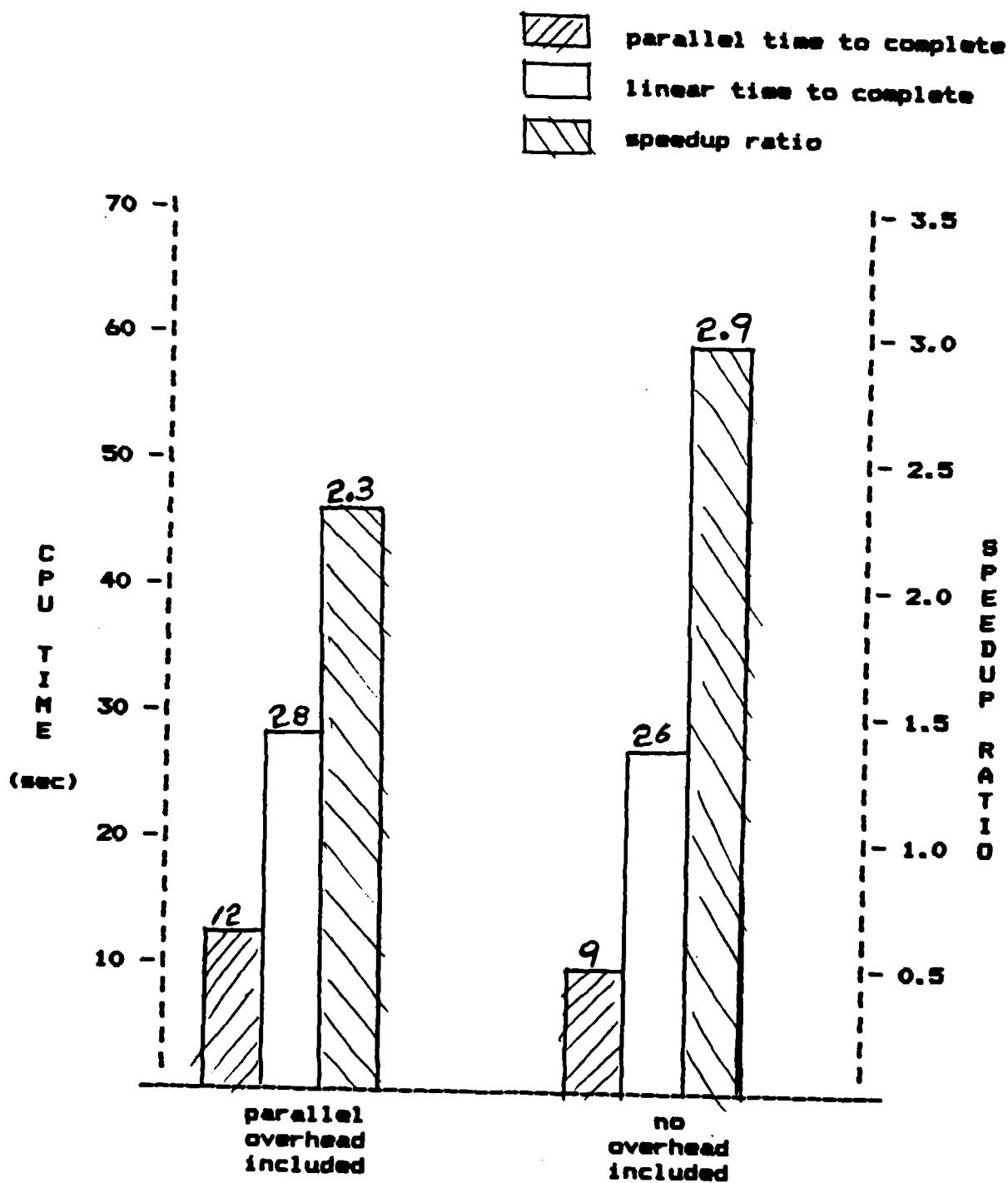


Fig. 17. Completion Times and Speedup for Scenario 1 Using PPM

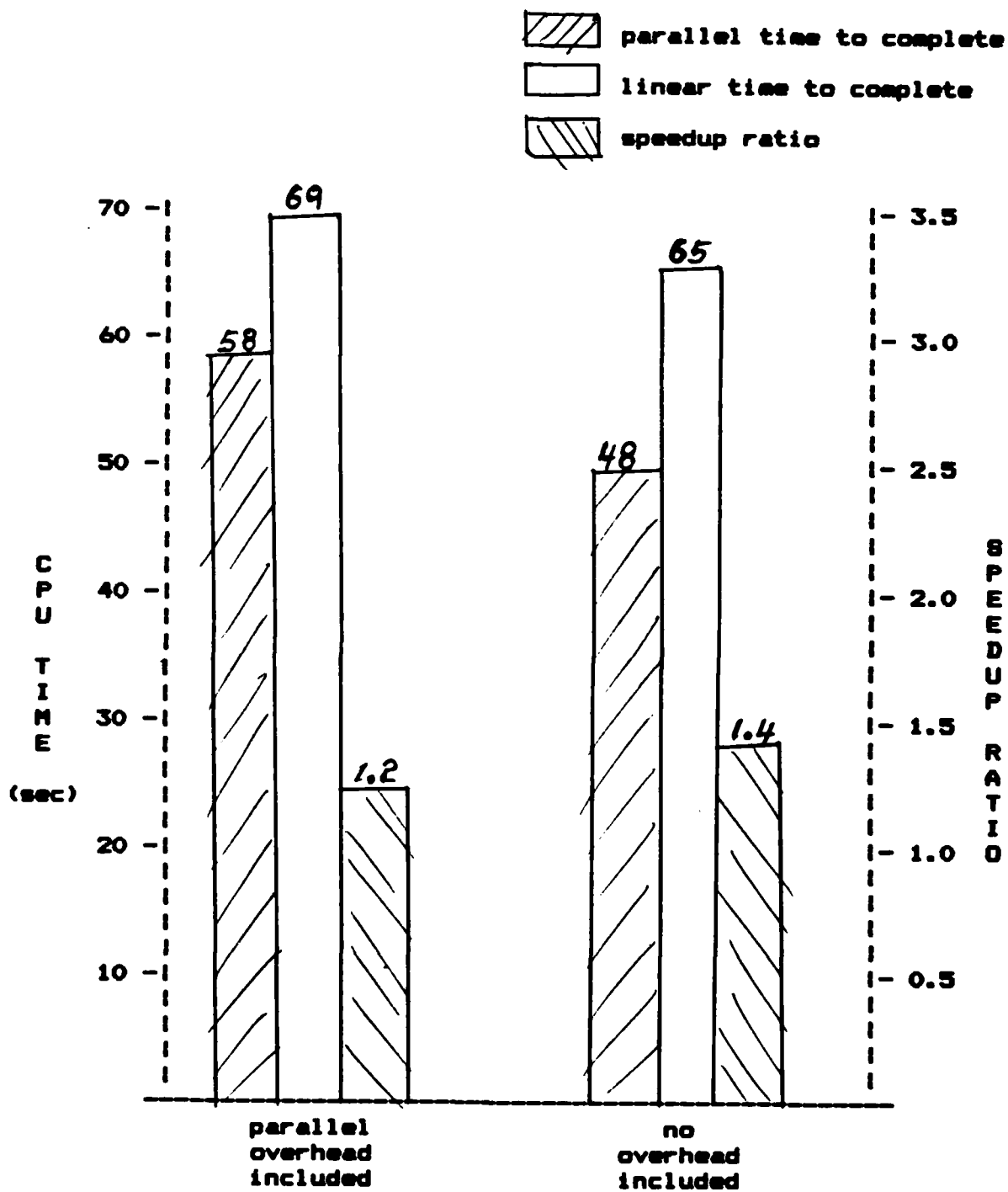


Fig. 18. Completion Times and Speedup for Scenario 2 Using SCM

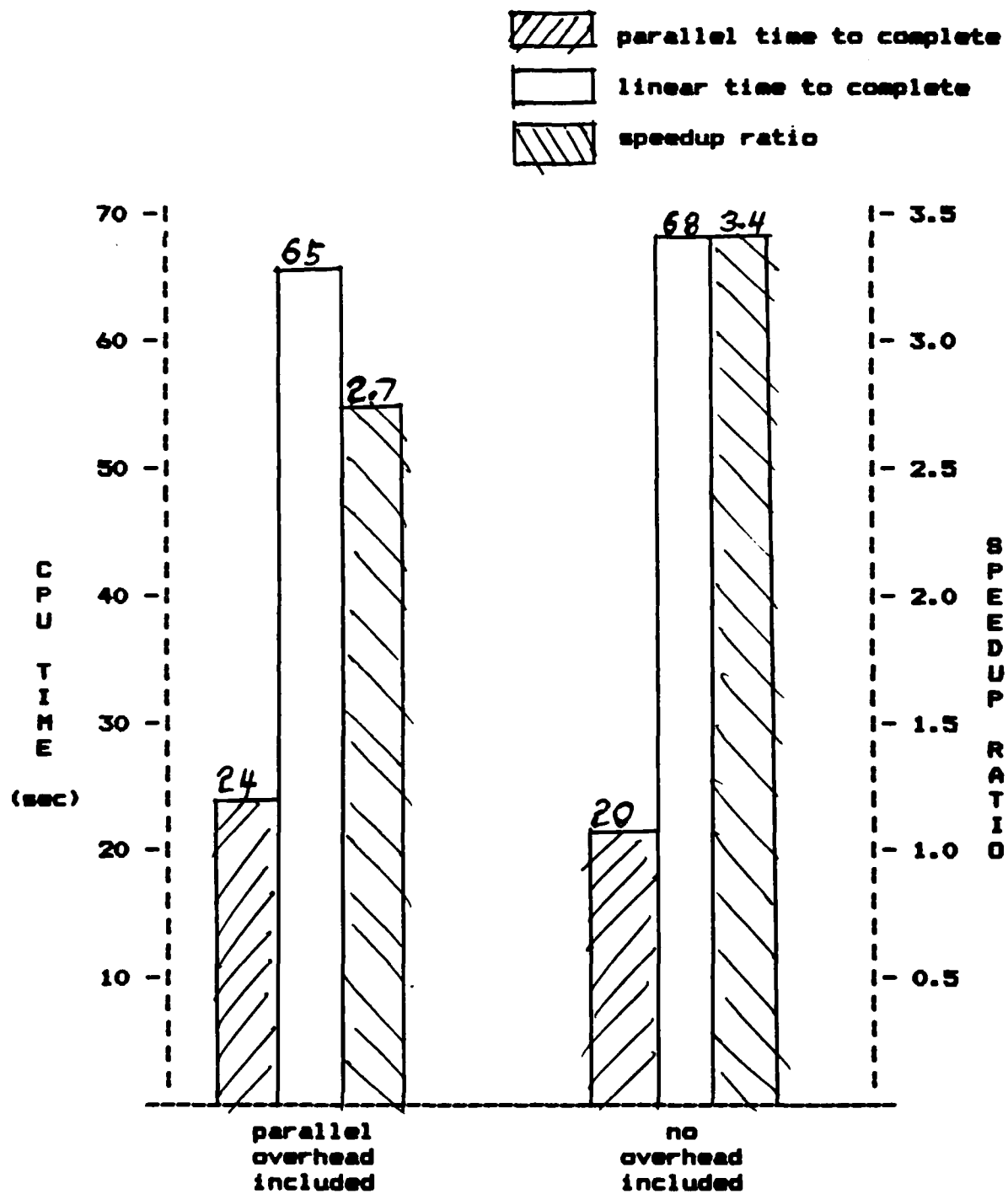


Fig. 19. Completion Times and Speedup for Scenario 2 Using PPM

T/O Weight: 1993 lbs.
CG: 87.7 in.
Calc. Range: 529 nm (with 25 min reserve
at 55% pwr)

Profile:

	time (hrs)	distance (nm)	fuel used (g)
climb	.28	22.4	3.2
cruise	1.94	272.3	21.7
descent	.18	23.1	1.4
total	2.40	317.8	26.3

DDAFCON produced a correct plan for scenario 1. The identical plan was produced by each of the four presentations. This demonstrates that the result produced is deterministic, and not sensitive to the location of the agents or data-items.

The speedup afforded by the parallel-paths method (Figure 17) was far superior to the result produced using the strong-components method (SCM) (Figure 16). In fact, the speedup achieved by the parallel-paths method (PPM) could not be realized by the SCM at all. With only two processors assigned, the absolute maximum speedup is 2x.

The SCM did a better job of utilizing the processors than the PPM. Since the linear time was 30 cpu seconds, the minimum time possible for the SCM was 15 cpu seconds. It was able to complete in 17 cpu seconds. Using Hayes' (18) method for measuring the processor utilization in

