

AD-A218 717

DTIC FILE COPY

4

RADC-TR-89-259, Vol IV (of twelve)
Interim Report
October 1989



NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Distributed AI for Communications Network Management

Syracuse University

Robert A. Meyer and Susan E. Conry

DTIC
ELECTE
MAR 01 1990
S B D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded partially by the Laboratory Director's fund.

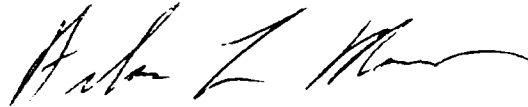
ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 02 28 048

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-259, Vol IV (of twelve) has been reviewed and is approved for publication.

APPROVED:



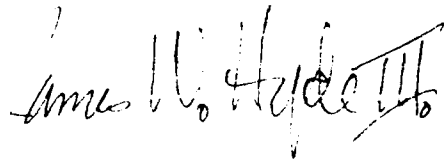
ARLAN L. MORSE, Capt, USAF
Project Engineer

APPROVED:



JOHN A. GRANIERO
Technical Director
Directorate of Communications

FOR THE COMMANDER:



JAMES W. HYDE III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS N/A			
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A		5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-259, Vol IV (of twelve)			
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC)		6b. OFFICE SYMBOL (if applicable)		7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)	
6c. ADDRESS (City, State, and ZIP Code) Science & Technology Center, Rm 2-296 111 College Place, Syracuse University Syracuse NY 13244-4100		7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (if applicable) COES		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008	
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO 62702F	PROJECT NO 5581	TASK NO 27	WORK UNIT ACCESSION NO 13
11. TITLE (Include Security Classification) NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Distributed AI for Communications Network Management					
12. PERSONAL AUTHOR(S) Robert A. Meyer, Susan E. Conry					
13a. TYPE OF REPORT Interim		13b. TIME COVERED FROM Jan 88 TO Dec 88		14. DATE OF REPORT (Year, Month, Day) October 1989	
15. PAGE COUNT 96					
16. SUPPLEMENTARY NOTATION This effort was funded partially by the Laboratory Directors' Fund. This effort was performed as a subcontract by Clarkson University to Syracuse University, Office of Sponsored Programs.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Distributed Artificial Intelligence, Distributed Planning, Knowledge-based Reasoning, Simulation, Communications Network, Graphical User Interface		
12	05				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the fourth year of the existence of the NAIC on the technical research tasks undertaken at the member univer- sities. The topics covered in general are: versatile expert system for equipment mainten- ance, distributed AI for communications system control, automatic photointerpretation, time- oriented problem solving, speech understanding systems, knowledge-based reasoning and planning, knowledge base maintenance, hardware architectures for very large systems, and a knowledge acquisition, assistance, and explanation system. The specific topic for this volume is the use of knowledge-based systems for communications network management and control via an architecture for a diversely distributed multi-agent system.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Arlan L. Morse, Capt, USAF			22b. TELEPHONE (Include Area Code) (315) 330-7751		22c. OFFICE SYMBOL RADC (DCLD)

DD Form 1473, 10/88

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

Item 10. SOURCE OF FUNDING NUMBERS (Continued)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	5581	27	23
61102F	2304	J5	01
61102F	2304	J5	15
33126F	2155	02	10
61101F	LDFP	27	01

UNCLASSIFIED

NAIC

Northeast Artificial Intelligence Consortium

1988 Annual Report

Volume 4

Distributed Artificial Intelligence For
Communications Network Management

Robert A. Meyer and Susan E. Conry
Electrical and Computer Engineering Department
Clarkson University
Potsdam, New York 13676

Contents

4.1	Executive Summary	5
4.2	Introduction	7
4.3	Role Recognition in Multiagent Distributed Planning	9
4.3.1	Problem Description	9
4.3.2	Related Work	10
4.3.3	Planning as a Distributed Resource Allocation Problem	13
4.3.4	Distributed Plan Generation	17
4.3.5	Example: Generation of Service Restoral Plans	20
4.3.6	Experimental Results	25
4.3.7	Future Directions	38
4.4	Cooperation Using Constraint-Based Reasoning	39
4.4.1	Abstraction of Constraints and Conflicts	39
4.4.2	Computation of Conflict	46
4.4.3	Status and Future Directions	48
4.5	Maintaining Consistent Beliefs in a Shared Knowledge Base	48
4.5.1	The Role of Truth Maintenance	49
4.5.2	Comparisons with Existing Truth Maintenance Systems	52
4.5.3	Functional Design of The MATMS	54
4.5.4	The MATMS Interface	70
4.5.5	Implementation for Communications Network Management	75
4.5.6	Areas for Future Research	84

Accession for	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

List of Figures

1	Global Perspective of a Complication	16
2	Limited View of Multiple Requests	16
3	Multiple Plans That Locally Appear to be One	17
4	Multiple Participation of a Single Agent	19
5	Example Network	21
6	Line Topology	27
7	Ring Topology	28
8	Tightly Coupled Topology	28
9	Experimental Results: Line Topology	29
10	Experimental Results: Line Topology	30
11	Experimental Results: Line Topology	31
12	Experimental Results: Ring Topology	32
13	Experimental Results: Ring Topology	33
14	Experimental Results: Ring Topology	34
15	Experimental Results: Tightly Coupled Topology	35
16	Experimental Results: Tightly Coupled Topology	36
17	Experimental Results: Tightly Coupled Topology	37
18	Net Connecting MATMS Frames	57
19	Inference Tree for Example Illustrating Data Structures	58
20	Decision Tree for <i>Problem Solver Proposes Adding Assumption</i>	61
21	Decision Tree for <i>Problem Solver Proposes Removing Assumption</i>	62
22	Decision Tree for <i>Problem Solver Proposes Inference</i>	63
23	Inference Tree for <i>Inference Replaces Assumption Example</i>	66
24	Inference Tree for Example Illustrating Multiple Derivations	67
25	Decision Tree for <i>Problem Solver Proposes Retracting Justification</i>	68
26	Proposed Interagent Communications Paths	74
27	Sample Communications Network	75
28	Sample Equipment Configuration	76
29	Sample Trunk and Circuit Configuration	77

30	Communications Network Knowledge-Based System Architecture	78
31	Communications Network for Example 2	81

List of Tables

1	Local Resource Control	21
2	Time Slice View of Example, T1-T7	23
3	Time Slice View of Example, T8-T10	24
4	Results of Plan Generation Example	24
5	Global Plans Generated	40
6	Local Knowledge About Plan Fragments	41
7	Example Data Structures	59
8	Data Structures for <i>Inference Replaces Assumption</i> Example	66
9	Data Structures for Example Illustrating Multiple Derivations	68
10	Data Structures for Example 2	83

4.1 Executive Summary

This volume describes work done during 1988 at Clarkson University on the task, Distributed Artificial Intelligence For Communications Network Management, of the NAIC research contract with the Rome Air Development Center. The objective of this effort is to investigate potential applications of distributed AI to system control and network management problems for large-scale, world-wide communications networks. This report includes a brief summary of the typical context in which these problems arise, and outlines the architecture we have developed for application of AI technology to these problems. We have identified specific issues which raise fundamental research questions to be resolved in order to bring the application of AI technology to full fruition in this area. The central focus of our work has been to study these questions, proposing answers, and testing the merits of these answers in a realistic simulated environment. We have results, based on implementations which have been tested in our testbed environment, in three important areas. These areas are: multiagent distributed plan generation, constraint-based reasoning in multiagent planning, and multiagent truth maintenance for shared knowledge bases.

The problem domain which forms the context for our work is the management and control of a large scale, world-wide communication system such as the U.S. Defense Communication System (DCS). We have concentrated on network management and control at the subregion level. The subregion level represents a group of ten to twenty individual sites or nodes in the communications system architecture which are monitored and controlled from a single control center. System-wide management and control is distributed over a network of subregion control centers, typically eight to twelve in number. Our view of the role of distributed AI in this environment is to provide cooperating, intelligent, semi-autonomous agents to serve as problem solving assistants to the human controllers. This set of agents must be distributed both spatially and functionally. The spatial distribution is a natural consequence of the underlying communications network and control system architecture which is distributed over a large geographical area. The functional distribution arises from the requirement for multiple, distinct, but related, problem solving tasks in performing network management. These tasks are: performance assessment (PA), fault diagnosis or isolation (FI), and service restoral (SR). *Our design for this system incorporates new ideas in distributed problem solving: specifically, a diversely distributed problem solving architecture which supports coordination and cooperation among functionally and spatially distinct agents.* During this past year we have devoted most of our efforts to the service restoral task, and to developing a basis for local multiagent cooperation using a shared knowledge base.

The service restoral task requires distributed planning subject to constraints imposed by network topology and resource availability. We have developed a distributed planner

which extends the current work in planning by designating certain objects as resources so that they may be efficiently allocated for effective use in multiple goals. The planner consists of two stages, plan generation and multistage negotiation. During plan generation, agents are required to generate plans which utilize limited system resources in a domain where both the knowledge about resources and the control over these resources are distributed among the agents. After a set of plans has been established, agents must cooperatively select specific plans to execute as many goals as possible, subject to resource constraints. Multistage negotiation has been developed as a means by which an agent can acquire enough knowledge to reason about the impact of local decisions on nonlocal system state and modify its behavior accordingly.

As a result of the incomplete knowledge and distributed control agents have, plan decomposition is dynamic and diffuse in nature. To effectively plan using such decompositions, each agent must be capable of determining its role in multiple plan decompositions with only a partial view of the associated global plans. We have introduced the concept of support names as a means to allow each agent to recognize which sets of local actions are required by various plan decompositions for the same goal. The use of support names has been implemented in a system which generates plans for the restoral of service in a communications network. Experimental results are presented which show that plan generation in this class of problems can be accomplished by sending a limited amount of information between agents. It is unnecessary for any single agent to acquire complete global information about the system.

Because no single agent is in control and no single agent has complete knowledge of the entire system state, an important aspect of multistage negotiation is the mechanism for providing agents with nonlocal information. We have developed a formalism for abstracting and propagating information about the nonlocal impact of decisions made locally. Our work provides mechanisms for determining impact at three levels: locally on the level of plan fragments, locally on the level of goals, and nonlocally. This approach may be viewed as promoting cooperation among agents by using constraint-based reasoning to develop good, local heuristic decision making. This phase of our work is currently in the theory development stage, and will be a major thrust for implementation in the next year.

A common method by which problem solving agents cooperate involves sharing knowledge. We present a method by which knowledge can be shared in a local knowledge base in the form of inferences and default assumptions. Specifically, a truth maintenance system, MATMS (Multiagent Assumption-based Truth Maintenance System), has been developed to manage a knowledge base shared by multiple problem solvers. The most important feature of the MATMS is that it provides the foundation for resolving inconsistency between agents, while supporting the notion that two problem solvers can have different views concerning the state of a particular piece of knowledge. The MATMS handles differing views by allowing independent belief sets for each of the agents. It supports resolving inconsistency between agents by providing a mechanism for comparing two agents' belief sets. An

agent's belief set is characterized as the default knowledge base (which is common to all agents) with an overlay placed upon it. The MATMS is efficient largely because it focuses its efforts on managing these overlays, not the entire belief set of an agent. By concerning itself only with the overlays, the MATMS can switch from addressing one problem solver's belief set to addressing another's expeditiously. It can also change an individual problem solver's belief set quickly, because the default knowledge is not explicitly carried over from one belief set to another. We have implemented and tested MATMS in the context of a distributed knowledge base system for managing a communications network.

4.2 Introduction

In our investigation of distributed AI for communications network management, we have focused on the European Theater of the Defense Communication System (DCS). The DCS is a large, complex communications system consisting of many component subsystems. It provides the long-haul, point-to-point, and switched network communications for the Department of Defense. We have chosen the European Theater because the DCS network structure in Europe is particularly interesting for the study of distributed problem solving paradigms. It consists of a large number of sites (about 200) which are interconnected in an irregular structure. It is currently controlled by close cooperation and coordination among a group of highly skilled human controllers distributed throughout the system. The control task is one which requires extensive, specialized knowledge and the ability to reason using this knowledge in solving problems. In the past, system control has been a difficult area to automate because the number of situations which may arise and alternative solutions available are very large, and thus traditional, purely algorithmic approaches have been found lacking. There is a clear requirement for sophisticated problem solving tools to assist these human operators in providing the best possible control of the system.

The reason for studying distributed problem solving lies in the observation that humans often rely on teams of people to solve complex problems. Within a team there is usually a division of labor so that each member of the team is a specialist on some part of the problem. Each of these specialists has only a limited perspective about the overall problem, and each finds that he can only deal with those aspects of the problem for which he is responsible through cooperation with others on the team.

Distributed artificial intelligence is concerned with problems that arise when a group of loosely coupled problem solving agents works together to solve a problem. These agents have characteristics that closely parallel those mentioned above: functional specialization, local perspective, and incomplete knowledge. Each agent uses its own local perspective in performing its tasks, though it may have a need for some knowledge about another agent's local perspective.

We have developed an architecture for a diversely distributed, multi-agent system in which each component is a specialized and localized knowledge-based system designed to

provide assistance to the human operator. Each agent must be able to cooperate with similar agents performing other functions at the same local site as well as cooperating with identical agents located in physically separate facilities. This view of the role of a knowledge-based system as a collection of autonomous, cooperating independent specialists is an important characteristic of our approach to network management and system control for the DCS in the future.

We have found three fundamental kinds of problem solving activities required: (1) data interpretation and situation assessment; (2) diagnosis and fault isolation; and (3) planning to find and allocate scarce resources for restoral of service in the event of an outage. In addition to this functional distribution of problem solving activities, our model requires a spatial distribution of control as well. We present an architecture designed to meet these requirements which consists of a distributed knowledge-based system built on a community of problem solving agents. Each agent is a functionally specialized knowledge-based problem solver at a specific site. These agents coordinate and cooperate to solve global problems among themselves, crossing functional or spatial boundaries as required.

At a local level, the system is seen as a number of functionally specialized agents that cooperate in a loosely coupled fashion. These agents comprise a local participant in a network-wide team of problem solvers. At the global level, the system may be viewed as a group of relatively independent, spatially distributed problem solving systems cooperating to solve a collection of problems.

An important feature of the system is the cooperation of the agents. Cooperation at the local level is by two methods. First, problem solvers cooperate by coordinating their actions. An agent may request another to perform some task to further the overall problem solving. This is achieved through an exchange of messages. The other mechanism for cooperation is through sharing knowledge concerning the current state of the communications network. Inferences of one agent are shared with the others, in a central knowledge base. The shared knowledge base is managed by the Knowledge Base Manager (KBM).

At the present time a testbed has been implemented which supports simulation of multiple agents on one or more physically distinct Lisp processors. Detailed design and implementation of specific agents has been our major activity for this year. We have developed agents which cooperate across the spatial dimension, and we have developed the tools for managing local cooperation using the shared knowledge base concept. In the remaining sections of this volume, we describe the three areas in which we have made substantial progress this year. First, we have developed a cooperation mechanism which allows an agent to recognize its specific, localized role in generating global plans. Second, we have formalisms which allow a group of agents to understand how local decisions impact the ability of the group to achieve global goals when selecting alternative plans subject to constraints. Third, we have implemented an efficient local manager of a shared knowledge base.

4.3 Role Recognition in Multiagent Distributed Planning

In this section we describe a *distributed* planner which extends the current work in planning by designating certain objects as resources so that they may be efficiently allocated for effective use in multiple goals. The planning strategy consists of two stages, plan generation and multistage negotiation. The plan generation phase is the focus of the work presented in this section. During plan generation, agents are required to generate plans which utilize limited system resources in domains where both the knowledge about resources and the control over these resources are distributed among the agents. As a result, plan decomposition is dynamic and diffuse in nature. To effectively plan using such decompositions, each agent must be capable of determining its role in multiple plan decompositions with only a partial view of the associated global plans. *Support names* are introduced as a means to allow each agent to recognize which sets of local actions are required by various plan decompositions for the same goal. The use of support names has been implemented in a system which generates plans for the restoration of service in a communications network. Results from experimentation are presented which show that plan generation in this class of problems can be accomplished by sending a limited amount of information between agents. It is unnecessary for any single agent to acquire complete global information about the system.

4.3.1 Problem Description

When working with large systems, it is desirable to distribute system information among several problem solving agents. Planning in these distributed domains [21] is distinguished from conventional planning in that plans are composed of sub-plans or plan fragments, each of which represents a solution to a subproblem that is executed by some agent in a multiple agent system. Most of the systems that address planning for distributed domains have assumed that there is a single active planner. This planner knows the capabilities of each agent present in the system and is charged with the responsibility of generating a multiagent plan.

We are concerned with planning for the efficient allocation of distributed resources in multiagent systems where multiple goals coexist. The domains involved are quite large, making it difficult if not impossible to maintain complete, detailed and accurate information about system resources at each agent. Therefore, planning is a process that must be carried out by a group of semi-autonomous agents, each of which has a limited view of the global system state, has control over only a subset of the resources required to satisfy a goal, and has only partial knowledge about the complete set of resources needed and who controls them. We have developed a planner which operates in two phases, plan generation and multistage negotiation [3]. The plan generation phase determines multiple global plans for satisfaction of each system goal. Multistage negotiation then attempts to determine a set of alternative plans that satisfies the maximum number of global goals.

subject to resource constraints. The focus of this presentation is the plan generation phase of the planner.

Our planner is a **distributed planner** as opposed to a centralized planner for a multiagent system. Due to the distribution of both the knowledge about system resources and their control, plan generation is dynamic and diffuse in nature. Consequently, a single agent may be asked multiple times to aid in the construction of a global plan. As a result, each agent must be able to determine how multiple requests for partial satisfaction of a single goal fit into that agent's distinct alternatives to satisfy the goal. Is every request part of the same plan? Which requests are actually part of the same alternative and which are the results of distinct plan decompositions? Agents must be able to determine their role in multiple plan decompositions for a single goal with only a limited view of these plans.

Support names are introduced as a means by which each agent can recognize how it participates in various plan decompositions for the same goal without forming global views of these plans. Support names are part of an incremental tagging procedure which allows each agent to recognize its actions in various plan decompositions for the same goal. Agents use support names to construct a limited, abstract view of global information, but no agent is ever provided with complete detailed global information. This is partly due to the maintenance problem described earlier, but more importantly it is *unnecessary* when using our planner.

Plan generation using support names has been implemented in the domain of communications network management. In this system, plan generation determines plans for the restoral of communication service between two users. Through experimentation, we show that plan generation can be accomplished by passing merely a limited amount of information among system agents. All that is required are descriptions of the goal state and the present state of the plan, and information which allows agents to recognize their past actions in the construction of the plan. This last piece of information is provided through the use of support names. Experimental results show that this distributed mechanism is much faster than a centralized method which provides a single agent with complete details of the entire system. In addition, it does not require the construction of such a global view by any agent.

4.3.2 Related Work

4.3.2.1 Comparison with Research in Planning Our planner differs from existing planners in that certain objects are designated as resources so that they may be efficiently allocated for satisfaction of multiple system goals. Thus, our work represents an extension of the current use of resources in planning. SIPE [38] is a planner which was explicitly designed to handle the notion of resources. However, SIPE uses objects designated as resources for early detection of subgoal interactions among subgoals for the *same* goal.

There is no provision made for efficient allocation of resources to satisfy multiple goals. In addition, SIPE is designed for centralized planning, therefore the planner has complete knowledge about the available system resources. It should also be noted that SIPE does allow for the temporary allocation of a resource in the execution of a plan. This is an extension planned for the model presented in this report.

The notion of dividing a large problem, and thus a large search space, into smaller problems resulting in regional search spaces is very similar to the planning strategy of GEMPLAN [22]. GEMPLAN is a general constraint satisfaction system which exploits any inherent structure in a given problem domain. The structure of the user's chosen domain is used to identify entities which can be defined as regions where localized constraints can direct a localized search. This structure can take on a hierarchical form whereby one region has access to the subplans of all its subregions. Thus, local search can be used to satisfy local constraints and in fact it may be possible for several subregions to perform their local searches in parallel. Then, moving back up to the regional search space, a regional search can be performed to satisfy regional constraints. Our use of multiple agents with detailed local information is similar to identifying regions by local constraints. However, our planner does not rely upon a higher level or global search space which can use the results of the local searches to satisfy global constraints.