parallel systems (Figure 20), this shows a utilization of .70 compared to the PPM utilization of .30. Apparently, a doubling of the speed over the SCM result required four-times the processors.

	SCM (m=2)	PPM (m=8)
scenario 1	.70	.30
scenario 2	.60	.34

$$utilization = \frac{S}{m}$$

where

S = speedup achieved
m = number of processors used

Fig. 20. Processor Utilization for Both Scenarios

Note: Processor utilization for both scenarios and both partitioning methods. [Hayes method (18)]

Scenario 2.

Setting: Take-off is from Dayton International Airport (DAY). The destination is Frederick Municipal Airport (FDK).

Load: Pilot who weighs 170 lbs., with 25 lbs. of baggage; 3 passengers, each 170 lbs. with 25 lbs. of baggage.

Weather: Clear, BP: 30.02 in Hg

winds aloft:

altitude (MSL):	3500	6500	9500
direction (true):	300	320	320
velocity (knots):	5	17	20

Plan:

True course:	095 degrees true
Distance:	317 nautical miles
Cruise Alt.:	9500 ft. msl
Power:	65%
True Air Speed:	112 knots

Ground Speed: 126 knots
 Fuel burn: 7.5 gph
 Fuel Load: 25 g
 T/O Weight: 2440 lbs.
 CG: 89.1 in.
 Calc. Range: 320 nm (with 45 min reserve
 at 55% pwr)

Profile:

	time (hrs)	distance (nm)	fuel used (g)
climb	.28	22.4	3.2
cruise	2.20	272.3	16.5
descent	.18	23.1	1.4
total	2.66	317.8	21.1

DDAFCON produced a correct plan for scenario 2. A review of Figures 18 and 19 show that the time to produce this plan took about double the time for scenario 1. As in scenario 1, the four plans produced were identical.