Systems which perform distributed task decomposition in multiple agents systems do exist. The Contract Net protocol [35] and the distributed NOAH system [5] are perhaps the most well known. The Contract Net protocol performs well in domains where the task can be divided into nearly independent subtasks. Such a decomposition does not require that global information be passed among the agents since interactions among the subtasks are assumed to be nonexistent or unimportant. Thus, no provision is made that allows an agent to reason about multiple participations in the construction of a single plan. Furthermore, for domains such as those described in this report, there is no means by which an agent can reason about its participation in *multiple* plan decompositions. The distributed NOAH system does provide mechanisms for an agent to reason about multiple participation in the construction of a single plan. However these mechanisms require complete and accurate information concerning the global plan to be resident at each agent in the system.

Other notable work in Distributed Artificial Intelligence include the research efforts of Durfee and Lesser [11], Rosenschein and Genesereth [34], Georgeff [17], and Cammarata, McArthur and Steeb [2]. The work of these researchers is related to the multistage negotiation phase of our planner.

4.3.2.2 Comparison with Routing Algorithms In our domain implementation we refer to the function of planning new routes for disrupted circuits as Service Restoral. At first glance, it may appear that Service Restoral performs standard routing of disrupted circuits. The problem of routing circuits is well understood and indeed, many

algorithms exist which route circuits in distributed networks. However, the assumptions concerning node connections as well as the overall objective of Service Restoral differs from those of the conventional algorithms. It is these differences which make existing algorithms inappropriate for Service Restoral. In the following paragraphs, several conventional algorithms are described briefly along with their assumptions and objectives. This description is then contrasted with the assumptions and purpose of Service Restoral.

Many distributed routing algorithms have been developed as a result of computer network construction. Most can be grouped into classes by their basic approach to the problem [16].

One class depends upon global knowledge residing at each processor node in the system [29] and the use of some graph algorithm [9]. Another class of existing routing algorithms requires only condensed information at each node and uses "preferred next neighbor" tables to designate the next node in the shortest path to every possible destination in the network [14, 28, 20, 37, 36]. An improvement to this class lead to the formation of another set of algorithms [15, 19, 30] which also require only partial topology information and rely on "preferred next neighbor" tables but these algorithms establish and maintain the shortest path between two nodes through strict control of how the routing tables are updated. Another algorithm which has gained recognition is one that uses a saturation technique [23] whereby each node only needs to know its nearest neighbors and the trunk groups associated with each search message.

Each of these algorithms makes the same assumption about node connectivity, namely that every trunk into a node connects to the same main switching device. Given this assumption, it is senseless for a proposed route to pass through a node more than once. Each of these algorithms prevents, or attempts to prevent, the existence of such a route. It is also important to emphasize that each of the algorithms routes one circuit at a time with the goal of finding the route with minimal cost. The purpose of these algorithms is to dynamically route temporary circuits. Cost factors in these algorithms usually include at least the length of the route and sometimes the current demands upon the trunks traversed. This overall philosophy is applied to each circuit in isolation. That is, if more than one circuit needs to be routed, these circuits are routed without regard for their mutual existence.

In contrast, Service Restoral does not make the same assumption about the existence of a central switching device at each processor node. Service Restoral acts with a coarse grained level of processor distribution. Instead of a processor residing at each individual site, a processor is responsible for several sites, where each of these sites belongs to the same subregion. As a consequence, a processing node corresponds to a subregion and thus intrasubregion connectivity becomes an important issue. Service Restoral must work in an environment where multiple paths exist through the subregion which visit disjoint sets of sites. Thus it is possible that a plausible restoral route for a circuit may pass through a node more than once. In fact, a plausible restoral route may pass through multiple nodes

more than once depending upon inter- and intranode (subregion) connectivity.

Another reason the conventional distributed routing algorithms are inappropriate for Service Restoral is that the overall philosophies differ. Whereas the existing algorithms are intended for temporary routing of previously nonexistent circuits, Service Restoral reroutes existing dedicated circuits which have been disrupted. Service Restoral attempts to make the most effective use of the network resources so as to restore as many circuits as possible. This is accomplished by collectively restoring circuits, making use of their previously reserved trunks as well as spare trunks and trunks that may be preempted. In order to determine the best utilization of the network resources, multiple alternatives for restoring each disrupted circuit must be generated. From these alternatives, Service Restoral selects those plans which collectively restore the greatest number of circuits. Thus, merely determining the shortest or minimal cost path for a circuit is not the aim of Service Restoral. In fact, such a route may actually prevent the restoral of other circuits in the system. Instead, it is the intention of Service Restoral to move away from the dogmatic procedure of routing circuits in isolation to an approach which utilizes perceptions of what's happening in the system as a whole. By being aware of a group of circuits that have been disrupted, Service Restoral will be able to accurately reallocate network resources in the most effective manner.

The following section describes the characteristics of distributed planning in more detail and defines the requirements of the plan generation phase. Section 4.3.4 presents techniques by which the requirements of plan generation are met. This is followed in Section 4.3.5 by an example taken from the domain of communications network management. In Section 4.3.6, experimental results comparing distributed plan generation to other plan generation strategies are presented. Then in Section 4.3.7 we describe extensions to distributed planning that should be addressed as research continues.

4.3.3 Planning as a Distributed Resource Allocation Problem

4.3.3.1 Problem Class Description In many multiagent domains, planning can be viewed as a form of distributed resource allocation problem in which actions require resources in order to satisfy system goals. In such domains, goals require allocation of distributed system resources, but criteria for goal satisfaction are not specified by enumerating the resources required. In fact, the resources required are not known at the time of the instantiation of a goal but are determined as a plan for satisfaction of that goal is constructed. In addition, it is usually the case that there are several combinations of system resources which could be used to satisfy a single goal depending upon the local actions taken by system agents.

The resources which are involved in this class of problems are assumed to be indivisible (not consisting of component resources). Their supply is limited and they cannot be time shared for concurrent satisfaction of multiple goals.

Allocation of a resource in this context has several ramifications. Once a resource is allocated for partial satisfaction of a goal, it is in use as long as this goal is being satisfied. Furthermore, a goal is satisfied only if each and every one of the required resources is currently allocated for this goal. As a result, if even one of these resources is allocated for some other purpose, the original goal is no longer satisfied. Thus, execution of a plan to satisfy a goal implies the concurrent allocation of distributed resources. Moreover, allocation of a resource does not imply consumption of that resource. In fact, it is more than likely that the same resource will be allocated for different purposes at different times as the needs of the system change.

Problems of the type addressed in this report are also characterized by their large size. That is, these problems involve vast amounts of detailed information. In addition, this information is constantly being modified. Maintaining a complete, accurate, and detailed view of such a large, dynamic system at each agent is obviously difficult if not impossible. For these reasons it is desirable to limit each agent's view of the entire system. Therefore, control over resources and knowledge about these resources are distributed among problem solving agents. Some of the resources are under the direct control of a single agent, while control over others is *shared* by two agents. Allocation of a shared resource requires coordination between the agents that share its control. In addition, agents have a limited view of the resources that are not under their direct control. Thus, no single agent has complete knowledge about what resources exist in the system or who controls them. As planning progresses, agents do construct abstract views of global information but they never form detailed pictures of global state. This is partly due to the maintenance problem, but more importantly it is *unnecessary* when using the planner described in this report.

4.3.3.2 Requirements of Plan Generation As stated previously, the overall objective of the planner presented is to efficiently allocate system resources so that as many global goals as possible are concurrently satisfied in a multiagent domain.

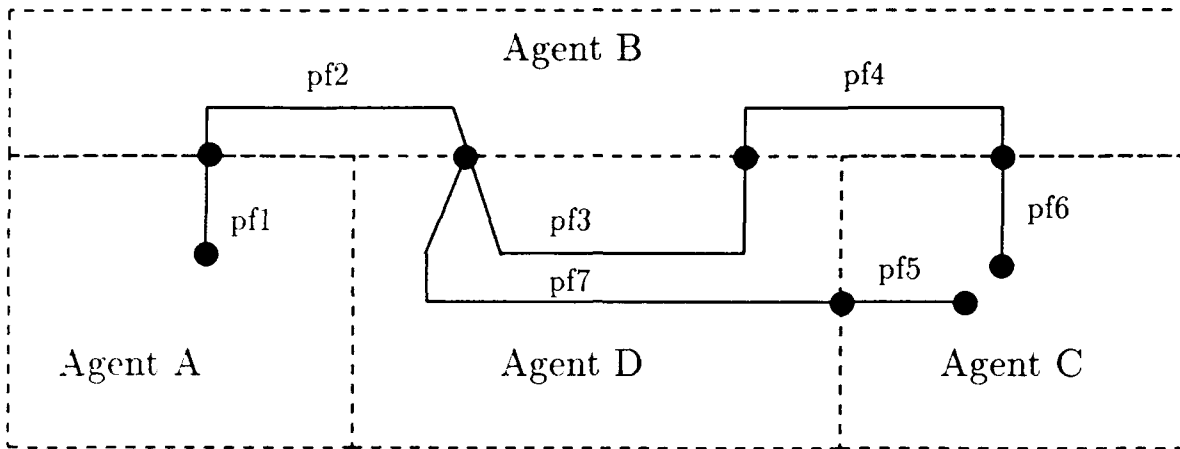
Distributed plan generation is difficult because each agent must have the ability to recognize its role in global plan decompositions with only a partial view of the global plans in which it participates. As in multiagent systems using a centralized planner, generating an acceptable plan requires problem decomposition and assignment of subtasks to different agents in the system. The major difference lies in the character of the decomposition. In **distributed** planning the decomposition is *dynamic*, with each agent determining the extent to which it can contribute to satisfaction of a subgoal. Given that contribution, the agent must then determine which other agents may be able to aid in the completion of the plan. The decomposition is also *diffused* in that no agent has knowledge of the entire system state, the entire goal-subgoal structure, or a complete view of any of the multiple plan decompositions currently under construction. This is due to the fact that both knowledge and control of system resources is distributed among the agents. Using the

goal description, agents must dynamically determine what combinations of their resources will satisfy this goal. However, this must be accomplished without any single agent having complete knowledge of what resources exist or who controls them.

A complication arises as a consequence of the dynamic and diffuse nature of plan decomposition. Specifically, a particular agent may be asked to contribute at different times and in different ways to the satisfaction of many subgoals relative to the satisfaction of a single global goal. In order for an agent to correctly determine which of its alternatives can satisfy a particular subgoal, it must be able to assess which subgoals are part of the same global plan decomposition and which are part of distinct decompositions. Those alternatives that are part of the same global plan must be combined into a single alternative. This is necessary for the success of multistage negotiation. During multistage negotiation, agents must be able to reason about the impact of selecting an alternative. In particular, an agent must be able to determine how choosing one plan fragment to satisfy one local subgoal will effect its ability to select plan fragments to satisfy other local subgoals. Execution of a particular plan fragment necessarily limits the resources available for use in satisfaction of other subgoals. Therefore, selection of any specific alternative has potential side effects on the agent's ability to participate in satisfaction of additional global goals. Reasoning about such subgoal interactions can only occur if each available alternative is part of a distinct global plan.

To clarify this point consider Figure 1 which depicts a four agent system in which two global plans have been constructed for satisfaction of a single goal. The plans are presented pictorially as sequences of plan fragments distributed among the planning agents. Each plan fragment can be assumed to be a set of local actions which achieve part of the global goal. In particular, notice Agent B's participation in the plan generation phase. Agent B has received two requests to participate in plan construction for this goal. Using the global view presented by Figure 1, it is obvious to discern that Agent B should use pf2 as one of its distinct alternatives and pf2 and pf4 should be combined into a second distinct alternative for satisfaction of this goal. However, it is important to realize that Agent B *does not* have this global picture. What Agent B "sees" is shown in Figure 2. Yet, Agent B must be able to determine how these two requests fit into its distinct alternatives for this goal. Therefore, plan generation must provide some additional information so that agents may formulate distinct alternatives with only a limited view of the multiple plan decompositions created.

It should be noted that when an alternative available to an agent appears to be part of a distinct global plan, this is a local perspective. In fact, this alternative may be part of several global plans for the same goal. However, if the agent's participation in each global plan is the same, they locally appear to be a single plan. To clarify this point, consider Figure 3 which shows two global plans for a single goal. As before, the plans are presented pictorially as sequences of plan fragments distributed among four planning agents. Once again, notice Agent B's participation in each of the global plans. At the



Plan 1 : pf1-pf2-pf3-pf4-pf6

Plan 2 : pf1-pf2-pf7-pf5

Figure 1: Global Perspective of a Complication

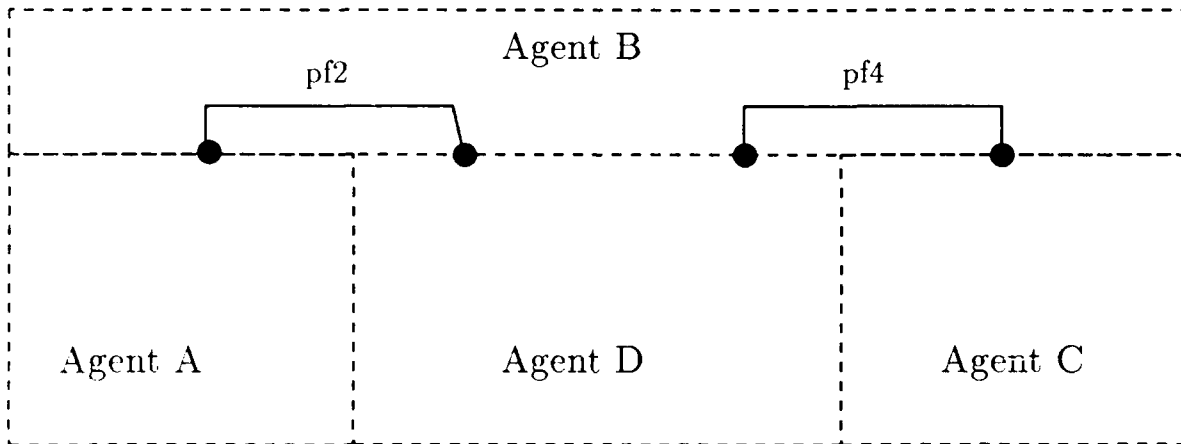
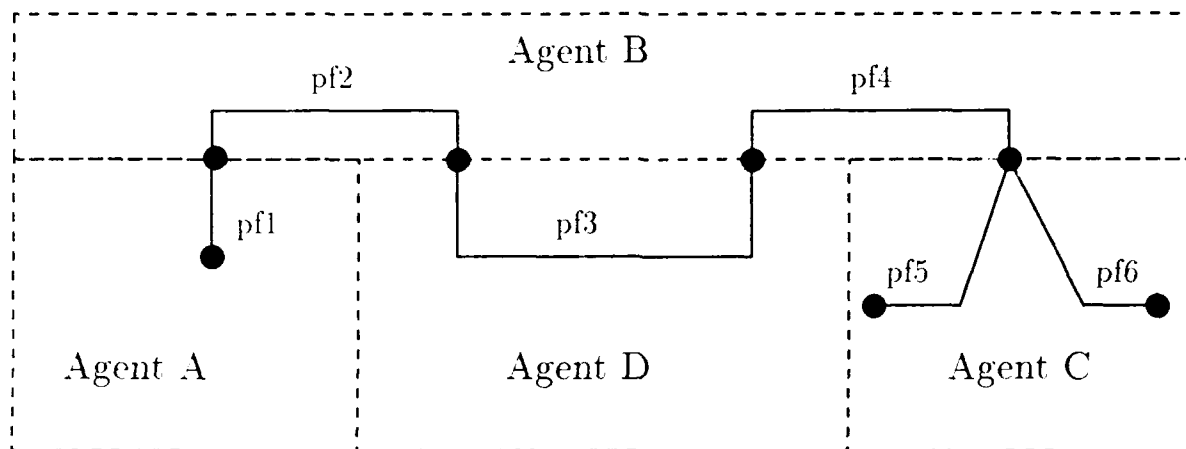


Figure 2: Limited View of Multiple Requests



Plan 1 : pf1-pf2-pf3-pf4-pf5

Plan 2 : pf1-pf2-pf3-pf4-pf6

Figure 3: Multiple Plans That Locally Appear to be One

end of plan generation. Agent B should combine plan fragments pf2 and pf4 into a single plan fragment because they represent a single alternative available to Agent B to partially satisfy this goal. From Agent B's limited view, this new plan fragment appears to be part of a single global plan when in reality it is part of two global plans. Whether this new plan fragment is part of a single global plan or multiple global plans is unimportant for the planning performance of Agent B. What is important, is the requirement that Agent B recognize that pf2 and pf4 should be combined into a single plan fragment, because they represent a single alternative for Agent B's participation in partial satisfaction of this goal.

4.3.4 Distributed Plan Generation

As is clear from the previous discussion, the objective of distributed plan generation is to determine sets of local actions that can be performed in a coordinated fashion by distributed agents to satisfy global goals. Thus, the collection of local actions (in multiple agents) that satisfies a global goal constitutes a global plan that exists as plan fragments distributed among the agents. A plan fragment, then, is a sequence of operator applications to objects under the control of an agent that would transform the global system, possibly through intermediate states, to a new state. When planning is viewed as a resource allocation problem, these operations include allocation of resources local to an agent. An agent can *extend* a plan fragment if the agent can formulate a plan fragment which would transform the system from the proposed new state to a state that is closer to the goal state. As previously indicated, agents have a limited view of resources which are

not under their direct control. Thus, each agent has limited knowledge concerning what state transformations other agents can make. Therefore it is impossible, in most cases, for a single agent to devise a global plan.

Plan generation begins when an agent is notified of the instantiation of a global goal. The agent creates a subgoal corresponding to this global goal and determines all sequences of actions it could take to bring the system to a state that locally appears closer to the goal state. Each alternative local sequence becomes a *plan fragment*. If any of these plan fragments would bring the system state to a new state that is not the goal state, the agent must issue requests for extension of the partial plan to agents that may be able to transform the system from the new state to the goal state or a state that may be nearer to the goal state. The search strategy is a modified version of the means end analysis strategy that has been used in several other planners [12, 13]. The approach in this context is somewhat different in that there is no global information available for an agent to determine whether, in fact, it can bring the system to a state that is closer to the goal state. Using local knowledge, the best each agent can do is determine state transformations that *locally* appear closer to the goal state.

It is clear that every request to extend a plan must carry certain information which will permit an agent to achieve a state that locally appears closer to the goal state. Specifically, a request must contain a description of a global goal, a description of the appropriate intermediate state, and a set of tag lists which are known as support names. Support names embody the information which enables each agent to recognize its own role in multiple plan decompositions without requiring complete knowledge of the global plan.

During plan generation a given agent may be asked to add an additional set of actions to the same global plan several times. Thus it is necessary that an agent be able to detect when it is being asked to build another piece of a global plan it has already partially constructed. If the agent has already built one or more parts of a plan, it must know *which* of its plan fragments were previously used in that plan. This information is needed for two reasons.

First, the agent must determine if it can extend the partial plan in a coherent manner based upon its previous participation in the construction of the plan. Specifically, the agent should not inadvertently build a plan which would bring the system to the same state twice. Permitting the system to cycle through the same state multiple times has two drawbacks: unnecessary work is performed and non-termination is a possibility. In addition, the agent must not allocate more "copies" of any given resource than it has available to a single global plan. Clearly such a plan would involve demands upon resources which could not be met.

To illustrate these concepts, consider Figure 4 which depicts plan generation as a series of operator applications that transform a possible plan from state to state. The resources involved in the application of each operator are displayed in parentheses. For the purposes

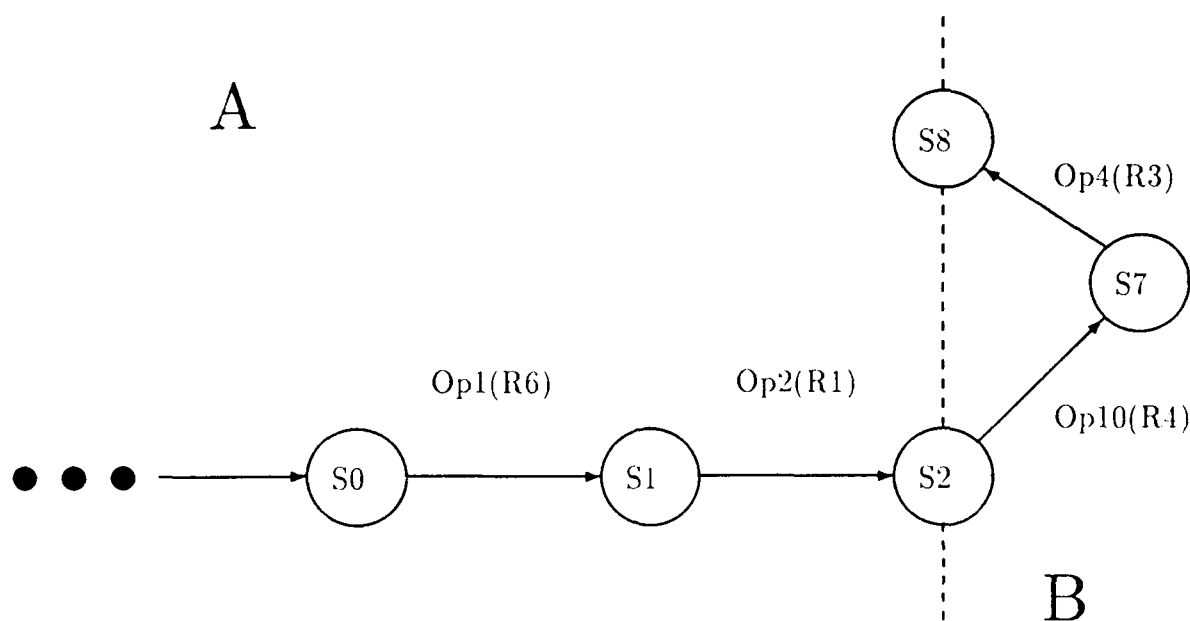


Figure 4: Multiple Participation of a Single Agent

of this example, assume that there is only one copy of each resource mentioned. Agent A has determined that it has one set of actions that would transform the system from state S_0 to a new state, S_2 , that locally appears to be closer to the goal state. This transition involves one intermediate state, state S_1 . However, in order to reach this new state, Agent A needs to use resource R_1 which it shares with Agent B. Therefore, Agent A must coordinate the use of this resource with Agent B. Agent B determines that if R_1 is used to reach state S_2 , it can extend the plan and bring the system to state S_8 which locally appears closer to the goal state. Reaching this new state involves coordinating the use of resource R_3 with Agent A. The critical issue illustrated here is that if Agent A attempts to fulfill Agent B's request to extend the plan starting from state S_8 , it must *not* do so by bringing the system through states S_0 , S_1 or S_2 because the decomposition of this plan has already been through those states. In addition, Agent A cannot propose an alternative that would use resources R_6 or R_1 , since this plan is already utilizing those resources, and thus they can not be allocated again. Therefore, Agent A must be able to recognize that it has already participated in this plan decomposition and identify the local plan fragments that are also used in this plan decomposition so that planning for the new request can take place in the proper context.

Furthermore, as discussed in Section 4.3.3.2, it is the responsibility of plan generation in a single agent to determine when groups of actions that have been formed as part of the same global plan decomposition eventually become components of feasible plans. When this occurs, the agent must gather the actions resulting from the various requests

into a single plan fragment. This is required for proper identification of potential subgoal interactions (such as contention for the same resource), where the interactions of concern are those *between* subgoals for different global goals. As stated before, reasoning about such subgoal interactions can only occur if each plan fragment available to an agent represents a distinct alternative for partial satisfaction of a particular goal.

Our mechanism for providing an agent with the means to recognize distinct roles in multiple plan decompositions involves attaching a list of *support names* to each plan fragment. Support names represent abstractions of the global plans associated with a plan fragment. They are incrementally constructed, with each agent appending a "tag" to identify its own plan fragments. These tags allow an agent to determine how a particular plan fragment is used in a global plan. They *do not* embody information which allows an agent to reason about the specific actions of other agents in a particular global plan. Support names do indicate which agents have participated in the construction of a plan but do not reveal the form of that participation. Since these abstracted plans are constructed incrementally as planning progresses, the support names do not even convey a skeletal structure of the complete plan. Instead, when an agent is requested to extend a plan, it can use each support name that is passed with the request as an abstract history of the construction of a single global plan thus far. This history contains information which allows each agent to recognize if and how it previously participated in the construction of the plan and what other agents aided in the construction of the plan.

Support names follow a plan as it is developed (in a semi-autonomous manner) by the agents. Once a plan has been completed, the requisite plan fragments can be marked by tracing continuation requests using the support name. Thus, an agent can determine which requests are part of the same global plan and which belong to distinct global plans. If it is determined that a plan cannot be completed, the appropriate support names are deleted.

4.3.5 Example: Generation of Service Restoral Plans

To illustrate the use of support names, an example taken from the domain of communications network management is presented. Consider the communications network shown in Figure 5. There are five problem solving agents, each controlling part of a network of geographically distributed communication facilities. The network partitions are called subregions, the circles represent communication sites and the lines joining sites represent communication links. In this domain, the problem of restoring disrupted service can be viewed as a planning problem in which one operator, **Allocate**, is utilized to allocate communication resources. In this simple example, the only resources are links and a global plan is a collection of local connections each of which allocate a link to restore communication between two sites. A partial plan or plan fragment involves an allocation of resources that transforms the system from a state in which it has a path ending at one site to one in which it has a new path ending at another site. The availability of

a resource depends upon its use by currently existing circuits or circuits whose service has been disrupted. Links which span subregion borders are controlled by the resident subregions of their endpoints. Links such as these are modeled as shared resources. In addition, no global topological information exists. Agents only know about the links they control directly and those they share. Thus, if one agent needs the aid of other agents to construct a plan in this domain, it asks those agents with whom it shares the control of a resource.

Table 1 summarizes the knowledge about resources available to each agent and the associated control relationships.

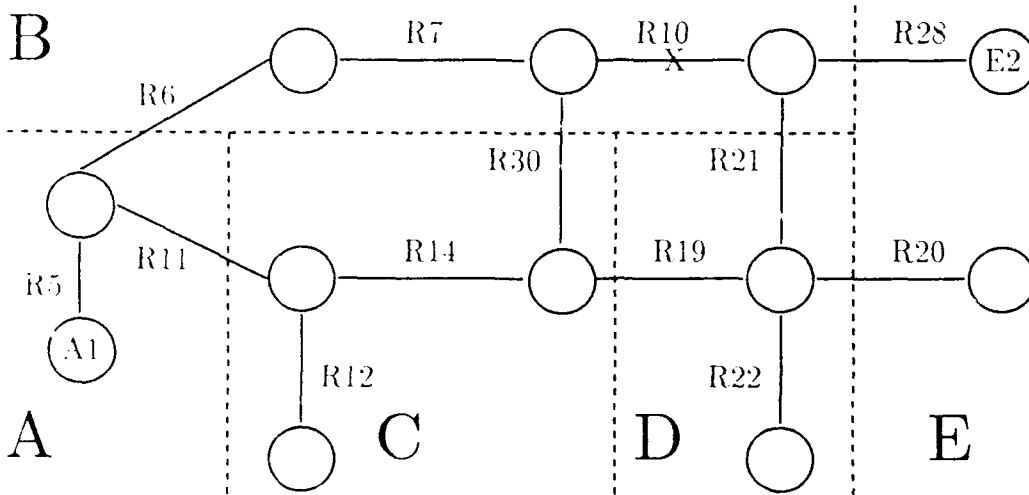


Figure 5: Example Network

Agent	Resources
A	R5 R6 R11
B	R6 R7 R10 R21 R28 R30
C	R11 R12 R14 R19 R30
D	R19 R20 R21 R22
E	R20 R28

Table 1: Local Resource Control

For the purpose of this example, assume that originally communication between sites E2 and A1 followed a path over links R28-R10-R7-R6-R5, but link R10 has failed and communication between sites E2 and A1 must be restored over a different route. Furthermore, suppose that Agent E is notified that a global goal to restore the path between sites E2 and A1 has been instantiated.

It is perhaps easiest to explain how plan generation proceeds by viewing the activities of agents at global time slices (see Tables 2 and 3). The actions presented in each time slice are described in two parts. The first part identifies a planning agent and what action is being taken by that agent. The second part is either a description of the action or a result returned by the action. The latter is denoted by a preceding arrow. The following legend describes the forms used in Tables 2 and 3.

local subgoal	<i>description</i>
local search	<i>(plan-fragment, resources-to-allocate, support-name-added)</i>
request <i>agent</i>	<i>extend(goal, shared-resource-to-use, support-name-to-add)</i>
notify <i>agent</i>	<i>remove(goal, resource-used, support-name-to-remove)</i> <i>acceptable(goal, resource-used, acceptable-support-name)</i> <i>add support(goal, resource-used, support-name-to-add)</i>
remove support	<i>remove-support(matching-plan-fragment, support-name-to-remove)</i>
acceptable support	<i>mark-acceptable(matching-plan-fragment, acceptable-support-name)</i>
add support	<i>add-support(plan-fragment, support-name-to-add)</i> .

The example begins with Agent E instantiating a subgoal to restore a communications path from site E2 to site A1. Agent E then conducts a local search and determines that it has one alternative that locally satisfies this subgoal, pfE1, which uses resource R28. Since Agent E has initiated the plan generation process, there is no previous support to associate with this plan fragment. Thus, a "*" is used to denote this situation. Control over resource R28 is shared with Agent B, therefore, a request is sent asking Agent B to extend the plan for this goal using this shared resource, R28, with support name (E1). Plan generation continues from this point.

Observe time slice T4. Here Agent E has determined that it has no alternatives which extend Agent D's request. Therefore, Agent E notifies Agent D that the support name that followed this plan decomposition should be removed. In time slice T5, Agent D uses the destination and resource in Agent E's notification to identify plan fragment pfD1 and Agent D removes the appropriate support name from this plan fragment.

In time slice T6, when Agent A completes a global plan using a request from Agent B, it sends back a notification to Agent B that the request has resulted in an acceptable plan. Agent B receives this notification in time slice T7 and determines that it owns two of the tags in the support name associated with the request. Agent B identifies pfB2 and pfB1 with these tags and thus realizes that they are part of a single plan decomposition that has resulted in an acceptable plan. Therefore, Agent B creates a new plan fragment, pfB3, which uses the resources of both pfB2 and pfB1 and Agent B gives this new plan fragment support. Since no acceptable global plan uses pfB2 alone, its support names are removed.

Note that in time slice T8, Agent C removes a support name from pfC2 due to the propagation of support name removal started in Agent B. However, this propagation ends

T1	E: subgoal E: local search E: request B	to restore path from E2 to A1 → (pfE1, (R28), *) extend (A1, R28, (E1))
T2	B: local subgoal B: local search B: request D	to A1 using R28 → (pfB1, (R28, R21), (E1)) extend (A1, R21, (B1, E1))
T3	D: local subgoal D: local search D: request E D: request C	to A1 using R21 → (pfD1, (R21, R20), (B1, E1)) → (pfD2, (R21, R19), (B1, E1)) extend (A1, R20, (D1, B1, E1)) extend (A1, R19, (D2, B1, E1))
T4	C: local subgoal C: local search C: request A C: request B E: local subgoal E: local search E: notify D	to A1 using R19 → (pfC1, (R19, R14, R11), (D2, B1, E1)) → (pfC2, (R19, R30), (D2, B1, E1)) extend (A1, R11, (C1, D2, B1, E1)) extend (A1, R30, (C2, D2, B1, E1)) to A1 using R20 → no alternatives remove (A1, R20, (D1, B1, E1))
T5	A: local subgoal A: local search A: notify C B: local subgoal B: local search B: request A D: remove support	to A1 using R11 → (pfA1, (R11, R5), (C1, D2, B1, E1)) acceptable (A1, R11, (C1, D2, B1, E1)) to A1 using R30 → (pfB2, (R30, R7, R6), (C2, D2, B1, E1)) extend (A1, R6, (B2, C2, D2, B1, E1)) remove-support (pfD1, (B1, E1))
T6	A: local subgoal A: local search A: notify B C: acceptable support C: notify D	to A1 using R6 → (pfA2, (R6, R5), (B2, C2, D2, B1, E1)) acceptable (A1, R6, (B2, C2, D2, B1, E1)) mark-acceptable (pfC1, (D2, B1, E1)) acceptable (A1, R19, (D2, B1, E1))
T7	B: new plan fragment B: remove support B: notify C B: notify C D: acceptable support D: notify B	combine (pfB3, (pfB1, pfB2), (C2, D2, B3, E1)) remove-support (pfB2, (C2, D2, B1, E1)) remove (A1, R30, (C1, D2, B1, E1)) add support (A1, R30, (C1, D2, B3, E1)) mark-acceptable (pfD2, (B1, E1)) acceptable (A1, R21, (B1, E1))

Table 2: Time Slice View of Example, T1-T7

T8	B: acceptable support B: notify E C: add support C: remove support C: notify D	mark-acceptable (pfB1, (E1)) acceptable (A1, R28, *) add-support (pfC2, (D2, B3, E1)) remove-support (pfC2, (D2, B1, E1)) add support (A1, R19, (D2, B3, E1))
T9	D: add support D: notify B E: acceptable support	add-support (pfD2, (B3, E1)) add support (A1, R21, (B3, E1)) mark-acceptable (pfE1, *)
T10	B: add support	add-support (pfB3, (E1))

Table 3: Time Slice View of Example, T8-T10

here because there is another plan fragment, pfC1, which uses the same support name (see T4). This represents a place where the search for a global plan split into two parallel search paths. In this example, the second plan fragment that uses the same support name has been marked as part of an acceptable global plan in time slice T6. The propagation of this acceptable support name has reached Agent B in this time slice. Agent B is notified that pfB1 is part of an acceptable plan. Using the support name, Agent B determines that pfB1 is part of an acceptable global plan that does not use other plan fragments in Agent B. As a result, the support names for pfB1 are marked acceptable.

Agent	Plan Fragments	Resources	Support Names
A	pfA1 pfA2	R11-R5 R6-R5	(C1 D2 B1 E1) (B2 C2 D2 B1 E1)
B	pfB1 pfB2 pfB3	R28-R21 R30-R7-R6 R28-R21 R30-R7-R6	(E1) none (C2 D2 B3 E1)(E1)
C	pfC1 pfC2	R19-R14-R11 R19-R30	(D2 B1 E1) (D2 B3 E1)
D	pfD1 pfD2	R21-R20 R21-R19	none (B1 E1)(B3 E1)
E	pfE1	R28	*

Table 4: Results of Plan Generation Example

Table 4 shows the plan fragments created by each agent, the resources used by these plan fragments, and the support names associated with each plan fragment at the end of plan generation. Note that pfD1 has no support names because this plan fragment is not part of any acceptable global plan.

To summarize the important points of this example, Agent B created two plan fragments, pfB1 and pfB2, as a result of requests to add to partially constructed plans for the same global goal. Through the use of support names, Agent B has determined that pfB1 and pfB2 are used in one set of acceptable global plans (in this case one plan) and pfB1 is also part of a different set of acceptable global plans (also one plan in this example). When pfB1 and pfB2 are used as parts of the same global plan, they represent a single alternative in Agent B for this global plan. This has been reflected by the creation of pfB3. The determination of how these plan fragments, created out of separate requests, fit into global plans has been accomplished without any single agent having complete knowledge about any of the acceptable global plans generated.

4.3.6 Experimental Results

4.3.6.1 Description of Experiments Research in distributed planning is currently being conducted in the context of the communications domain described in the previous example. The implementation model¹, however, contains much more of the detail associated with a real world communications network [4]. Local searches for plan fragments are not simple searches for paths of links in and out of a subregion as might be assumed given the example above. On the contrary, local searches involve tracing through complex interconnections of various types of communications equipment at the sites within a subregion.

Existing planners use several different architectures and moreover, the level of abstraction at which planning occurs varies from system to system. Experiments have been conducted so that distributed plan generation as presented in this report may be compared to plan generation schemes with various architectures using different levels of abstraction. In each of the tested schemes, an agent which has control over part of a network has detailed information about that part of the network and *only* that part of the network. If any other information is used for plan generation, it is either abstract knowledge in the form of plan fragments, or limited abstract knowledge in the form of support names. The following is a description of the plan generation paradigms used in these experiments. The first is a single agent system and the rest are multiple agent systems.

Single Agent/Detailed Global View (SA/DGV) A single agent is responsible for the entire system rather than distributing system knowledge among multiple agents. In this approach, a local search for plan fragments is equivalent to a global search for global plans that will satisfy system goals.

Multiple Agent/Limited Abstract Global View (MA/LAGV) This is the approach described in this report. Plans are constructed by multiple agents which

¹The methods described in this report have been implemented on a TI Explorer in Common Lisp. Simulation of the multiagent processing has been accomplished through the use of SIMULACT [24].

have an incomplete, limited view of the global plans. This incomplete, limited view is determined by the incremental construction of support names and therefore, is different at each agent in the system.

Multiple Agent/Central Abstract Global View (MA/CAGV) Agents use the descriptions of the circuits which are to be restored to determine all the possible ways they might be able to participate in a global plan. The results of these local searches are sent to a single agent who pieces the plan fragments together into acceptable global plans. Once this is completed, each agent is notified of its participation in global plans. The view of this single planning agent is not limited in the sense that it does know about the complete set of plan fragments in the system. However, its view is abstract since this agent knows nothing about the detail of communications equipment and its interconnection at each site.

Multiple Agent/Replicated Abstract Global View (MA/RAGV) As in the MA/CAGV approach, local searches are conducted by each agent using high level circuit descriptions. The results of these searches, however, are sent to every other agent in the system. Then, with complete knowledge of every plan fragment in the system, each agent forms the global plans and determines its own role in each.

The key parameters monitored in these experiments are the simulated time required to generate plans, the average cpu time required by each processing node to generate plans, and the amount of message traffic sent during the simulation.

In addition, three network configurations were chosen to observe the effect of various topological extremes. In this domain, the network topology actually defines the complexity of the roles of agents in the multiple plan decompositions. Therefore, by varying these topological extremes it is also possible to observe the performance of these strategies when agent participation takes on roles of different complexity. Each network contains twelve sites divided into five subregions with various inter- and intrasubregion connectivity. Figure 6 shows the configuration where the subregions are connected in a straight line and Figure 7 shows the subregion connections which form a ring. The third topology chosen is shown in Figure 8. Here each subregion is connected to every other subregion creating a tightly coupled network.

4.3.6.2 Experimental Results The results of these experiments are shown in Figures 9, 10, 11, 12, 13, 14, 15, 16, and 17. As expected, the SA/DGV strategy performs the worst in terms of the time taken to devise global plans. This observation holds true over each of the tested topologies. This points to the desirability of distributed multiagent systems over centralized single agent systems when the systems are large.

The MA/CAGV, MA/RAGV, and MA/LAGV strategies all take about the same amount of time to determine global plans for the line topology. As well, the cpu time

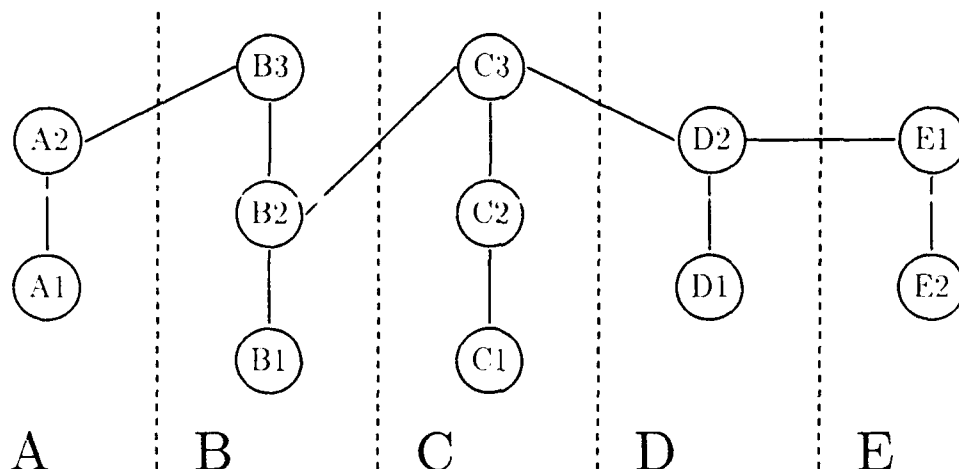


Figure 6: Line Topology

per agent is approximately the same. However, the amount of message traffic required by the MA/RAGV strategy exceeds that of both the MA/CAGV and MA/LAGV strategies with the MA/CAGV strategy performing better as the number of goals grows.