From the analysis of DDAFCON, it was shown that the rate of convergence of a solution is partially a function of the rate in which a controlled variable becomes properly constrained (see Chapter III, Appendix A). This suggested an experiment. An assumption of the application as written is "go as fast as possible," thus the planner starts assuming 100 percent power, and only reduces this when it has to. Suppose the meta-theme assumption could be changed to "use as little fuel as possible," would that result in a performance speedup?

The new meta-theme was coded into the agents by changing the default value for power setting. The times and speedup were equivalent to scenario 1. This is because no constraints were violated when "maximum range" was considered rather than "quickest rate" for this scenario.

The result shows the importance of using proper meta-themes to reduce the search space. Proper meta-planning strategies for the flight domain would depend on what conditions were fixed and what conditions were not. For example, if a payload was fixed, the proper first strategy would be "maximum range." If this strategy was successful, higher power settings could be tried to arrive at the maximum speed possible. The real advantage to this approach is two-fold, the first answer would be either a proper plan with further iterations of the planning converging on an optimal plan or it would be a failure; secondly, the finding implies a further way to achieve even higher performance. If redundant DDAFCON systems are available the second system could be working on a refinement to the plan while the first is calculating the original plan.

Estimating the Overhead

Data were gathered for each scenario which did not include the network overhead for the parallel processing. These data were used to get a rough estimate of the overhead

costs. Without a specified physical network architecture for interconnecting processors it is impossible to actually assign costs; however, each partitioning method would require differing numbers of data transfers between processors, and this would impact on their relative efficiencies. It was felt that some measure of these differences should be included in the comparisons.

When gathering the input data or posting the output data for each agent, the time required to perform this i/o was added to the total running time for the packet and agent resided in if the specific data item was a network transfer.

The linear times were used to normalize the parallel times, and the normalized parallel times were used in the overhead estimate. Since the linear times did not ever include the parallel overhead costs they should always be equal. Slight differences in the linear run times among presentations of a scenario were due to unmeasurable overhead from the operating system.

$$\hat{O} = \frac{\overline{T_{P_{no}}} - T_{P_o}}{\overline{T_{P_{no}}}} \quad (8)$$

where

$\hat{O}$ = estimate of overhead

$\overline{T_{p_{no}}}$ = normalized parallel processing time
(no overhead)

T_{p_o} = parallel processing time (overhead included)

$$\overline{T_{p_{no}}} = \left| 1 + \frac{T_{l_{no}} - T_{l_o}}{T_{l_{no}}} \right| * T_{p_{no}} \quad (9)$$

where

$T_{l_{no}}$ = linear processing time (no overhead)

T_{l_o} = linear processing time (overhead)

$T_{p_{no}}$ = parallel processing time (no overhead)

Figure 21 shows the overhead estimates for two scenarios and the two partitioning methods. One can conclude that overhead, even when only marginally measurable, is a significant cost factor in parallel processing. Between the two partitioning methods, SCM always resulted in the lower overhead. This is not surprising since with only two processors it is more likely that data will be located on the same processor and network accesses will be less.

	SCM	PPM
scenario 1	.27	.24
scenario 2	.14	.20

Fig. 21. Overhead Estimates for Both Scenarios

Note: Overhead estimates for the two partitioning methods and the two scenarios.

Summary

This chapter has shown the results of writing a flight planning application using DDAFCON. The system is shown to be effective for this application. The contrasting of the two partitioning methods showed, as expected, that the ability to assign more processors to a problem allows the problem to be solved quicker. The utilization of the processors decreased, however, as the number assigned increased. Given the cost of hardware, and the importance of high performance, this "under-utilization" of the processors should not be too bothersome.

The speedup achieved grew as the problem difficulty grew. The problem structure was found to account for this observation since more time was spent in the parallel parts of the model when resolving constraint violations.

Meta-planning was shown to be an important reducer of search space which resulted in higher performance. In this demonstration, no meta-planning was performed by the system, rather the "human" did the meta-planning by changing defaults in the system to force the system to proceed in a specific direction.

Overhead was found to be significant. Roughly 25 percent of the time spent by the parallel implementation was spent moving data over the physical network. This finding verifies the "locality of reference" cautionary notes

found in the literature; and it validates the approach taken in DDAFCON of distributing data among the processors.

VI. Conclusions and Recommendations

Discussion and Conclusions

Since the major question asked in this thesis is: what about real-time performance?, the results uncovered by this application were quite illuminating. Recognizing that the performance improvement due to parallel processing is absolutely bounded by the number of processors used, and the number of processors used is a function of the number of processors that a problem can support (see Chapter II), successful parallel processing depends on the ability to partition the problem into pieces which can be run on separate processors. Since the data-flow of a problem was shown to define the parallel pieces of the problem structure, and since the data-flow can be viewed as a data network graph, graph-theory algorithms can be used to partition the graph into pieces which can be processed in parallel.

Two data-network partitioning methods were proposed and tested on an example flight-planning application. The strongly-connected components method (SCM) produced a network with two packets while the parallel-paths method (PPM), a new approach, produced a network with eight.¹ At first glance, it appears that the SCM could produce a two-fold

¹A packet is a collection of data and code which is assigned to a single processor. See Chapters II and III.

increase over a single processor and the PPM would offer almost an order-of-magnitude improvement. This was not to be the case. While SCM managed to offer a 1.2x - 1.4x improvement, the PPM offered a 2.3x - 2.7x improvement over linear processing. This was nowhere near the theoretical maximum of an 8x improvement for the PPM; however, 2.3x is better than the SCM method could ever hope to produce (two processors limit the maximum performance to 2x).

The reason for the apparent failure of the parallel-path method to perform as expected was both the strong partial-order imposed by the flight planning paradigm and the lack of a balanced load on each processor. Although it was possible to partition the problem among eight processors, the order of processing among the pieces didn't permit the expected advantage to be felt.

A conclusion one can draw from these results is that flight planning benefits marginally from parallel processing. Having said this, what about a real-time application of flight planning? Many overlapping constraint-sets (see Chapter III "a complexity analysis") and the inability to partition the problem into many independent pieces may preclude a full airborne planning system that reasons from a detailed world model. That is not to say that partial systems on the aircraft or ground-based off-line systems aren't possible; just that a full-blown-all-

conditions-accounted-for system model seems improbable for real-time reasoning in the aircraft.

A review of the data gathered shows that parallel processing did indeed help. In fact the parallel advantage grew as the problem got harder. The speedup in scenario 2 was greater than the speedup in scenario 1. This result is attributable more to the structure of the flight planning problem than to parallel processing per se. For this problem, the space which was searched during the backtracking could be searched in parallel. If a problem were such that the backtracking took place along a linear section, the results would be the opposite of those obtained. This points to the problem structure as the final determiner of potential performance. If the real-time requirement for this problem were "an answer within 30 seconds," then parallel processing (using the PPM for processor assignment) might render real-time performance. The equivocation is due to the "halting" problem discussed in an earlier chapter.

Another conclusion is: no conclusion is possible for other problems.

. . . there is strong evidence to suggest that all NP-Complete problems are intractable . . . almost all sequencing problems stated in complete generality are NP-Complete. . . .