For the ring topology, the cpu time per agent for the multiagent strategies begins to separate with the MA/CAGV strategy clearly performing better as the number of goals increases. The MA/RAGV and MA/LAGV strategies appear to be following approximately the same line. Regarding the time to construct global plans, the MA/RAGV and MA/CAGV strategies outperform the MA/LAGV when the number of goals is small. However, as the number of goals increases, the lines appear to be converging. The results for the message traffic required shows that the MA/RAGV and MA/LAGV strategies have approximately the same requirements while the MA/CAGV strategy requires less message traffic.

When the topology is tightly coupled, the strategies perform with significant differences. The MA/LAGV strategy clearly requires less time than both the MA/CAGV and MA/RAGV to devise plans as the number of goals increases. However, the cpu time per agent required is clearly less for the MA/CAGV strategy with the MA/LAGV strategy coming in second and the MA/RAGV performing worse. In addition, there is a marked difference in the amount of message traffic required by the different strategies. The MA/LAGV strategy requires the most message traffic, the MA/RAGV less, and the MA/CAGV still less.

For the network topologies tested, there is a clear question of trade offs. For the ring and line topologies, the MA/CAGV strategy performs better overall. The price paid however is vulnerability. In domains where survivability is an important concern, such

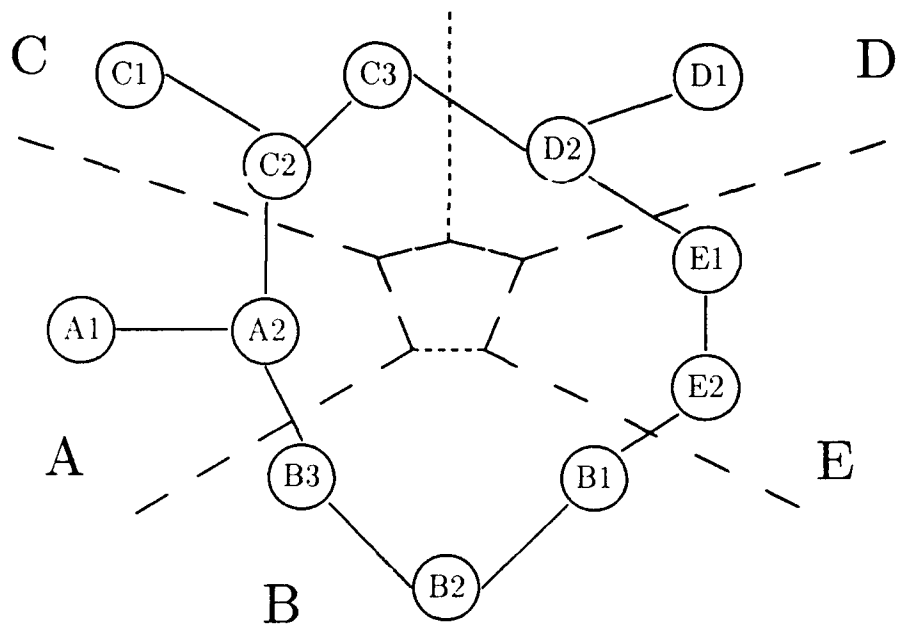


Figure 7: Ring Topology

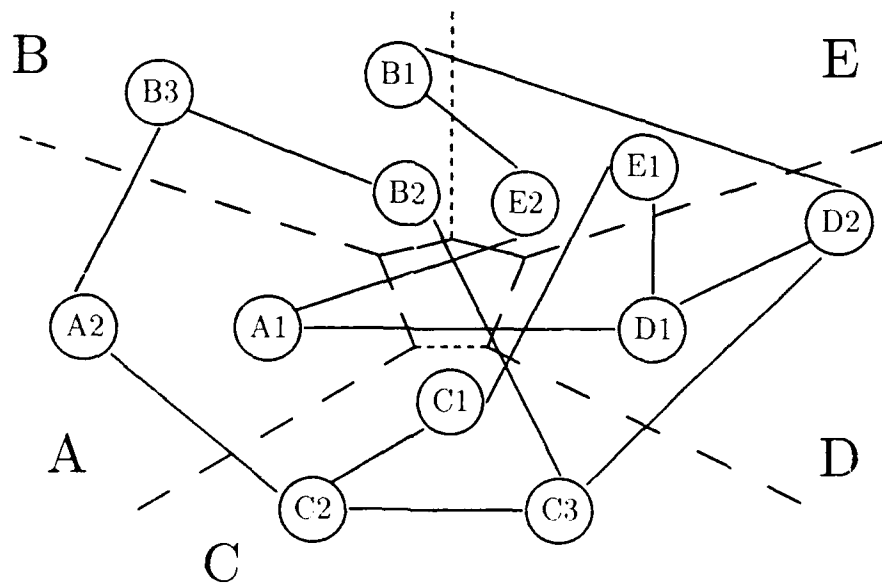


Figure 8: Tightly Coupled Topology

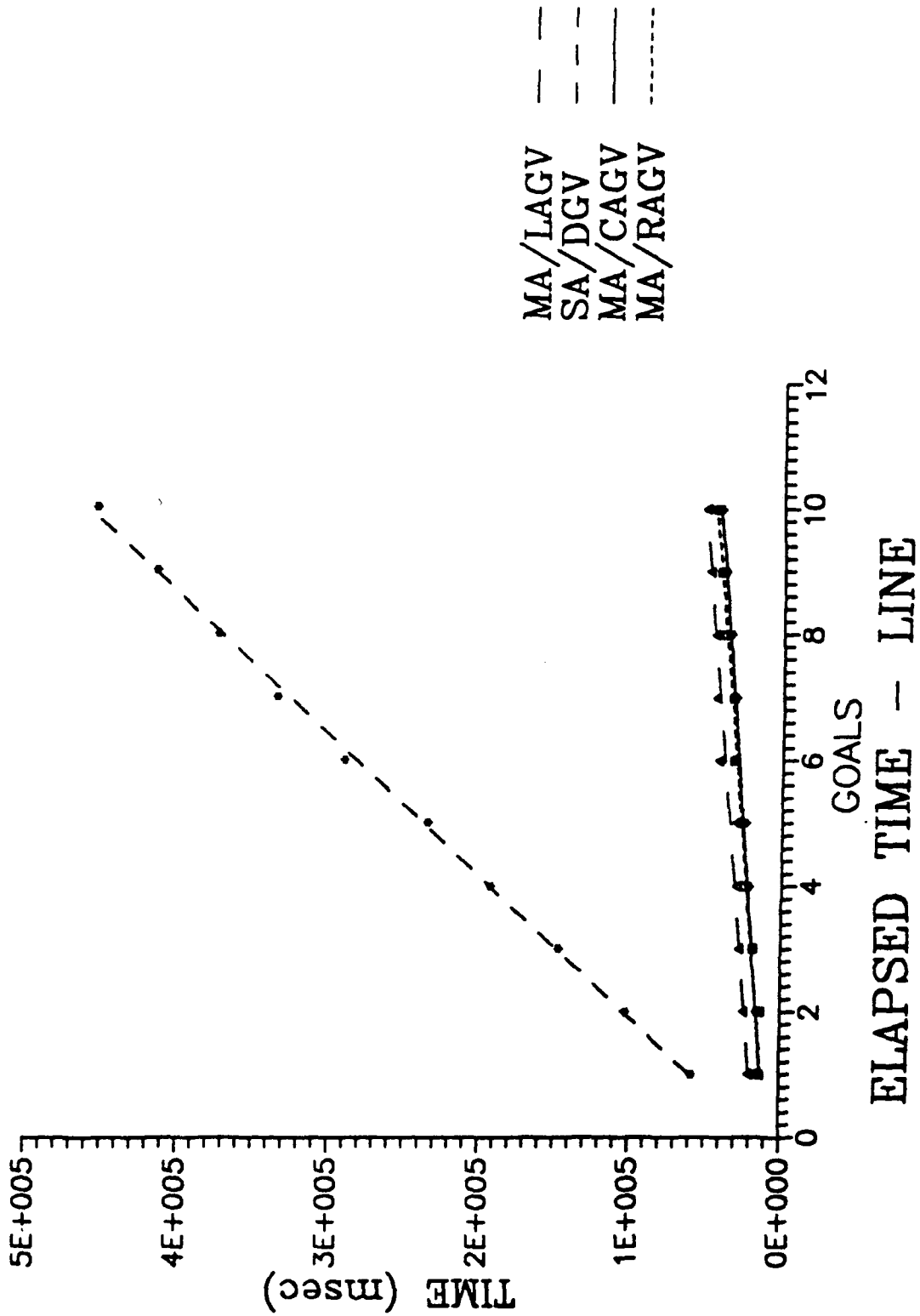


Figure 9: Experimental Results: Line Topology

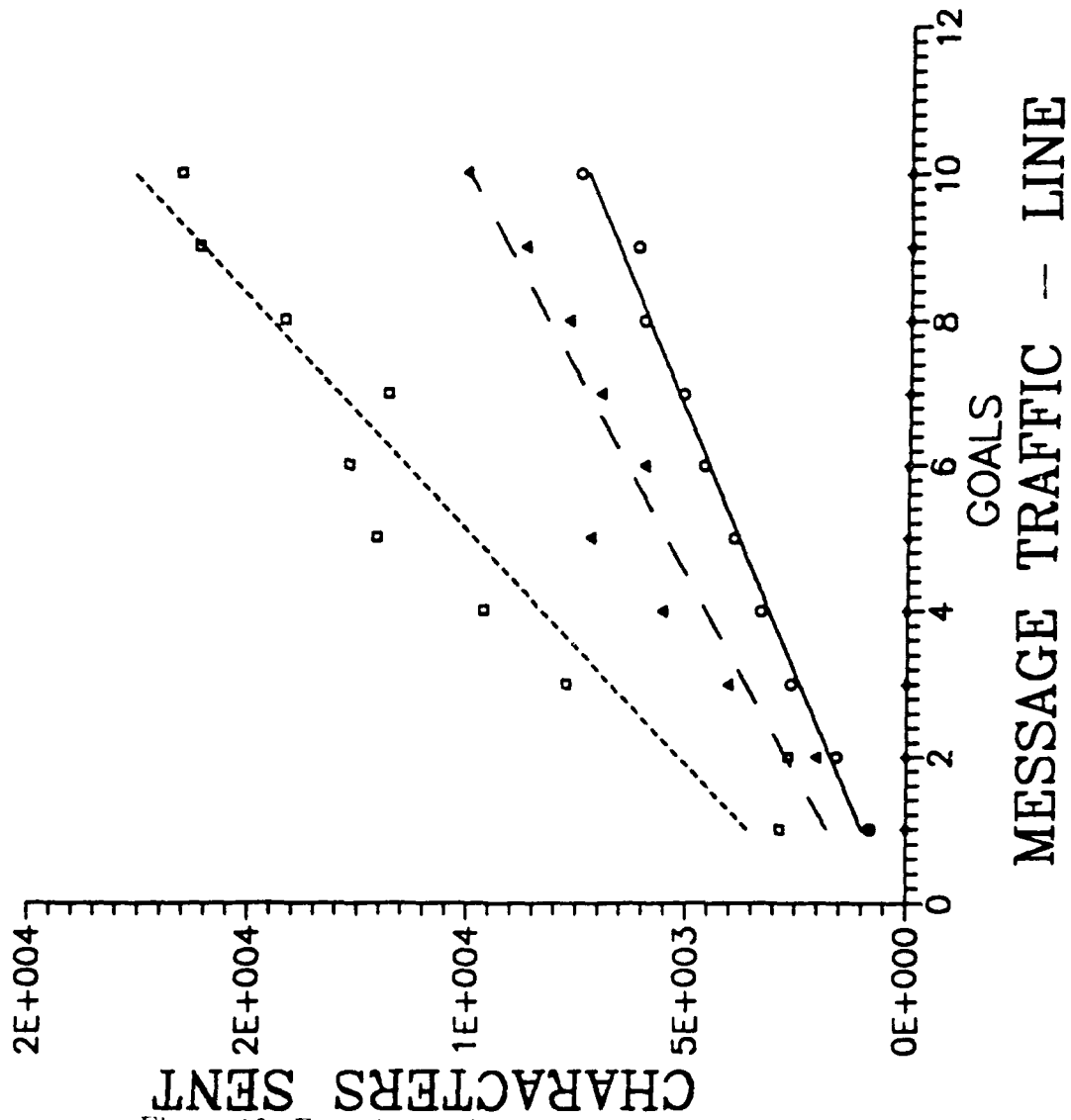


Figure 10: Experimental Results: Line Topology

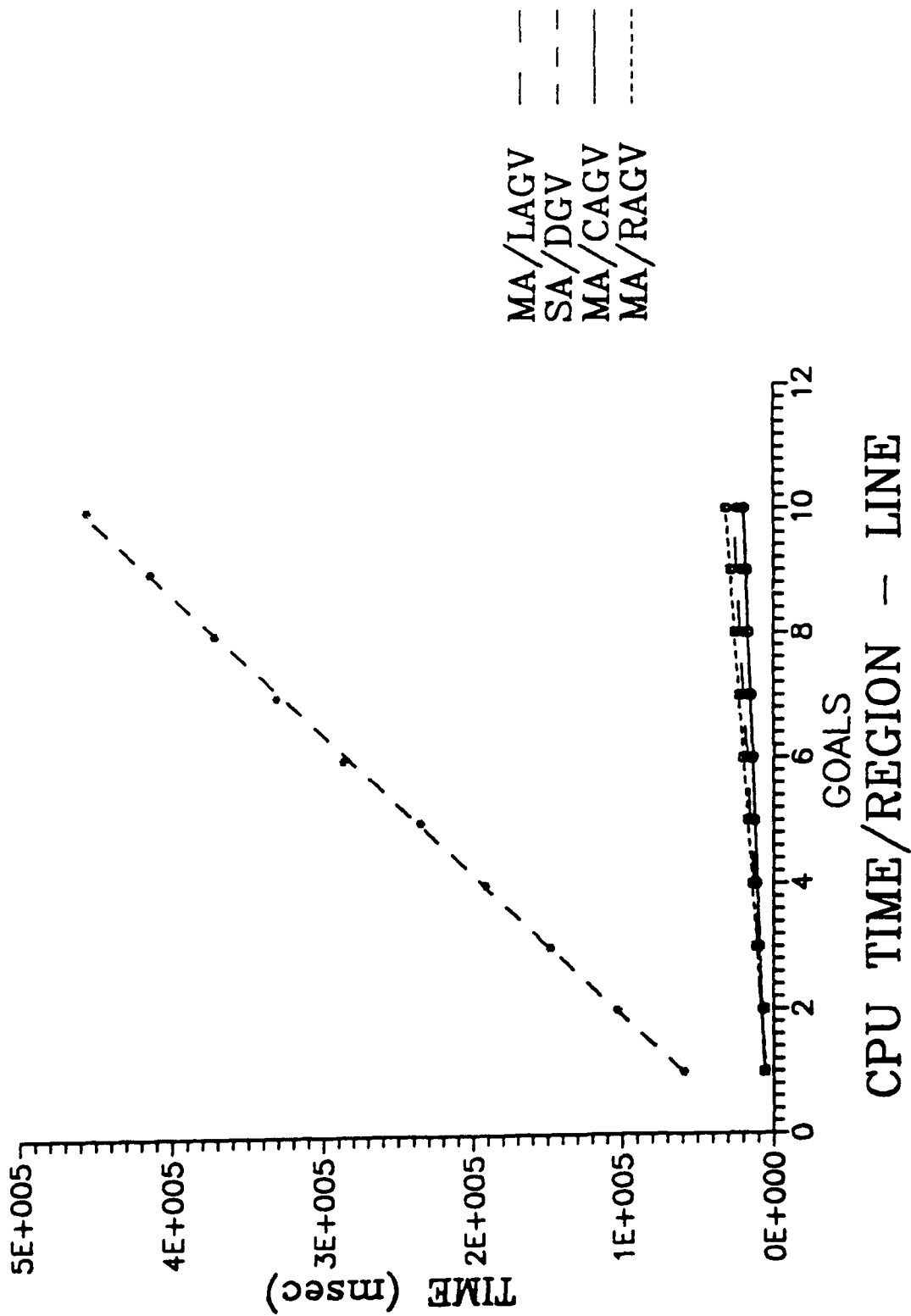


Figure 11: Experimental Results: Line Topology

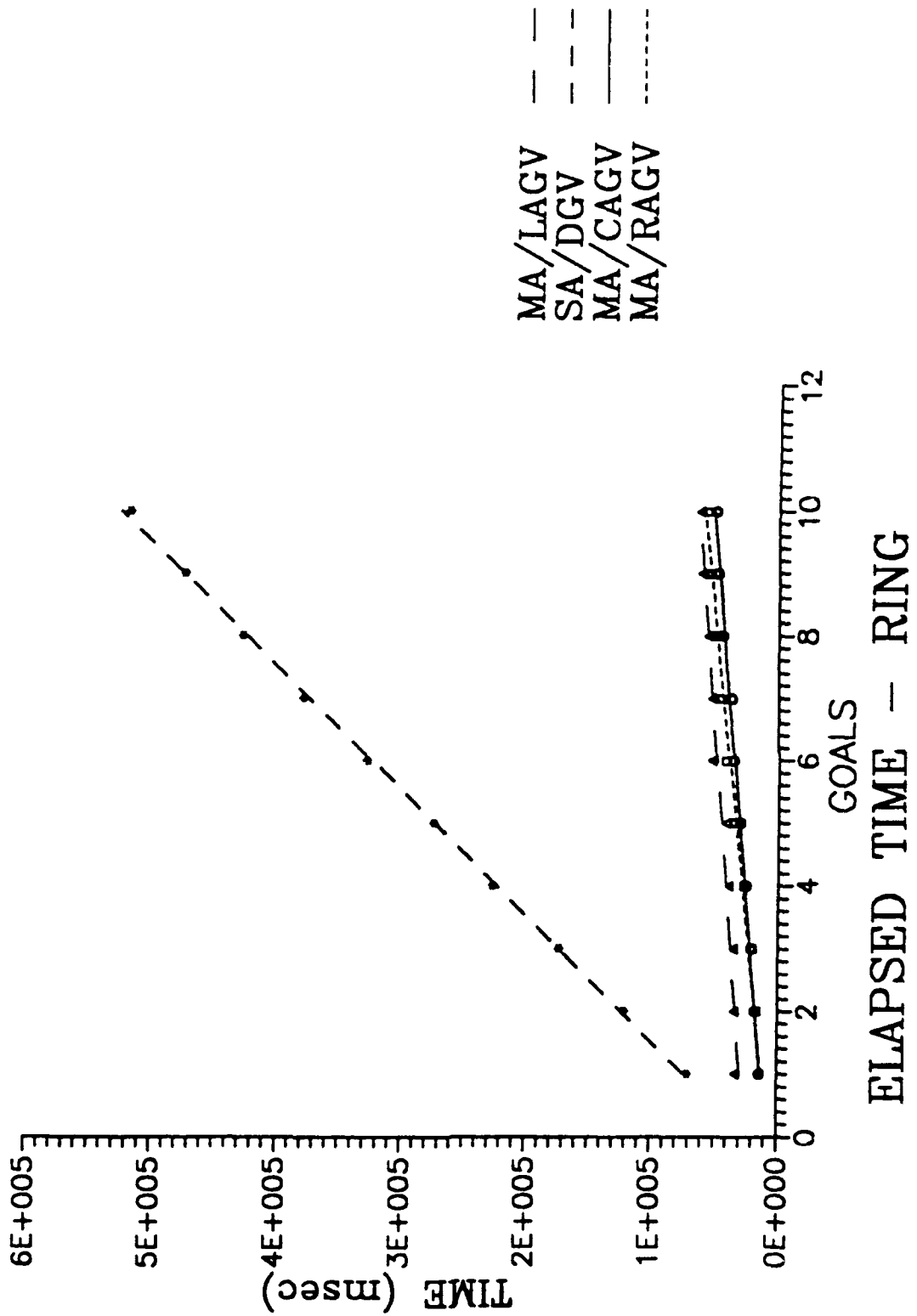


Figure 12: Experimental Results: Ring Topology

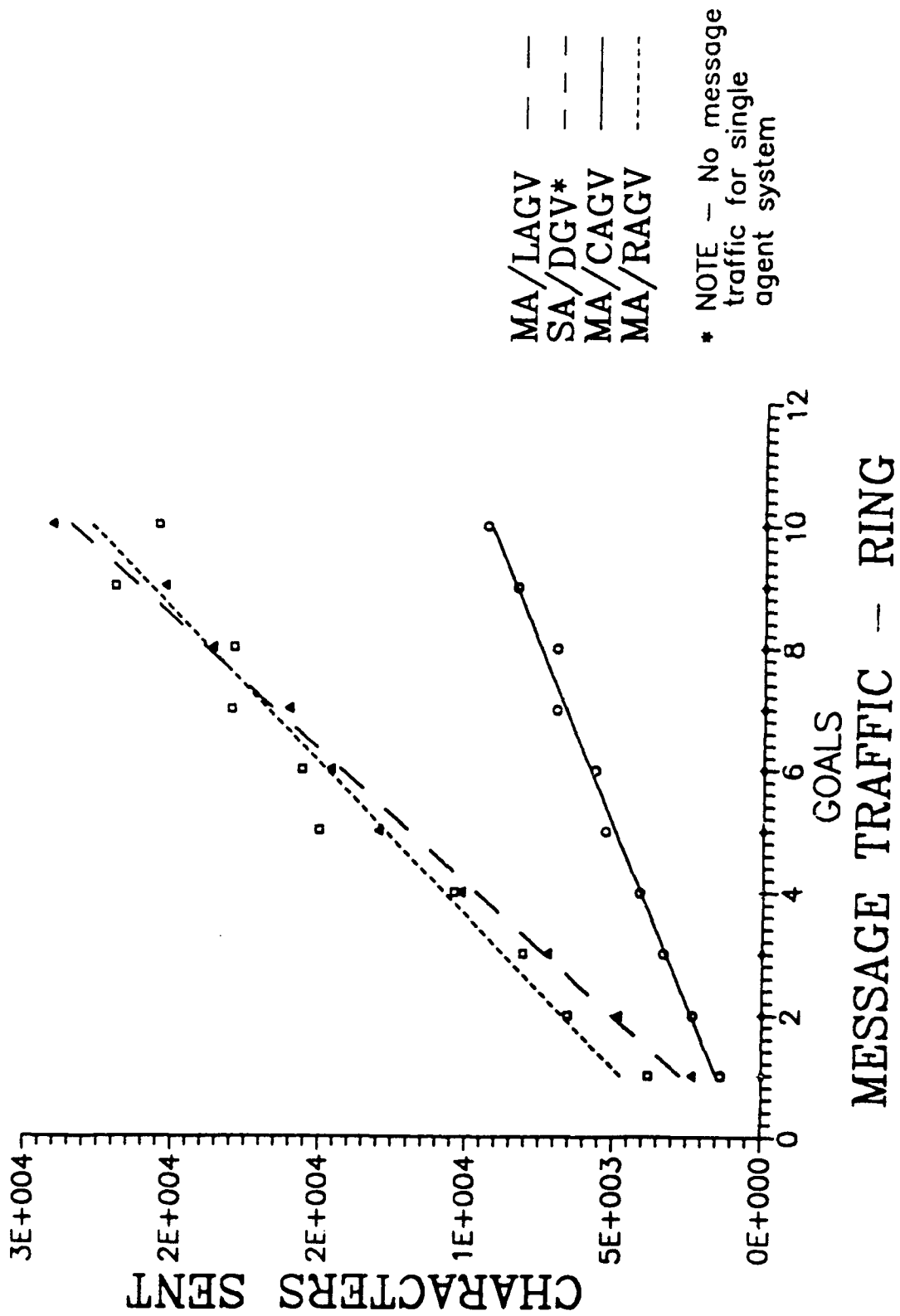


Figure 13: Experimental Results: Ring Topology

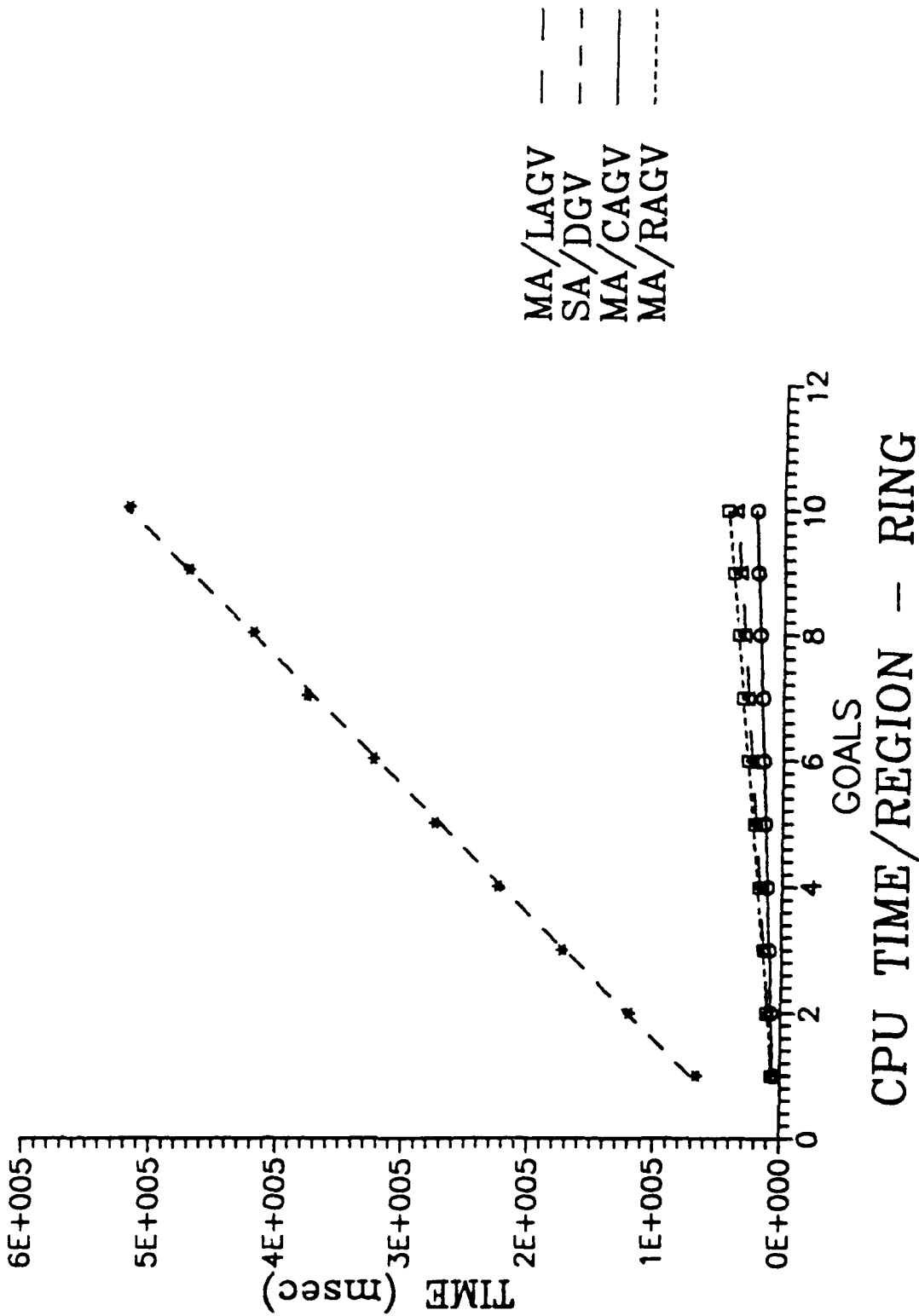


Figure 14: Experimental Results: Ring Topology

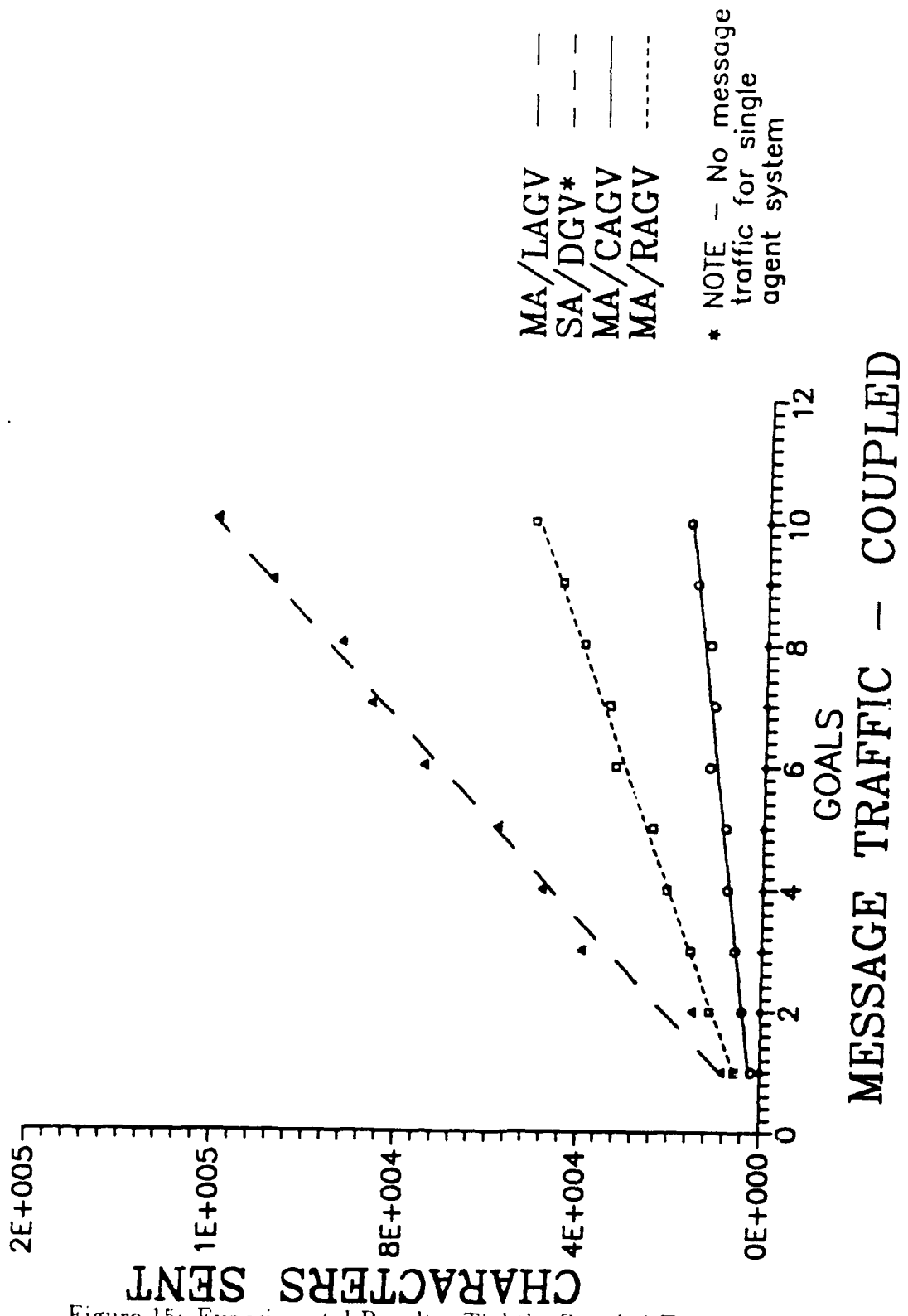


Figure 15: Experimental Results: Tightly Coupled Topology

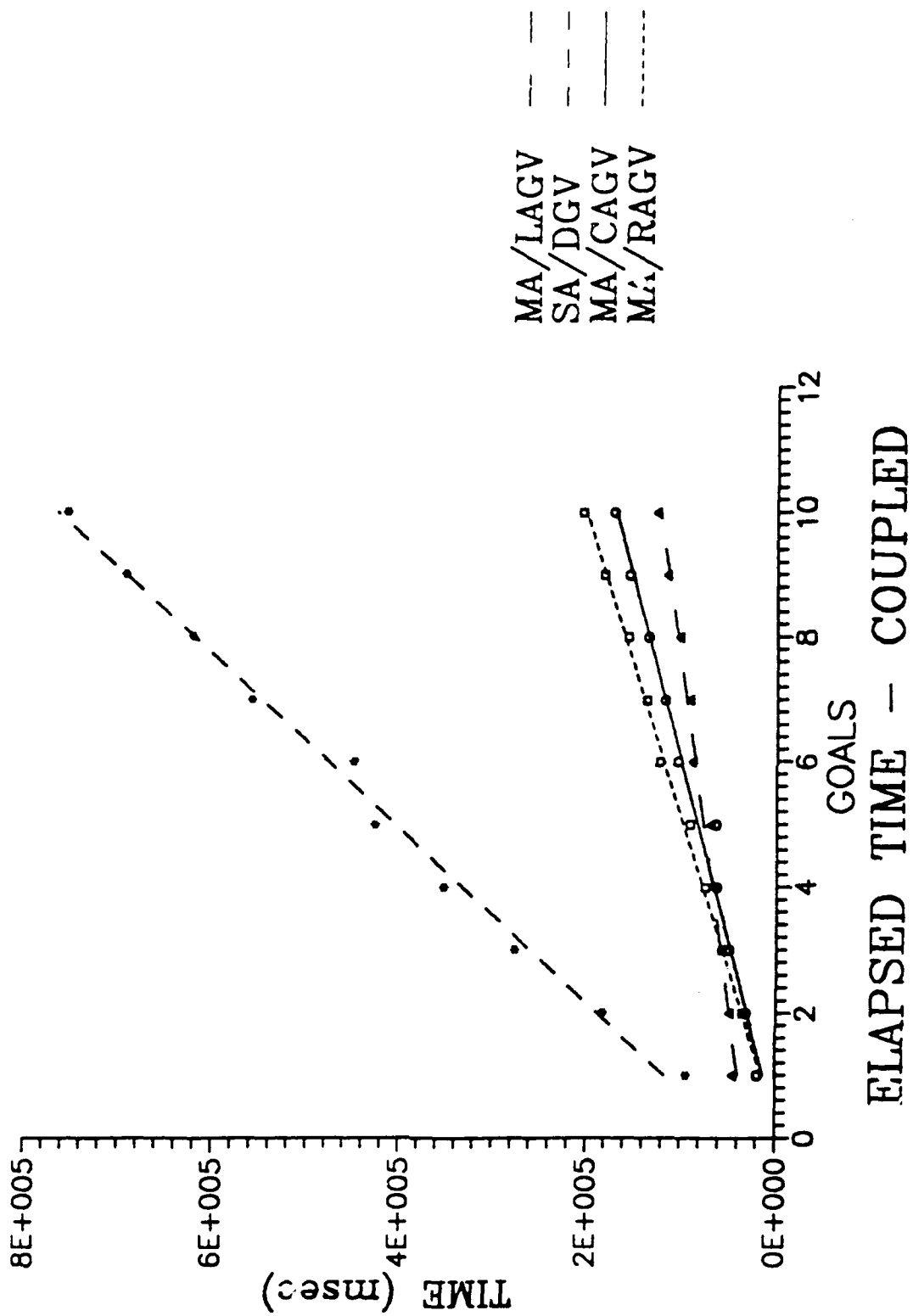


Figure 16: Experimental Results: Tightly Coupled Topology

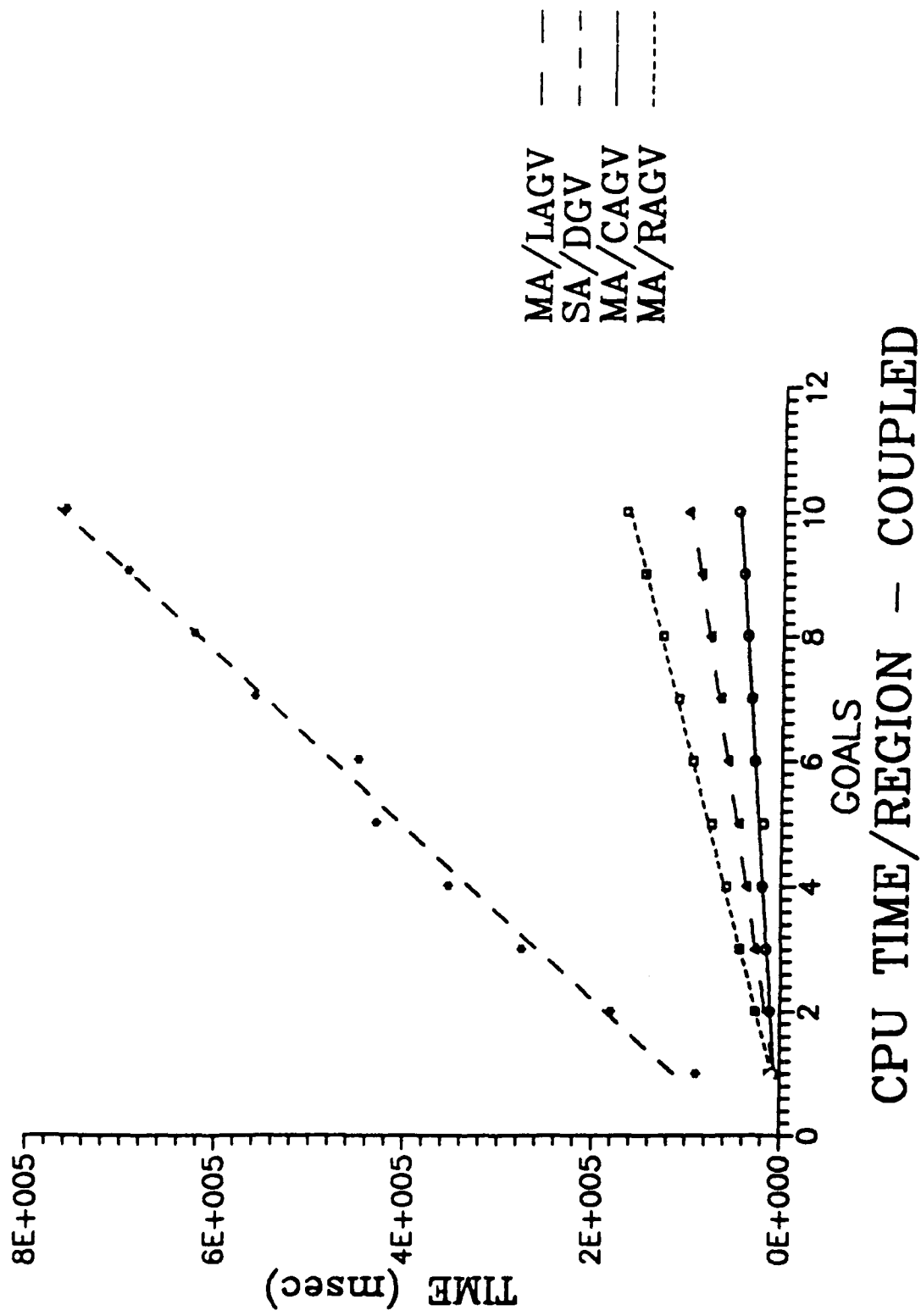


Figure 17: Experimental Results: Tightly Coupled Topology

as a military communications network, the MA/CAGV strategy obviously is undesirable because of the dependence upon a single agent. For the tightly coupled topology, the MA/LAGV strategy will take less time to construct plans but the price paid is in the amount of message traffic required.

4.3.6.3 Performance Analysis The performance of distributed plan generation can be analyzed by considering the time required to generate plans and the amount of message traffic sent.

The time required to generate plans is influenced by factors on two levels. At one level, this parameter is dependent upon the amount of time required to pass the plan among each of the agents involved in its construction. Therefore, from a global viewpoint, the time required to generate plans is directly related to the length of the longest chain of agents involved in building a plan. At another level, the amount of time required to generate plans is determined by the processing time of each individual agent. As the relations between requests to extend a plan and multiple plan decompositions become more complex, so does the processing involved to determine distinct alternatives. Thus, from a global perspective, the time required to generate plans is also directly related to the complexity of the roles of agents in multiple plan decompositions.

The message traffic necessary for plan generation is also directly related to the participation of agents in multiple plan decompositions. When an agent is notified that a plan it has helped to build has been deemed acceptable, that agent is responsible for the propagation of this information. If the agent participated only once in the plan construction, a single message is required to continue the propagation. However, if the agent participated multiple times in the construction, then two messages are sent, one to propagate the new support name and one to remove the old support name. Thus, the message traffic required to generate plans increases as the complexity of the roles of agents in multiple plan decompositions increases. However, it should be noted that the amount of message traffic required does not approach that which would be needed to transmit complete, detailed global information to each agent in the system.

These experiments illustrate that distributed plan generation can be accomplished by passing merely a limited amount of information among system agents. The only information required includes descriptions of the goal state and the present state of the plan, and information which allows agents to determine their previous actions in the construction of the plan. This last piece of information is provided by the implementation of support names. Experimentation shows that building a complete, detailed global

4.3.7 Future Directions

As a result of the preliminary experiments reported in this section, new directions for experimentation have become clear. Perhaps the most immediate is to design a network

that models the scale of a real world communications network. It will be interesting to observe the effects upon the relative performance of the algorithms in a network which contains a greater volume of information.