— E. G. Coffman, Jr. (6)

Since it can be shown that planning, in the most general formulation, is NP-Complete (see Appendix A), DDAFCON,

which implements the major ideas of this general planning approach, must be NP-Complete also. Each activated agent represents a further extension and branching of the search tree which results in a solution (plan). Some potential branchings of the full search tree are inadmissible due to the scheduling paradigm; that is, only agents with at least one new value in the input vector will be scheduled. Thus any order relation among the data and agents cuts down on some branches. This does not make the problem tractable, it only reduces the effective branching factor (which is represented by the magnitude of the mantissa). As pointed out in the introduction, the mantissa would need to be reduced to one to change the nature of the problem, or sufficient processors must be available; however, "sufficient processors" may prove to be unbounded. The major lesson is that one must know and analyze the application to determine its potential for parallel processing and real-time performance (not a new lesson perhaps, but an affirmation of an old lesson many times forgotten).

This thesis has shown that it is not enough to merely state that one is going to take an AI application and put it on a parallel processing machine to get real-time performance. If readers become skeptical of claims of this type and ask the following questions then this thesis effort has made an important contribution.

1. How much faster will your application run?
2. How do you know this?
3. How many independent pieces does the application have (i.e. how many processors can be assigned given the problem structure)?

The introduction and analysis of the notion of dependency-sets is a further contribution. It shows that the complexity of a constraint satisfaction system is not a function of the number of variables to be constrained, but of the number of constraints to be satisfied, and their interaction. Since all planning systems are constraint satisfaction systems of one sort or another, this notion has a wide applicability.

Recommendations

Much work needs to be done both technically and theoretically. On the technical side, DDAFCON is in its infancy. Currently, only two constraint relations are recognized, "less than" and "greater than." This should be increased to the full set of logical binary relations as a minimum. Also, constraint-sets should be explicitly identified. This capability would permit certain unsolvable conditions to be found, and allow the calculation of problem complexity directly (since constraint-sets were only discovered during the analysis of DDAFCON, there is no recognition of them at all in the code). The user

interface for DDAFCON is essentially nonexistent. Whether a user interface belongs in DDAFCON or as an application-provided piece is left for others to decide.

Much theory is left undone and incomplete. The notion of constraint-sets is an intriguing and open area for some bright person to develop the theory in a more formal way. I believe it is a diamond in the rough. It may also be approached as a controls problem using the message-flow as the feedback in the system.

The degree of automation anticipated for the PA presupposes the ability of the computer to deduce the goals of the pilot/aircraft system and to formulate and project plans to discover goal interactions (i.e. the computer "understands" the pilot). This need is discussed by Cross and colleagues (12) and Geddes (17). To implement these notions, meta-planning should be examined, and appropriate meta-themes should be developed to help constrain the search space (see Chapter V).

Other AI problems and solutions should be examined for their potential for parallel processing and consequent real-time implementation. My suspicions are that diagnosis type problems, which are combinatorially explosive, would benefit from parallel processing. This includes malfunction diagnosis and emergency procedures. I also suspect they would be very expensive in the number of processors required since the problem forms a tree-like structure.

DDAFCON's structure would lend itself to exploring this conjecture using the parallel-paths analysis, and Chadrsekaran's method (5) of establish/refine would be an excellent model to follow.

A final note on model-based systems: the proper grain size for systems which are to run in real-time must be investigated. For a model-based system, this means deciding on the model depth that can be supported in the time available for an answer. Only then should the question of adequacy of the answers provided at this grain size be judged. This analysis method would preclude the creeping expansion of the "knowledge base" over time, as this is the usual growth pattern of all code. For real-time systems, which will probably be running on the ragged edge of acceptable performance, further growth in the knowledge base might well result in unacceptable performance.

What, then, do humans do? Are we really capable of examining a "deep-knowledge model" of the world in time-critical situations and drawing inferences from the examination? If we are honest with ourselves, the answer must be "no." Evidence for this position abounds. It is the implicit recognition of this condition that prompts the development and teaching of "bold-face" procedures to pilots. These are the things that must be done quickly, without thinking, in an emergency situation (that is why

they appear in the pilot's operating handbook in large, dark type; i.e. bold-face). Learned responses characterize most of the procedures that one goes through in an emergency situation. The "thinking" was done "off-line." If no procedures exist, often panic ensues. Perhaps panic is the recognition of the inability to examine one's internal model (or maybe it's the recognition that one doesn't have an internal model!). Doing the "thinking" off-line might be the way to attack the problem. This is a technique that might be used to "cock" the system. The system would use its model to examine likely scenarios, perhaps perform a sensitivity analysis (10; 11) on the results, and prepare "triggers" which would fire off bold-face equivalent procedures without examining the model "on-line" in real-time.

A question must be asked. Would we (human-kind) accept this level of performance from a machine? I don't know...maybe not... but it appears that this is the best that can be done in many cases.

Appendix A: An Abstract Model of Planning and a
Proof of "Planning" as an NP-Complete Problem

This appendix introduces some nomenclature and ideas about the process called "planning." A symbolic representation for planning is introduced. From this representation it is easily shown that a non-deterministic turing machine can be built which would accept this representation in polynomial time, thus establishing "planning" as a language which resides within NP. Further, it is shown that "planning" can be mapped to the set-covering problem using a simple algorithm of polynomial complexity. By showing that "planning" is polynomially reducible to a problem which is known to be NP-Complete, it is proven that "planning" is NP-Complete.

Planning may be thought of as the 3-tuple

$$\langle E, O, P \rangle \quad (10)$$

where

$$E = \{e_1, e_2, e_3, \dots, e_\ell\}$$

is the set of objects over which planning occurs (they are the "managed" items), and each

$$e_i \text{ in } E$$

consists of a current value V and a set of constraints

$$C = \{p_1(V), p_2(V), p_3(V), \dots, p_m(V)\} \quad (12)$$

where $p_k(V)$ is a predicate on V which serves to bound the acceptable values of V on e_i .

Definition 1:

An object e_i is said to be PROPERLY CONSTRAINED if, for its current value $e_i(V)$,

$$e_i(p_k(V)) = \text{TRUE}; k = 1..m.$$

Thus, (let "&" be the "and" operator)

THEOREM 1: An object e_i in E is properly constrained if and only if

$$p_1^i(V) \& p_2^i(V) \& \dots \& p_m^i(V) = \text{TRUE}.$$

Proof:

Assume $p_1^i(V) \& p_2^i(V) \& \dots \& p_m^i(V) = \text{FALSE}$.

Then there exists $p_k^i(V) = \text{False}$. But by definition 1, if $p_k^i(V) = \text{FALSE}$, e_i is not properly constrained. Q.E.D.

$$O = \{o_1, o_2, o_3, \dots, o_n\} \quad (13)$$

is the set of operators which map values V to objects e_i .

A plan

$$P = (o_1 o_2 o_3 \dots o_r) \quad (14)$$

is a string of length r formed by the concatenation of operators

$$o_j \text{ in } O, j = 1..n$$

A plan P applied to a set of objects E (written $P(E)$) denotes the operators $o_1 o_2 o_3 \dots o_r$ in P applied, in order, to the objects e_i in E .

Definition 2:

A plan over E $P(E)$ is ACCEPTABLE if, for every o_j in P , $j = 1..r$, every e_i in E , $i = 1..l$, is properly constrained.