Some modifications should be made to the plan generation phase to permit its use in a larger class of domains. One such alteration involves relaxing the definition of allocation in the model. Rather than requiring that resources be allocated for the duration of the satisfaction of a goal, the model should be modified to allow for the use of a resource for a period of time and then allow it to be relinquished for allocation for another purpose. Thus, the model will then be able to allow the dynamic scheduling of resources. In addition, the model as it exists makes an implicit assumption about the capabilities of the agents in the system. Namely, no two agents can bring a single plan to the same state. One solution to allow this situation would be to increase the information included in the support name. This might possibly be accomplished through a globally recognized encoding scheme for abstract state descriptions of plans.

4.4 Cooperation Using Constraint-Based Reasoning

In this section we present formalisms that form the basis for multistage negotiation. We demonstrate that these formalisms permit an agent in a distributed planning system to gain knowledge about the interaction between consequences of its local actions and constraints existing elsewhere in the system. Our work provides mechanisms for determining impact at three levels: locally on the level of plan fragments, locally on the level of goals, and nonlocally. Abstractions that reflect these interactions are formulated and properties of the abstraction mechanisms are discussed. In addition, algorithms are given for computing local structures and their complexity is analyzed as an indicator of the worst case performance that can be expected. Finally, bounds on the number of transactions required to propagate local impact to distant sites are derived. We also show how this formalism provides a natural mechanism by which agents incrementally expand knowledge about the nonlocal impact of their local decisions *without* constructing a complete global view.

4.4.1 Abstraction of Constraints and Conflicts

For the purposes of illustrating our definitions, we consider a scenario involving four agents in a distributed system cooperatively attempting concurrent satisfaction of four goals. A number of global plans have been constructed during plan generation, as indicated in Table 5. In Table 5, each goal is identified by g_i ($i = 1, 2, 3, 4$). The set of alternative plans for each specific goal g_k are identified by g_k-p_l ($l = 1, 2, \dots$). Thus we see that goal g_1 has five distinct alternative plans, g_1p_1, g_1p_2 , etc.

plan	plan fragments	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11
g1p1	A-a B-a C-a D-a	1	1			1	1			1		
g1p2	A-a C-a D-b	1	1			1		1	1			
g1p3	A-a C-c	1	1		1							
g1p4	A-b B-b C-b	1			1						1	1
g1p5	D-c						1		1			
g2p1	A-d C-d	1	1	1	1							
g2p2	A-e B-c D-d	1						1		1		1
g2p3	C-e D-e			1	1	1	1		1			
g3p1	A-f C-g D-f	1	1	1		1		1				
g3p2	A-f C-g D-g	1	1	1		1	1					
g3p3	C-h D-f				1	1		1				
g3p4	C-h D-g				1	1	1					
g4p1	A-g B-d	1										1
g4p2	A-g B-e C-j	1		1	1						1	1
g4p3	A-h C-i D-h		1			1		1				
g4p4	A-h C-i D-i		1			1	1					
g4p5	C-k D-h				1	1		1				
g4p6	C-k D-i				1	1	1					

Table 5: Global Plans Generated

It should be noted that Table 5 shows the global plans from a *global* perspective. No single agent in a distributed problem solving system has complete knowledge concerning any of these plans. Indeed, except in unusual circumstances, no single agent is even aware of the total number of alternative plans that have been generated.

From Table 5, it is evident that global plans are composed of collections of local *plan fragments*. For instance, global plan g3p3 is composed of plan fragments C-g and D-f. Plan fragment C-g denotes a set of local actions that agent C could take in partial satisfaction of goal g3. Satisfaction of g3 using g3p3 would require the actions D-f by agent D as well as the set of actions C-g in agent C.

Local knowledge about plan fragments is shown in Table 6. Notice that if the entry on the *resource count* line for resource r in agent i is k , then agent i has k copies of resource r to utilize in problem solving. The shared resources are evident, as they are known to more than one agent. Observe that $r10$ is a shared resource. There is only one copy of $r10$ in the system, and its allocation must be jointly controlled by agents B and C.

It is important to note that each agent has *only* the local knowledge about plan

Agent A				
goal	plan frag	r1	r2	r11
resource count		3	2	2
g1	A-a	1	1	
	A-b	1		1
g2	A-d	1	1	
	A-e	1		1
g3	A-f	1	1	
g4	A-g	1		1
	A-h		1	

Agent C						
goal	plan frag	r2	r3	r4	r5	r10
resource count		2	3	2	2	1
g1	C-a	1			1	
	C-b			1		1
	C-c	1		1		
g2	C-d	1	1	1		
	C-e		1	1	1	
g3	C-g	1	1		1	
	C-h			1	1	
g4	C-i	1			1	
	C-j		1	1		1
	C-k			1	1	

Agent B				
goal	plan frag	r9	r10	r11
resource count		1	1	2
g1	B-a	1		
	B-b		1	1
g2	B-c	1		1
g4	B-d			1
	B-e		1	1

Agent D						
goal	plan frag	r5	r6	r7	r8	r9
resource count		2	2	1	3	1
g1	D-a	1	1			1
	D-b	1		1	1	
	D-c		1		1	
g2	D-d			1		1
	D-e	1	1		1	
g3	D-f	1		1		
	D-g	1	1			
g4	D-h	1		1		
	D-i	1	1			

Table 6: Local Knowledge About Plan Fragments

fragments shown in Table 6. This means, for example, that agent A is aware that plan fragment A-b for goal g1 coordinates with some plan fragment known to agent B as a component in some global plan or plans in satisfaction of g1. Agent A knows this because resource r11 is shared between agents A and B. Agent A *does not know* anything about plan fragments that are local to agent B.

To enable an agent to efficiently exchange knowledge concerning the nonlocal impact of local decisions, we determine a *conflict set* for each plan fragment. We then use the conflict set to construct an *exclusion set* for each plan fragment that reflects the potential impact on an agent's ability to participate in satisfying other goals, assuming that plan fragment x is executed. At the highest level of abstraction, we use exclusion sets to form *infeasibility sets*. Knowledge summarized in its infeasibility sets allows an agent to reason about the way in which its decision to satisfy one goal may affect its ability to satisfy other goals. Finally, we propagate these local concepts to other agents with the construction of *induced exclusion sets*.

Before formalizing these concepts, we need to define the notational conventions used in the discussion which follows.

1. We define maximal and minimal subsets of sets whose elements are sets in the standard way. Given a set of sets $S = \{S_1, \dots, S_n\}$ with a partial order $<$ defined on subsets of S in the standard way (that is, $S_i < S_j \Leftrightarrow S_i \subseteq S_j$), we say that S_i is **maximal** if $\nexists S_j \ni: S_i < S_j$. Furthermore, S_i is **minimal** if $\nexists S_j \ni: S_j < S_i$.
2. $P_A = \{ \text{all plan fragments known to agent A} \}$.
3. If $pf_x \in P_A$, then pf_x is associated with satisfaction of some goal $g(pf_x)$.
4. The set of goals known to agent A is $G_A = \{g \mid g = g(pf_x) \text{ for some plan fragment } pf_x \in P_A\}$.
5. For each goal g in G_A , there is an associated set of plan fragments $pf_{S_g} = \{x \mid x \in P_A \text{ and } g = g(x)\}$.
6. $copies(r_i)$ denotes the number of copies of resource r_i available for use by agent A.
7. $resources(pf_x)$ denotes the resources required to execute plan fragment pf_x .
8. $r_i(pf_x)$ denotes the number of copies of resource r_i needed by plan fragment pf_x .
9. A set of plan fragments in Agent A, $P = \{pf_1, \dots, pf_n\}$ is said to be **compatible** if $\sum_{k=1}^n r_i(pf_k) \leq copies(r_i)$ for all i and $g(pf_j) \neq g(pf_k)$ for $j \neq k$.
10. A **maximal compatible set of plan fragments in A relative to pf_x** is a maximal subset of $S_r = \{P \mid P \text{ is a compatible set of plan fragments and } pf_x \in P\}$.

The conflict set for plan fragment pf_x indicates the minimal impact (locally) of a choice to execute pf_x . The conflict set for pf_x can be constructed by considering each maximal set M of mutually feasible plan fragments (*including* pf_x) known to an agent. For each such set, M , the complement of M is an element of the conflict set for pf_x .

More formally, the **Conflict Set** for plan fragment pf_x is constructed as follows: Let $X = (P_A - pfs_g) \cup \{pf_x\}$, where $g = g(pf_x)$. For each maximal compatible subset M of plan fragments in A relative to pf_x , the set $X - M$ is a member of the conflict set for pf_x . Thus, $CS_{pf_x} = \{c \mid c = X - M, \text{ where } M \text{ is a maximal compatible subset of plan fragments in } A \text{ relative to } pf_x\}$.

To illustrate this formalism, we compute the conflict set for D-b in our example scenario. The maximal compatible subsets of plan fragments in D relative to D-b are: $\{D-b, D-e\}$, $\{D-b, D-g\}$, and $\{D-b, D-i\}$. Thus the conflict set for D-b is:

$$\{\{D-d, D-f, D-g, D-h, D-i\}, \{D-d, D-e, D-f, D-h, D-i\}, \{D-d, D-e, D-f, D-g, D-h\}\}$$

Thus, if agent D selects plan fragment b in partial satisfaction of goal $g1$, then the only other choices *locally* compatible with this selection are $D-e$ or $D-g$ or $D-i$. This is also expressed in the conflict set by the three elements each of which is a set of plan fragments which are collectively in conflict with the choice of plan fragment b .

We are concerned with the conflict set because the conflict set for a plan fragment gives information as to the negative impact of executing that plan fragment. The maximal compatible subsets, on the other hand, indicate maximal sets of feasible choices that are available. There is no reason to believe that an agent should choose some one of these maximal subsets for execution. Indeed, a given agent might never participate in system satisfaction of some of the global goals. (This can be seen in the example scenario by observing that all four global goals can be satisfied through choice of $g1p3$, $g2p3$, $g3p1$, and $g4p1$. Agent D is only involved through partial satisfaction of $g2$ and $g3$.)

Though the view of the conflict set as being formed using the complements of maximal feasible sets is intuitively appealing, when the problem is underconstrained it is computationally more attractive to view conflict relative to pf_x in a dual form: as the collection of minimal mutually infeasible sets of plan fragments, given that plan fragment pf_x is to be executed.

Three significant observations can be made concerning the conflict set of plan fragment pf_x . First, the complement of each element of the conflict set is indeed a maximal feasible set. Secondly, the agent will be *compelled* to forego execution of all plan fragments in *some* element of the conflict set if it chooses to execute plan fragment pf_x . The local impact of a decision can thus be related to the size of elements in the conflict set. Finally, representation of impact in the form of a conflict set seems to provide a substantially more compact form of representation that can be more efficiently used in reasoning than many others.

The conflict set for a plan fragment reflects the impact of executing that plan fragment *at the level of mutually infeasible sets of plan fragments*. It is often necessary to reason about the impact that executing a particular plan fragment would have on the potential satisfaction of other goals.

The **Exclusion Set** for a plan fragment, pf_x , is a collection of sets of goals, one of which must be abandoned if pf_x is selected for execution. Thus, if the agent selects plan fragment pf_x then one of the elements of the exclusion set is a set of goals that *cannot* be satisfied through action on the part of this agent. The exclusion set is defined as follows:

For each $s \in CS_{pf_x}$, we define $g_s = \{g \mid pfs(g) \subseteq s\}$. Thus g_s , for an element s of the conflict set, is the set of goals that cannot be satisfied locally if plan fragments in s are eliminated from consideration. We let $G = \{g_s \mid s \in CS_{pf_x}\}$ and define the **exclusion set for plan fragment pf_x** , ES_{pf_x} , as the collection of the minimal subsets of G .

Returning to our example, we compute the exclusion set for D-b. The conflict set for D-b has three elements. Using the definition of g_s , we see that

- $g_{D-d,D-f,D-g,D-h,D-i} = \{g3, g4\}$
- $g_{D-d,D-e,D-f,D-h,D-i} = \{g2, g4\}$
- $g_{D-d,D-e,D-f,D-g,D-h} = \{g2, g3\}$

Thus, $G = ES_{D-b} = \{\{g3, g4\}, \{g2, g4\}, \{g2, g3\}\}$.

A choice by agent D to execute plan fragment D-a compels agent B to forego local action in partial satisfaction of two of the other three global goals about which it has local knowledge. *Which* two of the three should be abandoned is dependent on decisions made elsewhere.

The exclusion set exposes relationships between plan fragments and goals. It is often desirable to detect and reason about mutually infeasible goals. The relationship of infeasibility is a very strong one. Goal $g1$ is (locally) infeasible with goal $g2$ if each of the (local) plan fragments for $g1$ has $g2$ in every element of its exclusion set (and conversely). When two goals are (locally) mutually infeasible, an agent knows that it cannot act to satisfy both goals, due to local constraints. Once exclusion sets have been determined, infeasibility is not difficult to detect.

The three types of relationships we have discussed are all rooted in local constraints. Conflict, exclusion, and infeasibility are essentially concepts which would not be particularly significant were it not for the constraints on joint execution of plan fragments that exist locally. Although the concept of conflict does not appear to propagate in a meaningful manner, exclusion does. The key element in this propagation lies in the observation (which we have made before) that a choice on the part of one agent to satisfy a goal through execution of a specific plan fragment constrains the set of remaining choices that are available to other agents for satisfaction of that goal.

As we have seen, the construction of exclusion sets allows us to assess the impact of executing of a plan fragment that is due to local conflict. In addition, we would like to know how the conflict associated (locally) with execution of a plan fragment affects the ability of other agents to satisfy their goals. The **Induced Exclusion Set** is our mechanism that provides a vehicle for propagating this information by capturing the essence of the impact that local decisions have nonlocally.

In the discussion which follows, we assume that in a distributed environment one agent does not have knowledge concerning another agent's internal state. It specifically does not have any knowledge about resources over which it has no control. The agent must therefore gain knowledge about the impact its choice has on other agents *from those agents*, directly or indirectly.

The **Induced Exclusion Set** for a plan fragment, pf_x , in Agent A, is a collection of sets of goals, one of which must be abandoned by one or more non-local agents if Agent A executes pf_x . The induced exclusion set for pf_x , IE_{pf_x} , is defined in the paragraphs which follow.

Let $X_{pf_x} = \{pfi \mid pfx \in P_A, pfi \notin P_A, resources(pfx) \cap resources(pfi) \neq \phi, \text{ and } g(pfx) = g(pfi)\}$. Thus, each individual plan fragment in X_{pf_x} is a non-local plan fragment which may connect directly with pfx (via a shared resource) in some global plan.

For each agent, K, with plan fragments in X_{pf_x} we must determine the contribution to the induced exclusion set for pfx due to constraints known by agent K. For each plan fragment $p \in X_{pf_x} \cap P_K$, we therefore let

$$e_p = \{\epsilon \mid \epsilon = es \cup ie \text{ for } es \in ES_p \text{ and } ie \in IE_p\}$$

Notice that each e_p is a set of sets, each of whose members reflects potential conflict due that could arise if plan fragment p is selected by agent K. In this construction, each es represents a contribution to e_p that reflects constraints local to agent K, while each ie denotes a contribution that agent K has received from other agents relative to plan fragment p . For this reason, it is necessary to combine these contributions into a single element, $E_{K,pfx}$, that may be propagated to Agent A. $E_{K,pfx}$ is defined as the collection of minimal subsets of $\cup e_p$ for $p \in X_{pf_x} \cap P_K$.

Continuing the definition, the **induced exclusion set for plan fragment pfx**, IE_{pf_x} is the collection of the maximal subsets of $E = \cup E_{K,pfx}$. This definition of IE_{pf_x} permits incremental construction of induced exclusion sets under the assumption that initially $IE_{pf_x} = \phi$ for all plan fragments.

Once again, returning to our example, we compute the induced exclusion set for C-a. Observe that plan fragment C-a matches (in D) with either D-a or D-b and in A with A-a, so that $X_{C-a} = \{A-a, D-a, D-b\}$. As we have seen, the exclusion set for D-b is $\{\{g3, g4\}, \{g2, g4\}, \{g2, g3\}\}$. Coincidentally, the exclusion set for D-a is the same as that for

D-b, while for A-a the exclusion set is $\{\{g2\}, \{g3\}, \{g4\}\}$. The set E used in computing the induced exclusion set for C-a is the union of the exclusion sets just mentioned, so $E = \{\{g2\}, \{g3\}, \{g4\}, \{g3, g4\}, \{g2, g4\}, \{g2, g3\}\}$. The induced exclusion set for C-a is the set of maximal subsets of E, so $IE_{C-a} = \{\{g3, g4\}, \{g2, g4\}, \{g2, g3\}\}$.

Intuitively, this is telling agent C that agent A is forced to forego one other goal if C-a is chosen and agent D is forced to forego two of the other three goals if C-a is selected. Each nonlocal agent transmits a minimal set of exclusions it knows about. Clearly, agent D reports more extensive nonlocal impact, and the construction of the induced exclusion set via maximal subsets reflects this impact.

The induced exclusion set is incrementally built during negotiation. When one agent (agent A) requests information about the impact of executing plan fragment pf_x on another agent (agent B), agent B attempts to summarize all the knowledge it has about that impact. This knowledge is initially found in the exclusion sets of each of its plan fragments which coordinate with plan fragment pf_x . As has been mentioned, the induced exclusion set in agent A for plan fragment pf_x is empty initially. As nonlocal knowledge becomes available, this set is augmented in the obvious way. Given sufficient time, an agent can acquire knowledge about the system wide impact of executing each of its plan fragments. It does so, however, *without* the exchange of detailed information concerning resource availability in the system. It is not difficult to show that incremental construction of the induced exclusion set for a plan fragment can be managed so that it converges after no more than $2n$ exchanges of information, where n is the number of agents in the system.

4.4.2 Computation of Conflict

Most of the work involved in providing an agent with a reasonable level of understanding regarding the impact of local decisions lies in computation of conflict within each agent. In this section, we give two procedures for carrying out this computation. The first takes the view that the conflict set relative to a plan fragment is the collection of sets determined by finding complements of maximal feasible sets. The second constructs a representation of conflict directly as the collection of minimal infeasible sets. Both computations yield sets that provide the same information relative to exclusion and infeasibility.

Strategy I

For every plan fragment pf_i in pf_{s_A} :

1. compute-maximum-compatibles(*reservation-list* $goals_A - g(pf_i)$ pf_i)
2. take complement of each maximum compatible with respect to $pf_{s_A} - pf_{s_{g_i}}$, where $g_i = g(pf_i)$.

The function compute-maximum-compatibles is defined as follows:

```

compute-maximum-compatibles (reservation-list goals compatible-set)
  if (null goals)
    add compatible-set to maximum compatible sets and delete subsets
  otherwise
    for every plan fragment  $pf_x$  for  $g_x$ , the first goal in goals,
      if  $pf_x$  does not exceed resource availability based on reservation-list,
        then for every resource  $r_x$  required by  $pf_x$ ,
          add 1 to reservation-list entry for  $r_x$ 
          add  $pf_x$  to compatible-set
    compute-maximum-compatibles (reservation-list (goals -  $g_x$ ) compatible-set)

```

This algorithm computes the conflict set by finding maximal compatible sets and their complements. Its complexity is bounded by

$$|P_A| * [\max(|pf_{s_g}|) * \# \text{ of resources}]^{|G_A|}$$

In fact, our experiments indicate that this expression does not represent a tight bound when the scenario represents an overconstrained situation. Since there are many fewer feasible sets when the problem is overconstrained, this is not surprising. The second procedure, given below, computes minimal infeasible sets (under the assumption that plan fragment pf_x is selected). It is not hard to see that the second strategy is more efficient when the problem is underconstrained, as major portions of the algorithm are not exercised when there is no hard resource constraint to test. In the worst case, Strategy 2 is exponential in (number-of-plan-fragments * number-of-resources-known).