THEOREM 2: A plan over E $P(E)$ is ACCEPTABLE if and only if

$$C^1(p_k^1) \ \& \ C^2(p_k^2) \ \& \ \dots \ \& \ C^l(p_k^l) = \text{TRUE}.$$

Proof:

Assume $C^1(p_k^1) \ \& \ C^2(p_k^2) \ \& \ \dots \ C^l(p_k^l) = \text{FALSE}$. Then there exists $C^i(p_k^i) = \text{False}$. But by definition 2, if $C^i(p_k^i) = \text{FALSE}$, then e_i is not properly constrained, and if e_i is not properly constrained, the plan is not acceptable. Q.E.D.

Recall that planning is the triple $\langle E, O, P \rangle$.

Usually, E and O are given and the object is to find a P such that $P(E)$ is acceptable. The search for P can be thought of as a search of a tree where the nodes are the operators in set O , and E represents the state of the system. As the tree is searched the operator represented by the current node is applied to E . Search stops when all objects are properly constrained.

If a non-deterministic turing machine (NDTM) guesses at the first operator, and at each step guesses at the next operator, the string which represents the plan

can be discovered in time proportional to the cardinality of O . Thus "planning" is an NP language.

Given L objects and N operators, "planning" can be mapped to a set-covering problem (SCP) in the following way (note: the set-covering algorithm used is taken from Cristofides' Graph Theory). Each object in E is mapped as an item to cover in the SCP (order L time). Then, the N operators are mapped into N blocks of N distinct operators in each block (order N squared time). The problem is then solved as a standard SCP problem (see Cristofides [9] for a discussion of the algorithm). The items covered is determined by the application of the operator to the items when an operator is chosen in a block. If an item is properly constrained, it is covered. Thus the translation time is order $(N \text{ squared} + L)$ which is polynomial in time, and "planning" is shown to be NP-complete.

Appendix B: Flight Planning Application Code

This appendix contains the "agents" written for the flight planning application. See Chapter five for a description of the application. See Chapter four for a definition of the language used.

```
;
;
; AGENTS
```

```
(agent-frame      plan1

(input  ())

(output (true-course distance tko-ap ldg-ap))

(defaults (
  (tko-ap (ask-for-airport-info "the departure airport"))
  (ldg-ap (ask-for-airport-info "the arrival airport"))
))

(control (tko-ap ldg-ap true-course distance))

(change-funcs (
  (tko-ap (lambda (x)
    (progn
      (why)
      (setq dummy tko-ap))))
  (ldg-ap (lambda (x)
    (progn
      (why)
      (setq dummy ldg-ap))))
  (true-course (lambda (x)
    (progn
      (why)
      (setq dummy true-course))))
  (distance (lambda (x)
    (progn
      (why)
      (setq dummy distance))))))
```

```

(functions (
  (true-course (ifd* (find-true-course (tko-ap i) (ldg-ap i))))))

(aux-functions (
  (find-true-course (from to)
    ; there are two problems with this routine:
    ; 1) it assumes northern hemisphere
    ; 2) it assumes 1 min of long = 1 nautical mile * COS (avg lat)
    (progn ()
      (setq lat-from (squish (lat from))
            lat-to (squish (lat to))
            long-from (squish (long from))
            long-to (squish (long to)))
      (setq adj-to (* (abs (difference long-to long-from))
                     (abs(cosd (quotient
                               (plus lat-to lat-from) 2.))))
            ops-to (abs (difference lat-to lat-from))
            hyp (sqrt (plus (* adj-to adj-to)
                             (* ops-to ops-to))))
      (setf distance hyp) ;note - this assigns the other output value
      (setq theta (* 57.32484 (atan ops-to adj-to)))
      (cond
        ((<= lat-to lat-from) ; a southerly direction
         (cond
           ((lessp long-from long-to) ; a south-westerly direction
            (difference 270. theta))
           (t (plus 90. theta))))
          ; a south-easterly direction
          ; a northerly direction
        (cond
          ((<= long-from long-to) ; a north-westerly direction
           (plus 270. theta))
          (t (difference 90. theta)))))) ; a north-easterly direction

```

```

(squish (location)
  (plus (* (1-degrees location) 60.)
    (1-minutes location)
    (div (1-seconds location) 60.)))
(init-flight-planner ()
  (progn ()
    (deposit '(nil nil 0. nil) tko-ap)
    (deposit '(nil nil 0. nil) ldg-ap)))
)))
;=====
(agent-frame      best-alt

  (input (true-course distance winds-aloft))

  (output (cruise-alt))

  (control (cruise-alt))

  (defaults (
    (cruise-alt 2500.)
    (winds-aloft (ask-for-winds-aloft))
  ))

  (constants ( (area-elevation 1000.)))

  (functions (
    (cruise-alt (ifd* (find-best-altitude (true-course i)
      (distance p) (winds-aloft i))))))

  (change-funcs (
    (cruise-alt (lambda (x)

```

```

(prog (new-alt we-are-low)
  (cond
    ((lessp cruise-alt
      (plus area-elevation 3000.))
      (setq we-are-low t))
    (t (setq we-are-low nil)))
  (cond
    ((equal x 'increase)
      (cond
        ((we-are-low )
          (cond
            ((and(>= true-course 0)
              (< true-course 180.))
              (setq new-alt 5500.))
            (t (setq new-alt 4500.)))
          ( t (setq new-alt
            (plus cruise-alt 2000.))))))
      (t
        (cond
          ((we-are-low)
            (setq new-alt cruise-alt))
          ; if low, don't go lower
          (t (setq new-alt
            (difference cruise-alt 2000.)))))
        (return new-alt))))))

(aux-functions (
  (find-best-altitude (tc d wa)
    (prog (alt wind-list max-alt wind-component)
      (setq alt 2500.) ; assume you're gonna fly low
      (setq max-alt (max (* 1000. (/ (float d) 11.0))
        (plus area-elevation 1000.))) ; rule of thumb
      (setq wind-list
        (for& (this-wind in wa)
          (save

```

```

(cond
  ((and (lessp (w-alt this-wind) max-alt)
        (allowable-altitude (w-alt this-wind) tc))
    (list (minus (times (cosd
                        (find-angular-difference tc
                        (w-dir this-wind)))
                  (w-vel this-wind)))
          (w-alt this-wind))))))
(setq wind-component -1000.)
(cond ; make sure the altitudes are in ascending order
  ((> (cadr (car wind-list)) (cadr (caddr wind-list)))
    (setq wind-list (reverse wind-list)))
  (for& (this-wind in wind-list)
    (do
      (cond
        ((null this-wind))
        ((>= (car this-wind) wind-component)
          ; >= favors higher altitudes with the same wind component
          ; along the true course (higher true airspeeds found here)
          (setq wind-component (car this-wind))
          (setq alt (cadr this-wind))))))
    (return alt)))

(allowable-altitude (a c) ; west is even 1000's plus 500 ,
  ; east is odd 1000's plus 500
  (cond
    ((and (>= c 0) (< c 180.))
      (cond
        ((oddp (fix(div a 1000.))) t)
        (t nil)))
    (t
      (cond
        ((evenp (fix (div a 1000.))) t)
        (t nil))))))

```

```

(find-angular-difference (al a2)
  (prog (the-diff)
    (setq the-diff (difference al a2))
    (cond
      ((lessp the-diff 0) (setq the-diff
        (plus 360.0 the-diff))))
    (cond
      ((greaterp the-diff 180.) (setq the-diff
        (difference 360.0 the-diff ))))
    (return (float the-diff))))

```

```

0
;=====

```

```

(agent-frame      trip-wind-agent

  (input (cruise-alt winds-aloft))

  (output ( cruise-winds ))

  (constraints (
    ((greaterp cruise-alt 0) "one must fly off the ground")))

  (functions (
    (cruise-winds
      (prog ()
        (for& (a-wind in winds-aloft)
          (do
            (cond
              ((equal cruise-alt (w-alt a-wind))
                (return a-wind)))))))
    )

```

```

(agent-frame ac-leg-profile-agent
  (input (current-weight cruise-alt))
  (output (ac-leg-profile))
)

(agent-frame flight-phase-agent
  ; a demon which sets the initial flight phase value
  ; since initially, the flight phase will be 'nil, the default is used
  (input ( flight-phase))
  (output( flight-phase))
  (defaults ( (flight-phase 'pre-flight))))

(agent-frame initial-fuel-agent ; a demon which calculates the initial fuel load
  (input ( flight-phase current-fuel current-weight))
  (output( current-fuel))
  (functions (
    (current-fuel (ifd* (calc-initial-fuel (flight-phase i)
      (current-fuel i)
      (current-weight ip))))))

(aux-functions (
  (calc-initial-fuel (fp cf cw)
    (prog (dummy-fuel)
      (cond
        ((equal fp 'pre-flight)
          (cond

```

```

((or (equal cf 0) (lessp cw max-gross-weight))
 (setg dummy-fuel (min
  (div
   (difference max-gross-weight cw) 6.)
   max-fuel))))
(return dummy-fuel)
(T (return cf))))))

```

(agent-frame passengers

```

(input ())
(output ( passenger-list baggage-weight))

(control ( baggage-weight passenger-list))

(change-funcs (
  (passenger-list (lambda (x)
    (progn ()
      (why)
      (setg dummy passenger-list))))
  (baggage-weight (lambda (x)
    (progn ()
      (cond
        (**show-reasons**
         (msg n " I must " x " the baggage weight" n)
         (why)))
      (setf baggage-weight
        (cond
          ((equal x 'decrease)
           (cond
            ((greater p baggage-weight 200.) 200.)
            (t (difference baggage-weight 5.))))
          ((equal x 'increase)
           (plus baggage-weight 5.))))
        ))))

```

```

(defaults (
  (passenger-list (ask-for-passenger-data))))
(agent-frame  weight-and-balance

(input ( current-fuel passenger-list baggage-weight))

(output ( current-weight current-cg ))

(constants (
  ( max-gross-weight 2447. )
  ( max-baggage-weight 200.)
  ( empty-weight 1500.)

  ( fwd-lmt-CG 83.)
  ( aft-lmt-CG 93.)
  ( empty-weight-moment 128850.)
  ( front-pass-arm 80.5)
  ( rear-pass-arm 118.1)
  ( baggage-arm 142.8)
  ( fuel-arm 95.0)
  ( max-fuel 48.)))

(defaults (
  ( current-fuel 0.)
  ( passenger-list (ask-for-passenger-data))
  ( baggage-weight 0.)
  ( current-weight empty-weight)
  ( current-CG 86.)))

(functions (
  (current-weight
    (plus empty-weight
      (plus (ifd* (passenger-weight (passenger-listp)))
        (plus baggage-weight
          (* 6. current-fuel))))))

```

```

(current-CG
  (div
    (plus empty-weight-moment
      (plus (ifd* (passenger-moment (passenger-list p)))
        (plus (* baggage-weight baggage-arm)
          (* (*6 current-fuel) fuel-arm)))) current-weight))))

(constraints (
  ((lessp current-weight max-gross-weight)
    "takeoff weight may not exceed the max allowable")
  ((greaterp current-CG fwd-lmt-CG)
    "CG must not be forward of CG envelope")
  ((lessp current-CG aft-lmt-CG)
    "CG must not be aft of CG envelope" )
  ((lessp baggage-weight max-baggage-weight)
    " baggage weight can't exceed 200 lbs"))))

(aux-functions (
  ( passenger-weight (p-list)
    (apply 'plus (for& (person in p-list)
      (save
        (p-weight person))))))
  ( passenger-moment (p-list)
    (prog (moment pass-number)
      (setq moment 0 pass-number 1)
      (tagbody loop
        (cond
          ((null p-list))
          ((lessp pass-number 3) ; front-seat passengers
            (setq moment
              (plus moment
                (* front-pass-arm
                  (p-weight (car p-list))))))
            (setq p-list (cdr p-list))
            (go loop))

```

```

((greaterp pass-number 2) ; rear-seat passengers
 (setq moment
  (plus moment
   (* rear-pass-arm
    (p-weight (car p-list)))))
 (setq p-list (cdr p-list))
 (go loop)))
(return moment))))

```

(agent-frame weather

 (input ()))

 (output (baro-pressure winds-aloft))

 (defaults (

 (baro-pressure 29.92)

 (winds-aloft (ask-for-winds-aloft))))))

(agent-frame cruise-oat-agent

 (input (cruise-winds))

 (output (cruise-oat))

 (functions (

 (cruise-oat (ifd* (w-temp (cruise-winds i)))))

(agent-frame true-as

 (input (cruise-alt baro-pressure cruise-oat power-setting))

 (output (tas))

```

(functions (
  (tas (ifd* (calc-tas (cruise-alt p)
    (baro-pressure ip)
    (cruise-oat p)

(aux-functions (
  (calc-tas (ca bp coat ps)
    (prog (as)
      (setq pa (plus (difference 29.92 bp) ca))
      (setq da (plus pa
        (div
          (float
            (difference coat
              (difference 59.
                (* 2.5 pa))))
          2.5)))
      (cond
        ((equal ps 55.) (return (plus (* .001 da) 94.)))
        ((equal ps 65.) (return (plus (* .001 da) 105.)))
        ((equal ps 75.) (return (plus (* .001 da) 112.)))
        ((equal ps 100.) (return 127.)))))))

(agent-frame      range-agent

  (input ( gs fuel-burn current-fuel ))
  (output (range))

  (constants (
    (reserve-fuel 5.8))) ; 45 min at 55% power

  (functions (
    (range (* (div (float (difference current-fuel
      reserve-fuel))
        fuel-burn) gs))))))

```

```

(agent-frame      range-demon
  (input (range distance))
  (constraints (
    ((greaterp range distance)
     "must have the range to make the destination"))))

(defstruct (profile-info (:type :list))
  p-time
  p-distance
  p-fuel-used)

(agent-frame      trip-profile
  (input (gs tko-ap ldg-ap cruise-alt fuel-burn distance))
  (output (climb-profile descent-profile cruise-profile))

  (constants (
    (climb-speed 79.0)
    (descent-speed 126.0)
    (climb-rate 30000.0) ; per hour rate at 500 fpm
    (descent-rate 42000.0); per hour rate at 700 fpm
    (climb-fuel-burn 11.4)
    (descent-fuel-burn 7.8)
    (pattern-alt 800.0)))

  (functions (
    (climb-profile
     (prog (dummy)
      (setq dummy (make-profile-info))
      (setf (p-time dummy)
            (div (float (difference cruise-alt
                                   (elevation tko-ap))))