Strategy 2

1. For each resource, r , required by plan fragment pf_x :
 - (a) for each goal, let $S_r(g) = \{\text{plan fragments for goal } g \text{ that require resource } r\}$
 - (b) let $S_r = \{S_r(g_i) \mid g_i \in G_A - g_{pf_x}\}$
 - (c) if $\text{copies}(r) \leq |S_r|$ then
 - i. define $\text{CONF}(r)$ as

$$\{s \mid s = S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_n}, \text{ where } n = |S_r| - \text{copies}(r) + 1 \text{ and } S_{i_k} \in S_r.\}$$
2. Construct $\text{CONF} = \{c \mid c = c_1 \cup c_2 \cup \dots \cup c_m \text{ where } c_i \in \text{CONF}(r_i) \text{ and } r_i \text{ is known to agent } A\}$
3. Conflict is represented by the collection of minimal subsets of CONF .

4.4.3 Status and Future Directions

We have discussed formalisms that permit an agent in a distributed planning system to gain knowledge about the interaction between consequences of its local actions and constraints existing elsewhere in the system. These formalisms define an abstraction hierarchy representing impact at the level of plan fragments, at the level of plan fragments relative to goals, and at the level of goal interactions. This theory provides a basis for agents to begin negotiation by exchanging enough information to make good heuristic local decisions. Future work will involve the implementation of these ideas as well as the further development of multistage negotiation. Extensions of these formalisms that are useful in dynamic domains requiring incremental plan generation are also being formulated.

4.5 Maintaining Consistent Beliefs in a Shared Knowledge Base

In the previous sections we have examined cooperation among agents involving message passing. At the local level cooperation may also be achieved by sharing knowledge concerning the current state of the communications network. Inferences of one agent are shared with the others, in a central knowledge base. The shared knowledge base is managed by the Knowledge Base Manager (KBM).

The KBM has the responsibility for managing the knowledge base. Within the KBM, some type of truth maintenance system must be active. Its tasks involve regulating each problem solver's beliefs in a consistent manner, as well as providing a means by which one agent can easily share its inferences with the others. While the truth maintenance system will not directly address resolution of inconsistencies across agents, it must provide an efficient mechanism by which the KBM can recognize inconsistencies among agents.

The MATMS (Multi-agent Assumption-based Truth Maintenance System) has been developed to manage a knowledge base shared by multiple problem solvers. Each problem solver has its beliefs "independently" managed in a manner similar to that provided by a conventional truth maintenance system. That is, every problem solver in the system can add and retract beliefs from its belief set and the MATMS will ensure that the belief set remains sound and complete — every inference and only those inferences derivable from the set of assumptions in the belief set are included in the belief set. A fundamental difference in this system as opposed to conventional truth maintenance systems is that an inference provided to the MATMS is also made available to other problem solvers — any agent which believes the assumptions upon which an inference is based has the inference placed in its belief set.

4.5.1 The Role of Truth Maintenance

During the course of normal problem solving activity, an agent, or problem solver, may formalize or make use of assumptions. Assumptions can be divided into three types: default assumptions ("unless there is evidence to the contrary, assume that the employee is getting paid"), suppositions ("suppose that the employee is not getting paid"), and new observations of the current state of the world ("the employee is not getting paid"). The common bond among the three is a *believability* which is not dependent upon any other belief.

Every inference drawn by a problem solver can ultimately be traced back to a set (or sets) of assumptions. As opposed to assumptions, the believability of an inference is dependent upon the believability of the assumptions. If an assumption set upon which an inference is based is currently believed by a problem solver, then that inference should also be believed, regardless of the type of the assumptions involved.

The assumptions and the inferences based upon these assumptions with which a problem solver is operating are referred to as the current *belief set* of the problem solver. Every time a problem solver draws an inference, makes an assumption, retracts an inference, or retracts an assumption, it changes its belief set. When a problem solver changes its belief set, many difficulties arise. How much of what was believed before the change can still be believed after the change? This is commonly called the *frame problem*. In more general terms, the frame problem is "... the inability to model side effects of actions taken in the world by making corresponding modifications in the database representing the state of the world." [1] For example, which beliefs must be removed, and which beliefs can remain, when a particular assumption is removed?

Another problem arises when the agent introduces a belief to the knowledge base which conflicts with one which is already present. How can the intentions of the problem solver be correctly recognized? Suppose an agent wished to override a default assumption. For example, imagine that the default *is-a* attribute of each object in the knowledge base is *square*. If object RE33 is in the knowledge base, its default *is-a* attribute is *square*. If the agent realizes that RE33 is actually a circle, then *object RE33 is a circle* should be allowed to automatically override the default assumption. A belief which the problem solver has explicitly made should be allowed to override a default assumption. But this is the case if the problem solver introduces a belief that is inconsistent with another which is not a default assumption. If after asserting that object RE33 is a circle, the agent asserts *object RE33 is a triangle*, then the agent has explicitly made two assertions which are inconsistent. In general, when a problem solver adds a belief which contradicts an existing belief, does the belief set become inconsistent, or should the most recent belief simply override the belief with which it is inconsistent?

If a belief set of an agent does become inconsistent, classical first order logic suggests everything can be proven, and everything can be disproven, so the knowledge base is

essentially worthless. It seems evident that only the attributes of those objects which are logically affected by the inconsistency should be questioned. Consider a knowledge base which contains *object RE33 is a triangle* and *object RE33 is a circle*. If there exists no logical connection between object *RE33* and object *hy77*, then the shape of object *hy77* should not be suspect.

To address problems associated with changing beliefs, truth maintenance systems [10, 6, 27, 25] have been developed for use with single problem solvers. Whenever the problem solver adds or retracts a belief, the truth maintenance system is invoked to manage the beliefs. For instance, when an assumption is removed, the system can determine (after an indeterminate amount of time) which inferences have to be removed because they depended, either directly or indirectly, on the acceptance of the assumption. In addition, if two beliefs are inconsistent, the entire knowledge base is not rendered useless. Rather, the truth maintenance system can determine which subset of beliefs in the knowledge base are inconsistent — the rest of the knowledge base remains consistent. Default assumptions are also handled appropriately. That is, default assumptions are overridden when necessary, and “come back” if a problem solver should retract the belief which overrides it. As an example, consider the case when there exists a default assumption *Car X has four wheels* and a problem solver asserts *Car X has six wheels*. If the problem solver retracts *Car X has six wheels*, then the truth maintenance system would restore the belief that *Car X has four wheels*.

In addition to solving these problems, the truth maintenance system records every inference made. Problem solving becomes more efficient because every inference need only be made once. Suppose a problem solver is involved in a series of long, expensive computations. If it decides to stop the computations to perform another task, upon returning to the original task, without a truth maintenance system, the problem solver might be forced to start the task from the beginning again, thus having to recompute many results. With the truth maintenance system, the problem solver can continue essentially where it had halted. When the problem solver re-asserts the assumptions which were in its belief set while it was performing the original task, the truth maintenance system restores any inference the agent had in its belief set while performing the task.

Our work is concerned with a distributed problem solving environment in which there are a number of agents cooperating by passing messages to each other requesting action and by sharing inferences in a central knowledge base. Each agent's belief set is kept within the central knowledge base. When an agent adds an inference to its belief set, that inference is shared with the other agents because the validity of an inference in the central knowledge base depends only upon the validity of its preconditions. As an example, suppose the following rule exists in one of the agents:

$$A \wedge B \Rightarrow C$$

The rule can be interpreted as “If *A* and *B* are believed, then *C* is believed.” Once the

problem solver enters this knowledge into the shared knowledge base, any problem solver which believes *A* and *B* will believe *C*.

There are many difficulties in managing a shared knowledge base using the techniques that a conventional truth maintenance system employs to manage a knowledge base accessed by only one problem solver. Every problem present in the single agent environment is also present in the multi-agent environment, and many additional difficulties are encountered that are due to the distributed aspects of the problem solving system. First, a single agent truth maintenance system organizes its knowledge base so that the problem solver "sees" only those beliefs which are in its current belief set. A truth maintenance system in a multi-agent environment must perform this task for each of the agents. In doing so, the system must handle the possibility that one agent may believe a value for a piece of knowledge, while another may believe the opposite. To illustrate this phenomenon, consider the fact that the use of supposition by one agent should not modify the beliefs of another agent. For example, imagine that one agent believes that Resource X is available. If another agent, while engaging in hypothetical reasoning, adds the supposition that Resource X is unavailable, the first agent should still believe that Resource X is available. The truth maintenance system must also handle situations in which one piece of knowledge may be currently believed by any number of agents and at the same time disbelieved by any number of other agents.

Another difficulty arises when one realizes that the assessment of the current state of the world is achieved through the combined efforts of all agents. That is, part of each agent's task is to "fill in" the incomplete portions of an overall assessment in order to aid one another. For the most part problem solvers will agree with each other, but there will be times when two problem solvers disagree on a piece of knowledge in the knowledge base. For example, *PS*₁ (Problem Solver 1) might believe Resource X is available, and *PS*₂ might believe Resource X is unavailable. A discrepancy of this type could cause problem solving to diverge beyond the point of recovery. It is important that *inconsistencies between problem solvers' assessments of the current state of the world be recognized, and an attempt made to resolve them*. Often there will be times when the "differences of opinion" cannot be resolved. At these times, the discrepancies must be permitted to stand, hopefully to be resolved in the future.

Finally, arguments concerning the importance of efficiency in a truth maintenance system are significantly magnified when comparing a multi-agent environment to a single agent environment. The truth maintenance system managing a knowledge base shared by multiple problem solvers must be prepared to shift its focus of attention from one problem solver to another quickly, even if just to answer queries. This problem is not encountered in a single agent environment - the truth maintenance system is always concerned with the single agent. Therefore, even if the other problems in managing a knowledge base shared by multiple agents are addressed adequately, the system might be too slow to be useful in any practical problem solving system.

4.5.2 Comparisons with Existing Truth Maintenance Systems

Existing truth maintenance systems have failed to address the need for a system in which multiple problem solvers' inferences are controlled by a single truth maintenance system. However, because we have adopted much of the terminology of conventional truth maintenance systems, and the MATMS borrows heavily from concepts developed in existing truth maintenance systems, the two dominant classes of truth maintenance systems are presented.

Doyle's Truth Maintenance System (TMS) [10] was the first domain independent truth maintenance system. Doyle proposed that reasons for believing or using each belief, inference rule or procedure be recorded. This allows new information to displace previous conclusions and a consistent knowledge base to be kept. In the TMS, each belief in the knowledge base is explicitly marked as either IN or OUT, where IN means that the belief has at least one currently acceptable reason, and OUT means that it has no currently acceptable reason for belief. Given a belief or a justification for an existing belief, the job of the TMS is to determine the belief status of each of the beliefs in the knowledge base, thus retaining one consistent knowledge base.

Doyle's TMS defined the class of *justification based* truth maintenance systems. That is, the status of each belief is determined by searching through each justification until reaching a set of assumptions. If the set of assumptions are valid (or believed), then the belief is valid. Considering that these justifications are examined for each belief in the knowledge base, and any particular chain of inferences which eventually leads to a particular inference in question may be long, updating the knowledge base may require a long period of time.

The TMS was deemed inappropriate for a multi-problem solver environment because it maintains only one belief set at a time. Given any point in time, the TMS has one set of beliefs which are IN and one set which are OUT. Switching belief spaces is cumbersome because the status of each belief has to be directly recomputed. As we have observed, in a multiple problem solver environment, the truth maintenance system must be able to switch belief spaces quickly.

De Kleer, in his Assumption based Truth Maintenance System (ATMS)[6, 7], recognized this problem also, though not for the same reasons. De Kleer was interested in hypothetical reasoning, in which assumptions are made often, and results compared against the assumptions. Therefore, the ATMS was designed explicitly to switch belief sets efficiently. The ATMS is the foundation for the MATMS, and as such will be discussed in much greater detail.

In order to create a system which could switch beliefs sets quickly, de Kleer recognized that an inference is ultimately dependent on a set (or sets) of assumptions. That is, an inference may be derived from other inferences, and these inferences may have been derived from other inferences, but eventually this trace will find assumptions only. Therefore,

when a problem solver changes its belief set, the justifications of the inferences in the previous belief set do not have to be traced to determine if they still have valid support. Rather, each inference could be tested to see if the assumptions upon which it is based are still present in belief set.

In the ATMS the entire set of beliefs is divided into sets called *contexts*; each context represents a belief set. Essentially a context is defined by its assumption set, which is called an *environment*, and includes all inferences which can be derived, either directly or indirectly, from the environment.

As a context is associated with a particular set of beliefs, each belief is associated with a list of contexts to which it belongs. Much of the ATMS's work involves ensuring that each belief's label — the set of environments from which the node is derivable — is consistent, sound, complete, and minimal with respect to the justifications. (A label is consistent if each environment in the label is consistent, sound if every environment can derive the belief, complete if every way to derive the belief is included in the label, and minimal if no environment in the label is a superset of another in the label.) Labels must be kept this way primarily for efficiency.

There are three features which make the ATMS more appropriate than the TMS for the multi-agent system described. First, the ATMS maintains more than one belief set at a time by maintaining multiple contexts. Each agent in a multi-agent system could conceivably be operating with a different set of beliefs, so it is essential that a truth maintenance system handling their beliefs have the ability to maintain multiple belief sets. Second, a problem solver using the ATMS can change its belief set much more swiftly than if it were using the TMS, because the TMS is often forced to perform costly tracing in order to reassign belief status to each of the beliefs in the knowledge base. Switching belief sets is also quicker in the ATMS because the new belief set could already be defined. For example, suppose a problem solver utilizing the ATMS adds assumption X to its belief set. After the ATMS calculates the problem solver's new context, the problem solver retracts assumption X. When this happens, the ATMS simply returns the problem solver to its previous context. The TMS in this situation would have to reassign belief status to each belief in the knowledge base, only to return the problem solver to its original belief set. The third reason is that the ATMS handles multiple derivations for an inference better than the TMS. (Although it still does not handle it very elegantly, it is still far more capable than the TMS.) In a multi-problem solver environment, where inferencing schemes are numerous, a belief is likely to be inferred from more than one set of beliefs. A truth maintenance system in a multi-agent system must be able to determine that, if support for an inference is removed, there may be other support which keeps the belief in the current context.

The major drawback to the assumption based systems is the computation of support for each inference in the knowledge base. When an inference is added to the justification based system, only the immediate preconditions of the inference are recorded (this is what

makes the system *justification based*). Therefore, adding inferences is not difficult. In an assumption based system, the immediate preconditions must be traced until assumption sets are found. In situations involving inferences already in the knowledge base that are themselves immediate preconditions to other beliefs, this causes inefficiency. Each belief which is either directly or indirectly influenced by a "new" inference must have the assumption set upon which it is based recomputed. Therefore, adding inferences might require a significant amount of computation.

An interesting observation is how each truth maintenance system handles default assumptions. The focus of each is how to make default assumptions "come back" when a belief which previously overrode the default is retracted. Both the TMS and the ATMS have chosen to include default assumptions explicitly in the belief set of the problem solver. If the number of default assumptions is large, then each system is hampered.

Martins and Shapiro in [25] present a similar comparison of assumption based and justification based truth maintenance systems. They also present a useful example of how an assumption based system manages beliefs as opposed to a justification based system.

4.5.3 Functional Design of The MATMS

The MATMS has been designed for use in a system involving any number of agents sharing a central knowledge base. In such a system, each problem solver registers its beliefs with the MATMS. An inference is registered along with the beliefs upon which it directly depends, and the job of the MATMS is to maintain multiple belief sets. Thus the MATMS is responsible for placing an inference in any belief set which contains the assumptions upon which it is based, informing an agent when its belief set becomes inconsistent, changing the belief set of an agent efficiently, and switching its focus of attention from one agent to another quickly.

4.5.3.1 Definitions As a matter of convention, the operators of propositional logic are utilized from this point on. Specifically, the logical connectives of interest are \wedge (and), \vee (or), \Rightarrow (implication), \neg (not), and \perp (false). Some examples are $A \Rightarrow B$ (A implies B), $C \wedge D \Rightarrow \perp$ (the quantity C and D implies false), and $E \vee F \Rightarrow \neg G$ (E or F imply not G). Shorthand notation will be used for \wedge : $C \wedge D \Rightarrow \perp$ will usually be written as $CD \Rightarrow \perp$, and $((A)(B))$ means $(A) \vee (B)$.

A *proposition* is the MATMS datum that represents a piece of knowledge which a problem solver has told the MATMS. Each proposition is unique. Example propositions are "Jim is a golfer." "Radio R1 has failed," and "The Celica is being repaired."

Each proposition is attached to a *belief*, the basic datum on which the MATMS operates. Beliefs are explicitly divided into two categories: *assumptions* and *inferences*. An *inference* is a belief whose validity depends upon other beliefs. For instance, if an agent

has the rule $(A \wedge B) \vee (C \wedge (E \vee F)) \Rightarrow G$ and tells the MATMS that it believes G because of it believes C and E , then G is an *inference* because it is only believed in this case if C and E are believed. An *assumption* is a belief whose validity does not depend upon the acceptance of any other belief. Assumptions are divided into three types: default assumptions, suppositions, and new observations concerning the state of the world. The last two types will be referred to as *non-default assumptions*.

For discussion purposes, we represent inferencing mechanisms as rules, but they do not necessarily have to be interpreted in a strict sense. For instance, $A \Rightarrow B$ is merely meant to represent that inference operations can be activated in the presense of A to draw the logical conclusion B . The exact meaning of an inference rule can be interpreted as "in the presense of certain beliefs, another belief is implied, no matter which other beliefs are present." In other words, if $A \Rightarrow B$, B is included in any set which contains A , such as (AB) and (AEF) .

A *justification* for a belief is the set of beliefs which must be present for its validity. An assumption has no justification (the justification is nil), whereas an *inference* must have at least one justification, and may have many. The justification for an inference is comprised of two components: its immediate preconditions (the beliefs from which it can directly be inferred), and the assumptions upon which it is based (the assumptions that it ultimately depends upon). For example, considering $A \Rightarrow B$, the assumptions that B is based upon are the same as its immediate preconditions — $((A))$. If $B \Rightarrow C$, the assumptions that C is based upon are again $((A))$, but the immediate preconditions necessary for its derivation are $((B))$. If $C \Rightarrow D$ and $E \Rightarrow D$, the assumptions that D is based upon are $((A)(E))$ and the immediate are $((C)(E))$. (Recall that $((C)(E))$ should be interpreted as $(C) \vee (E)$.)

At any point in time, each problem solver has a belief set — a set of assumptions and inferences which have been derived from those assumptions. Some of the beliefs are default assumptions, some are non-default assumptions, and some are inferences. An *environment* is a unique set of non-default assumptions under which a problem solver has operated. Environments are created incrementally — whenever a problem solver retracts or adds a non-default assumption, an environment is created if one does not exist that matches the new set of assumptions.

A *context* is an environment and all inferences which have been derived from the environment; hence it is a group of beliefs. For every environment, there is exactly one context, and a context is created each time an environment is created. If a problem solver has never worked with a particular grouping of assumptions, the MATMS does not have this set of assumptions listed in an environment, so there is no context for this group.

A *premise* is a rule which states that a set of propositions are inconsistent. Beliefs with these propositions are therefore inconsistent. Examples of premises are:

$$\neg(\text{"Fred is dead"} \text{"Fred is alive"})$$

or
¬("a man is working" "a man is resting")

As is evident from these examples, premises can be specific or general. It is important to observe that a premise has no meaning until a belief is supplied to the knowledge base which has as a proposition one of the propositions named in the premise. A set of beliefs is said to be *contradictory* if the propositions of the beliefs violate any premise. The set will be referred to as an incompatible belief set, or *incompatible*.

A context is *inconsistent* if it contains contradictory beliefs. In other words, the context is inconsistent if any subset of its beliefs is an incompatible belief set.

The MATMS monitors the belief set of a problem solver by recording the context in which the problem solver is currently working. A problem solver is working in a particular context if it has explicitly told the MATMS that it holds all of the assumptions defining the context.

A problem solver *retracts* an assumption when it asks the MATMS to remove the assumption from its current belief set. Note that neither the assumption is actually removed from the knowledge base, nor the inferences which have been registered as depending upon it. The problem solver is just placed in a new context. Thus, the problem solver *switches contexts* whenever it adds or deletes a non-default assumption.

4.5.3.2 Data Structures The MATMS is a frame-based system in which there are five basic types of objects: beliefs, inferences, assumptions, contexts, and incompatibles. Our discussion of the data structures of the MATMS begins with belief. Each belief has slots *proposition*, *contexts in*, and *influences*. *Contexts in* is a list of contexts in which the belief holds. *Influences* is a list of beliefs which this belief directly influences. The frame for belief, as well as the other frames, is depicted in Figure 18.