```

```

    climb-rate))
  (setf (p-distance dummy) (* (p-time dummy)
    climb-speed))
  (setf (p-fuel-used dummy) (* (p-time dummy)
    climb-fuel-burn))
  (return dummy)))

(descent-profile
 (prog (dummy)
  (setq dummy (make-profile-info))
  (setf (p-time dummy)
    (div
     (float
      (difference
       cruise-alt
       (plus (elevation tko-ap)
        pattern-alt)))
      descent-rate))
    (setf (p-distance dummy)
      (* (p-time dummy) descent-speed))
    (setf (p-fuel-used dummy)
      (* (p-time dummy) descent-fuel-burn))
    (return dummy)))

(cruise-profile
 (prog (dummy)
  (setq dummy (make-profile-info))
  (setf (p-distance dummy)
    (difference distance
     (plus (p-distance climb-profile)
      (p-distance descent-profile))))
  (setf (p-time dummy)
    (div
     (float
      (p-distance dummy)) (float gs)))
  (setf (p-fuel-used dummy)
    (* (p-time dummy) fuel-burn))
  (return dummy))))

```

```

(agent-frame      fuel-burn-agent
  (input (power-setting))
  (output (fuel-burn))
  (defaults (
    (power-setting 100.)))
  (functions (
    (fuel-burn (ifd* (calc-fuel-burn (power-setting p))))))
  (aux-functions (
    (calc-fuel-burn (ps)
      (cond
        ((equal ps 100.) 11.2)
        ((equal ps 75.) 10.0)
        ((equal ps 65.) 8.8)
        ((equal ps 55.) 7.8))))))

(agent-frame      power-setting-demon
  (input  (cruise-alt power-setting))
  (output (power-setting))
  (control (power-setting))
  (functions (
    (power-setting (cond
      ((greater p cruise-alt 14000.)
       (max power-setting 55.))
      ((greater p cruise-alt 12000.)
       (max power-setting 65.))
      ((greaterp cruise-alt 8500.)
       (max power-setting 75.))
      (t power-setting)))))

```

```

(change-funcs (
  (power-setting (lambda (x)
    (progn ()
      (cond
        (**show-reasons**
         (msg n
          " I need to " x
          " the power setting")
         (why)))
        (cond
         ((equal x 'decrease)
          (cond
            ((equal power-setting 100.)
             (setf power-setting 75.))
            ((equal power-setting 75.)
             (setf power-setting 65.))
            ((equal power-setting 65.)
             (setf power-setting 55.)))))
         ((equal x 'increase)
          (cond
            ((equal power-setting 55.)
             (setf power-setting 65.))
            ((equal power-setting 65.)
             (setf power-setting 75.))
            ((equal power-setting 75.)
             (setf power-setting 100.)))))
        )))))

(agent-frame   g-speed-and course-made good
  (input (tas true-course cruise-winds))
  (output (gs track true-heading))

```

```

(functions (
  (true-heading
   (ifd*
    (calc-cmg (true-course p)
              (tas i)
              (cruise-winds i))))))

(control (true-heading track))

(change-funcs (
  (true-heading (lambda (x)
    (progn ()
      (cond
        ((equal x 'increase)
         (setf true-heading (plus true-heading 2.))))
        ((equal x 'decrease)
         (setf true-heading (difference true-heading 2.))))
      (cond)
        ((lessp true-heading 0)
         (setf true-heading (plus true-heading 360.))))
        ((greaterp true-heading 360.)
         (setf true-heading
          (difference true-heading 360.))))))))))

(aux-functions (
  (calc-cmg (tc as wind)
    (cond
      ((equal as 0) 0)
      (t
       (prog (x-wind-cmp t-wind-cmp cmg-ang cmg)
         (setq t-wind-cmp
          (minus
           (* (cosd
              (find-angular-difference tc
                (w-dir wind))))

```

```

(w-vel wind)))
(setq x-wind-cmp
  (abs
    (*
      (sind
        (find-angular-difference tc
          (w-dir wind))))
      (w-vel wind))))
(setq cmg-ang
  (atan
    (div x-wind-cmp as)
    (div (sqrt (plus (* as as)
      (* x-wind-cmp x-wind-cmp))))
      as)))
(setf gs
  (plus
    (sqrt
      (difference (* as as)
        (* x-wind-cmp x-wind-cmp)))
    t-wind-cmp))
(setq norm-ang
  (plus 360.
    (max (w-dir wind)
      tc)))
(setq norm-wind-ang
  (plus (w-dir wind) norm-ang))
(setq norm-tc-ang
  (plus tc norm-ang))
(cond
  ((greaterp norm-wind-ang norm-tc-ang
    (setq cmg (plus tc cmg-ang))))
  (t (setq cmg (difference tc cmg-ang))))
(cond
  ((greaterp cmg 360.)
    (setq cmg (difference cmg 360.))))

```

```

)
(cond
  ((lessp cmg 0) (setq cmg (plus cmg 360.))))
(return cmg))))))

(agent-frame      true-heading-demon

(input (track true-course ))

(constraints (
  ((greaterp (plus (normalize true-course) 5.) track)
   " must maintain the proper track over the ground")
  ((lessp (difference (normalize true-course) 5.) track)
   " problems with the track due to mind effects"))))

(aux-functions (
  (normalize (tc)
    (cond
      ((greaterp (difference track tc) 180.)
        (plus tc 360.))
      ((lessp (difference track tc) -180.)
        (difference tc 360.))
      (t tc))))))

(agent-frame      trip-legs-agent

(input (ac-leg-profile distance))

(output (trip-legs))

)

```

Bibliography

1. Aho, A., J. Hopcroft, and J. Ullman. The Design and Analysis of Computer Algorithms. Reading MA: Addison-Wesley, 1974.
2. Bahnij, R. An Intelligent Fighter-Pilot-Aide for Tactical Mission Planning. MS thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1985.
3. Barr, A., and E. Feigenbaum. The Handbook of Artificial Intelligence, Vols. 1, 2. CA: William Kaufmann, 1982.
4. Bobrow, D. G., and P. J. Hayes. "AI: Where Are We," Artificial Intelligence, 25: 375-415 (1985).
5. Chandrasekaran, B., and S. Mittal. "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving," Int. J. Man-Machine Studies, 19: 425-436 (1983).
6. Coffman, E. G., ed. Computer and Job/Shop Scheduling Theory. New York: John Wiley, 1976.
7. Coffman, E. G. and P. Denning. Operating System Theory. Englewood Cliffs NJ: Prentice-Hall, 1973.
8. Cohen, P., and E. Feigenbaum. The Handbook of Artificial Intelligence, Vol. 3. CA: William Kaufmann, 1982.
9. Cristofides, N. Graph Theory: An Algorithmic Approach. New York: Academic Press, 1975.
10. Cross, S. E. "Qualitative Sensitivity Analysis: A New Approach to Expert System Plan Justification," Proc. of the Canadian Society for Computational Studies of Intelligence, pp. 138-140, London, Ont., May 1984.
11. Cross, S. E. "Requirements of a Flight Domain Expert System Architecture," Proc. Int. Conf. on System Engineering, pp. 250-255, Dayton OH, September 1984.

12. Cross, S., R. Bahnij, and D. Norman. "Knowledge-based Pilot Aids: A Case Study in Mission Planning," Symposium in Artificial Intelligence, German Aerospace Research Establishment, Bonn, W. Germany, May 1986.
13. Defense Advanced Research Projects Agency (DARPA). Strategic Computing, AD-A141 282/9, Arlington VA, 1983.
14. Even, S. Graph Algorithms. Rockville MA: Computer Science Press, 1979.
15. Flynn, M. "Very High-speed Computing Systems," Proc. IEEE, 54: 1901-1909 (December 1966).
16. Galil, Z. and W. Paul. "An Efficient General-Purpose Parallel Computer," J. ACM, 30: 2 (1983).
17. Geddes, N. "Intent Inferencing Using Scripts and Plans," Proc. Aerospace Applications of Artificial Intelligence, Dayton OH, 1985.
18. Hayes, J. Computer Architecture and Organization. New York: McGraw-Hill, 1978.
19. Hillier, F. and G. Lieberman. Operations Research (Second Edition). CA: Holden-Day, 1974.
20. Horowitz, E. and S. Sahni. Fundamentals of Computer Algorithms. MD: Computer Science Press, 1978.
21. Hudlicka, E. and V. Lesser. "Construction and Use of a Causal Model of a Problem-solving System," Proc. National Artificial Intelligence Conf., 1984.
22. Johanssen, G., and W. Rouse. "Studies of Planning Behavior of Aircraft Pilots in Normal, Abnormal, and Emergency Situations," IEEE Transactions on Systems, Man, and Cybernetics, SMC-13: 267-278 (May/June 1983).
23. Knode, D. An Approach to Planning in the Inflight Emergency Domain. MS thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1984.
24. Lesser, V. "Overview of Important Issues in Distributed Problem Solving." Invited talk, Conf. on Aerospace Applications of Artificial Intelligence, Dayton OH, October 1985.

25. Manna, Z. Mathematical Theory of Computation. New York: McGraw-Hill, 1974.
26. Minsky, M. "Society Theory of Thinking," Artificial Intelligence: An MIT Perspective. P. Winston, R. Brown, eds. MA: MIT Press, 1979.
27. Nilsson, N. Principles of Artificial Intelligence. CA: Palo-Alto, 1980.
28. O'Reilly, C. A., and A. S. Cromarty. "'Fast' is not 'real-time': Designing Effective Real-time AI systems," Proc. SPIE, Applications of AI II. Arlington VA, 1985.
29. Peterson, J. Petri-net Theory and the Modeling of Systems. Englewood Cliffs NJ: Prentice-Hall, 1983.
30. Piper Aircraft Corp. Warrior II Information Manual. Report VB-1180, August 1982.
31. Ratelle, R., and R. Morishige. "Air Combat and Artificial Intelligence," Air Force Magazine, 68: 10, 90-93 (October 1985).
32. Rich, E. Artificial Intelligence. New York: McGraw-Hill, 1983.
33. Schank, R., and R. Abelson. Scripts, Plans, Goals, and Understanding. New York: J. Wiley and Sons, 1977.
34. Shapiro, E. "Systolic Programming: A Paradigm of Parallel Processing," Proc. Int. Conf. on Fifth Generation Computer Systems, 1984.
35. Shortliffe, E. Computer-based Medical Consultations: MYCIN. New York: North Holland, 1976.
36. Stefik, M. Planning with Constraints. Report No. 784. Computer Science Department, Stanford University, 1980.
37. Stein, K. J. "DARPA Stressing Development of Pilot Associate System," Aviation Week and Space Technology, pp. 69-74, 22 April 1985.
38. Sussman, R., and R. Stallman. "Forward Reasoning and Dependency Directed Backtracking in a System for Computer-aided Circuit Analysis," Artificial Intelligence, 9: 135-196 (1977).

39. Waltz, D. Generating Semantic Descriptions from Drawings of Scenes with Shadows. AI-TR-271, MIT, 1972.
40. Watson, C. "A Research Proposal for a Fifth Generation Computer Design," PARSYM Digest, 1(1) (1985).
41. Wilensky, R. Planning and Understanding. Reading MA: Addison-Wesley, 1983.
42. Williamson, S. Combinatorics for Computer Science. Rockville MD: Computer Science Press, 1985.
43. Winston, P. Artificial Intelligence (Second Edition). Reading MA: Addison-Wesley, 1984.

VITA

Capt Douglas O. Norman was born on 27 Nov 51 to Owen and Pat Norman at the St. Albans Naval Hospital in Queens, NY. The oldest of three boys born to the Normans, he was raised in Farmingdale, NY where he graduated from the Farmingdale High School in May of 1969.

Following his graduation from high school, Capt Norman enlisted in the United States Coast Guard. He was assigned as a paramedic air-crew member at Otis AFB, MA where he was awarded nine "Winged S" for life-saving missions in Sikorsky HH-3F and HH-52 helicopters.

After his tour of active duty, which ended in 1974, Capt Norman entered the State University of New York at Stony Brook where he received a BS in Biological Sciences in 1976. He entered graduate school, also at SUNY Stony Brook, and completed 60 graduate credits in the Neurobiology and Behavior program. He published a number of scientific papers during this period.

In 1978 Capt Norman began working as a commercial fisherman. After a short time, he and his wife purchased their own fishing vessel and started their own fishing business.

Capt Norman came on active duty with the USAF in 1980 and was assigned as a programmer at the Defense Nuclear

Agency's Armed Forces Radiobiology Research Institute (AFRRI). AFRRI is DoD's research arm for investigating and teaching about the biological effects of ionizing radiation. During his tenure at AFRRI, Capt Norman assumed the duties as AFRRI's Computer System Manager in addition to programming duties.

Following his tour at AFRRI, Capt Norman was assigned to AFIT to pursue a master's degree in computer systems. He completed the Artificial Intelligence sequence and the Theory of Computation sequence in addition to the computer systems requirements.

Capt Norman completed Squadron Officer's School in residence, and was voted "most valuable section contributor" by his classmates.

Capt Norman married Janice Conde in 1972. They have two children, Michael and Joseph.

Permanent address: 11 Sugar Bush Lane
Coram, New York

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A163947

REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1d. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/85b-12			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/EN	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, OH, 45433			7b. ADDRESS (City, State and ZIP Code)		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Pilot Associate Office		8b. OFFICE SYMBOL (If applicable) AFWAL/CCU	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, OH, 45433			10. SOURCE OF FUNDING NOS.		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
11. TITLE (Include Security Classification) See box 19			WORK UNIT NO.		
12. PERSONAL AUTHOR(S) Douglas G. Norman, Capt, USAF					
13a. TYPE OF REPORT Thesis		13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1985 December		15. PAGE COUNT 146
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB. GR.	Artificial Intelligence Pilots Planning		
09	02		Parallel Processing Real Time		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
Title: REASONING IN REAL TIME FOR THE PILOT ASSOCIATE: AN EXAMINATION OF A MODEL BASED APPROACH TO REASONING IN REAL-TIME FOR ARTIFICIAL INTELLIGENCE SYSTEMS USING A DISTRIBUTED ARCHITECTURE					
Thesis Advisor: Stephen E. Cross, Capt, USAF					
Abstract on back					
20. DISTRIBUTION AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Stephen E. Cross, Capt, USAF			22b. TELEPHONE NUMBER (Include Area Code) (513) 255-7776	22c. OFFICE SYMBOL AFIT/ENG	

END

FILMED

3-86

DTIC