The beliefs, as previously mentioned, are explicitly divided into two classes at any point in time — assumptions and inferences. Each class inherits from the belief frame. The inference class, however, also includes the slots *assumptions based upon* and *immediate preconditions*. *Immediate preconditions* is a list of sets of beliefs from which inference rules were applied to produce the resultant inference. Each belief in each of these sets has this resultant inference as a member of its *influence* slot. If the length of *immediate preconditions* is greater than one, multiple derivations for the inference have been provided to the MATMS. *Assumptions based upon* are the minimal² sets of assumptions from which the inference has been derived.³ If the environment of a context is a superset of any set in

²Minimal in terms of set inclusion. For instance, the minimal sets of ((AB)(ABC)(DE)) are ((AB)(DE)). (ABC) is not included because it is a proper superset of (AB).

³The *immediate preconditions* slot is not minimal because it is necessary to maintain records of every way the inference has been derived in case a derivation is retracted by the problem solver. This is

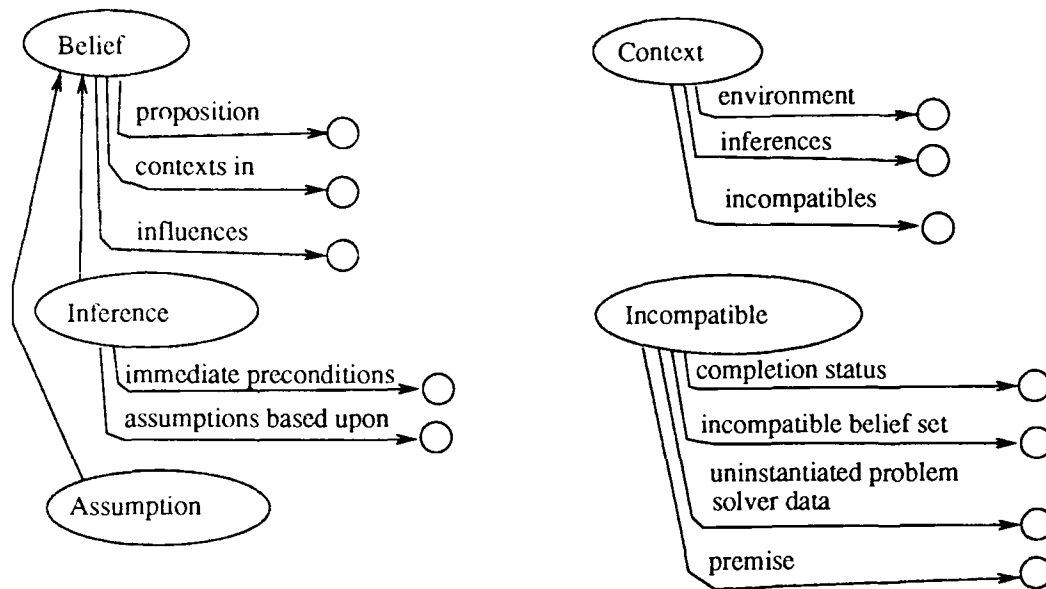


Figure 18: Net Connecting MATMS Frames

assumptions based upon, then the inference is included in the context. *Assumptions based upon* is constructed by tracing the chain of *immediate preconditions* until assumptions are found.

The frame for a context has *environment*, *inferences*, and *incompatible belief sets* slots. *Environment* is the unique set of non-default assumptions which defines the context. *Inferences* is the list of all inferences which have been derived from the environment. An inference is included in *inferences* if it is based upon at least one set of assumptions of which at least one is non-default, and all of the non-defaults are included in *environment*. The *incompatible belief sets* slot contains a list of sets of beliefs in the context which have been previously defined in a premise as being incompatible. Each belief in *incompatible belief sets* is a member of *environment* or *inferences*. By definition, if *incompatible belief sets* is not empty, the context is inconsistent.

At this point, it is appropriate to present an example to illustrate the data structures. Imagine a simple two-agent system consisting of Agent₁ and Agent₂, and suppose that part of each agent's task is to plan the schedule and activities of students. The following scenario occurs:

1. Initially, each agent is working with only the default assumptions. This "null" context will be referred to as *C0*. (Alternative symbolic representations are given so that a interpretable table can be presented at the end.)
2. Agent₁ adds the assumption "Economics 337 will be held in room 445 of Hamilton

discussed in "Write Operations."

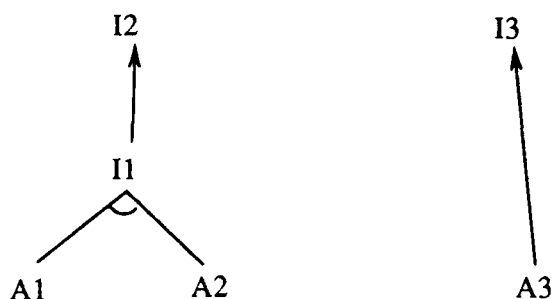


Figure 19: Inference Tree for Example Illustrating Data Structures

Hall 6/5/88" (*A1*) to its belief set. (There is a default assumption which states that Economics 337 is usually held in Morley Hall.) This causes the MATMS to create a new context, *C1*. Agent₁ is then placed in *C1*.

3. Agent₁ adds the assumption "Hamilton Hall is further from the dorms than Morley Hall" (*A2*) to its belief set. This causes the MATMS to create a new context, *C2*. Logically, Agent₁ is then placed in *C2*.
4. Agent₁ adds the inference "It will take longer than normal to go to class tomorrow" (*I1*) to its belief set. The inference is based upon "Economics 337 will be held in room 445 of Hamilton Hall tomorrow" (*A1*) and "Hamilton Hall is further from the dorms than Morley Hall" (*A2*). The MATMS adds the inference to every context which contains *A1* and *A2* — only *C2*.
5. Agent₁ adds the inference "A person in Economics 337 should leave early for class tomorrow" (*I2*) to its belief set. The inference is based upon only "It will take longer than normal to go to class tomorrow" (*I1*). The MATMS adds the inference to every context which contains *I1* — again only *C2*.
6. Agent₂ adds the assumption "Economics 337 will be cancelled tomorrow" (*A3*) to its belief set. This causes the MATMS to create a new context, *C3*. Agent₂ is then placed in *C3*.
7. Agent₂ adds the inference "A person in Economics 337 should play golf tomorrow" (*I3*) to its belief set. The inference is based upon only "Economics 337 will be cancelled tomorrow" (*A3*). The MATMS adds the inference to every context which contains *I1* — only *C3*.

Figure 19 shows the inference tree for this knowledge base, and Table 7 represents the data structures of the MATMS at this point in problem solving. In the table, "assumptions" is shorthand for "assumptions based upon," and "preconditions" is short for "immediate preconditions."

	C0	C1	C2	C3
environment	()	(A1)	(A1 A2)	(A3)
inferences	()	()	(I1 I2)	(I3)
incompatibles	()	()	()	()

	A1	A2	A3	I1	I2	I3
contexts in	(C1 C2)	(C2)	(C3)	(C2)	(C2)	(C3)
influences	(I1)	(I1)	(I3)	(I2)	()	()
assumptions	*	*	*	((A1 A2))	((A1 A2))	((A3))
preconditions	*	*	*	((A1 A2))	((I1))	((A3))

Table 7: Example Data Structures

Continuing with the discussion of the data structures, as noted in the previous section, premises may or may not be used by MATMS. For example, consider a premise “a man cannot be working and resting at the same.” If a problem solver never supplies a proposition pertaining to a particular man and his work status, then this premise will never be used. But suppose a problem solver supplies “Jim is working” after supplying “Jim is a man.” The MATMS must recognize that “half” of the premise has been supplied. If the other half is supplied -- “Jim is resting” -- then the instantiation of the premise will be complete: the beliefs representing “Jim is working” and “Jim is resting” form an incompatible. Note that this premise can be instantiated many times.

The data structure for *incompatible* is used to capture this notion of how incompatible belief sets are created. The slots are *completion status*, *incompatible belief set*, *uninstantiated problem solver data*, and *premise*. *Completion status* can have either of two values: complete or incomplete. A status of *INCOMPLETE* means that only a subset of the propositions involved in a premise have been proposed by problem solvers. This would be the case in the above scenario right after a problem solver supplies “Jim is working.” *Uninstantiated problem solver data* refers to the data mentioned in the premise which have not yet been supplied by a problem solver. An incompatible which has completion status of *COMPLETE* details a complete set of beliefs which cannot exist in the same context. This set is the *incompatible belief set*. As an example, the following incompatible will become complete when (and if) a problem solver provides the MATMS with an belief whose proposition is “Fred is asleep”. In this case it is important to differentiate between a problem solver datum and the MATMS belief which represents it, so we use the notation $B(x)$ to mean “the belief representing the problem solver datum x ”.

completion status: INCOMPLETE
incompatible belief set: (B(“Fred is awake”))

uninstantiated problem solver data: ("Fred is asleep")
 premise: \neg ("Fred is awake" "Fred is asleep")

When and if the incompatible becomes completed, all contexts will be searched to determine if any one of them includes the incompatible set. The completed frame would look like:

completion status: COMPLETE
 incompatible belief set: (B("Fred is awake") B("Fred is asleep"))
 uninstantiated problem solver data: ()
 premise: \neg ("Fred is awake" "Fred is asleep")

4.5.3.3 Write Operations Operations of the MATMS will be discussed from the perspective of the MATMS. The next two sections detail how the MATMS manipulates its data structures in response to problem solver requests.

There are four basic write operations: a problem solver proposes adding an assumption to its belief set, a problem solver proposes removing an assumption from its belief set, a problem solver proposes an inference, and a problem solver proposes removing a particular justification for an inference. Each is discussed in detail.

4.5.3.3.1 Problem Solver Proposes Adding Assumption The MATMS follows the operations described below and in Figure 20 when a agent proposes adding an assumption to its belief set. Note that implicitly an agent may only request to add a non-default assumption to its belief set. This will be discussed in Chapter 4.

When a problem solver proposes adding an assumption to its belief set, the MATMS first determines if a proposition is already present which matches the proposed assumption. If the proposition already exists, then there must exist either a default assumption, a non-default assumption, or an inference which has the proposition in its *proposition* slot. If it is a default assumption, the agent is not operating properly, because a problem solver cannot "re-accept" a default assumption by explicitly attempting to add it to its belief set. (This is discussed in Chapter 4.) If it is a non-default assumption, the MATMS must check to see if it is already present in the problem solver's current context. If it does, then the problem solver clearly made a mistake and is so notified. If an inference has the proposition in *proposition*, then the problem solver is notified that it is attempting to assume something which has already been derived.

If a proposition did not already exist, one is instantiated at this time. Whenever a new proposition is made, the incomplete incompatibles are examined to determine if the new problem solver datum will complete any of them. Then the premises are searched to determine if any new incompatibles should be started. After this search, the assumption is instantiated with the new proposition as its *proposition*.

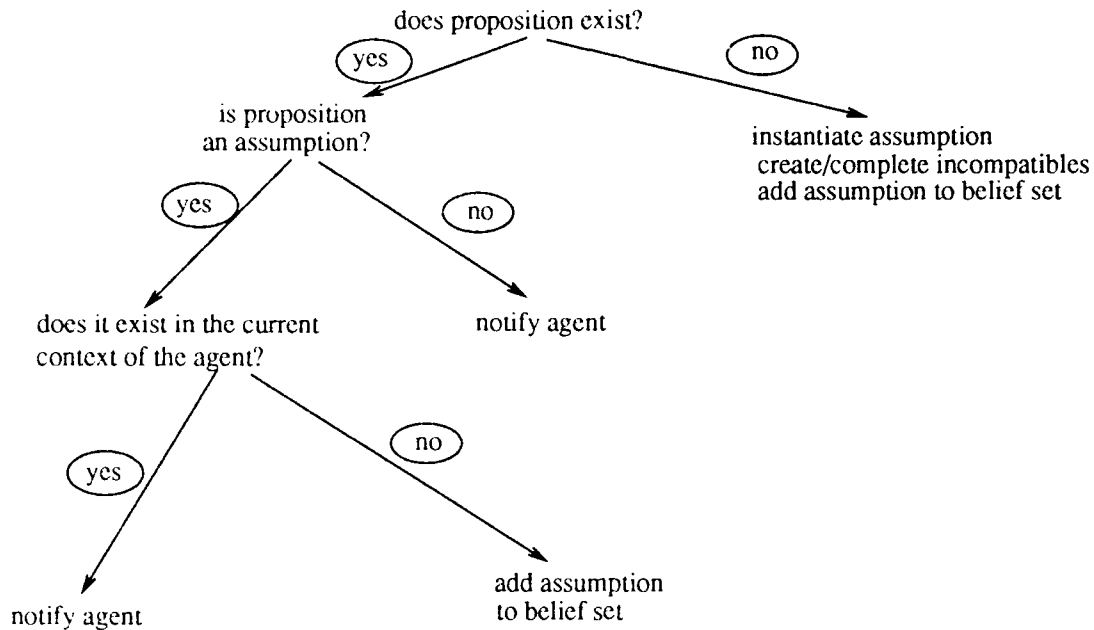


Figure 20: Decision Tree for *Problem Solver Proposes Adding Assumption*

At this point, if there is an assumption with the proposition as its *proposition*, and that assumption is not already in the problem solver's belief set, the MATMS must find or create a context which has an environment containing only the environment of the old problem solver context and the new assumption. Note that this has no effect on the belief sets of the other problem solvers — they remain in their current contexts.

The creation of a context occurs in four phases:

1. The context is first instantiated with most of the slots unfilled — only the *environment* slot is set, with a list including the new assumption and the assumptions of the previous context.
2. Each inference which has been derived from the set of assumptions is now placed in the *inferences* slot of the context.
3. All incompatibles are now examined to see if the new context is inconsistent. If it is, the *incompatibles* slot is set appropriately.
4. The context is appended to the *contexts in* slot of each inference and assumption included.

Whether or not a context had to be created for this different environment, the problem solver now switches contexts. If the context is inconsistent, the problem solver making the

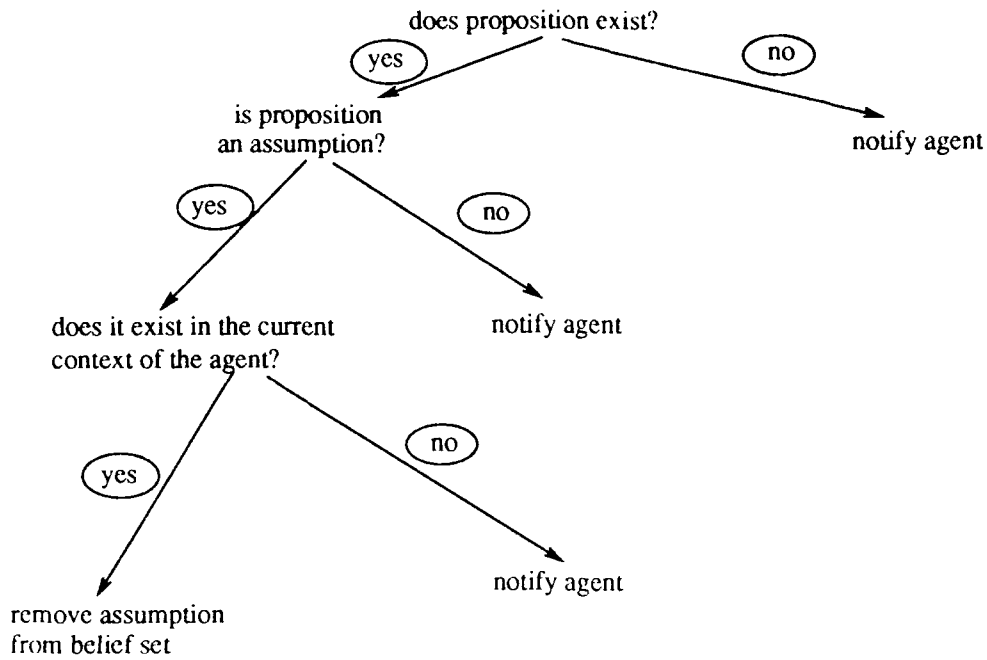


Figure 21: Decision Tree for *Problem Solver Proposes Removing Assumption*

assumption must be notified. A list of incompatible beliefs and how to remove each belief (this can only be done by retracting assumptions) are returned to the problem solver.

4.5.3.3.2 Problem Solver Proposes Removing Assumption Figure 21 illustrates the procedure the MATMS follows when a problem solver asks the MATMS to remove an assumption from its current belief set. Again, recall that a problem solver may only ask to remove a non-default assumption from its belief set.

When a problem solver asks to remove a particular assumption from its belief set, it is asking to be placed in a context which includes all assumptions of its current environment except for the assumption in question. Clearly the assumption must already be present in the MATMS knowledge base, and specifically it must be in the present context of the problem solver. The problem solver is notified accordingly if this is not the case.

To actually remove an assumption from a problem solver belief set, a context is sought whose environment matches the new set of assumptions. If it is not found, then it is created by the procedure described in the previous section.

The problem solver is then placed in the new context. If the context is inconsistent, the agent is notified.

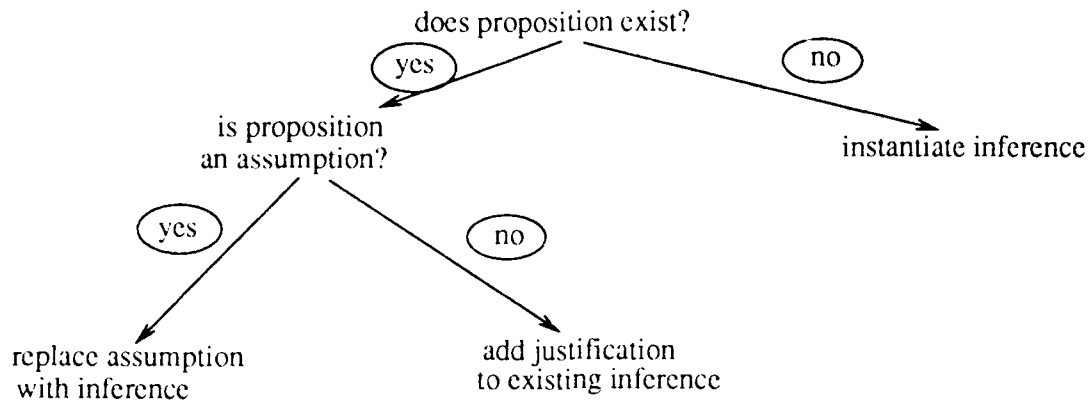


Figure 22: Decision Tree for *Problem Solver Proposes Inference*

4.5.3.3.3 Problem Solver Proposes Inference As with other operations discussed, the proposition which corresponds to the inference supplied by the problem solver is the key for how the MATMS decides on an action to take. If the proposition is not present, clearly the inference must simply be instantiated. If the proposition exists and is attached only to an assumption, the inference must be instantiated. If an inference is present which has the proposition as its *proposition*, the problem solver is proposing what it believes is a valid justification for that inference, whether or not it realizes that the inference already exists. The decision tree implied here is depicted in Figure 22.

To clarify the discussion, registering an inference with the MATMS will be divided into three types of operations: instantiating a new inference, replacing an assumption with an inference, and supplying an existing inference with another assumption.

Common to all three types of inference operations is a trace of the justification supplied. When an agent proposes an inference, it also provides the justification for the inference. The first action the MATMS takes is to trace each belief in the justification until the assumptions upon which the belief is based are found. The minimal combinations of these assumption sets are the assumptions upon which the inference is based. For example, suppose we have the following knowledge base for a two-agent system: "Route 101 is fast" because "there aren't many policemen on Route 101"; "Route 101 is fast" because "Route 101 is a four lane highway"; and "Route 56 is slow" because "there are many potholes on Route 56". An agent then proposes the inference "Route 101 is preferred over Route 56" with justification ("Route 101 is fast" "Route 56 is slow"). The minimal subsets upon which the inference would be based are: (("there aren't many policemen on Route 101" "there are many potholes on Route 56") ("there are many potholes on Route 56" "Route 101 is a four lane highway")).

Adding a new inference to the MATMS knowledge base is the most straightforward of the three types of operations. The procedure for adding an inference is to instantiate the inference with the proposition as its *proposition*, the minimal assumption sets determined

above as its *assumptions based upon*, and the justification itself as its sole *immediate preconditions*. After the inference is instantiated, the inference is used in an attempt to complete the existing incompatibles. Also, new incompatibles are created from relevant premises.

The more complicated steps in adding a new inference pertain to contexts. The *assumptions based upon* slot determines to which contexts the inference should be added. For each assumption set in *assumptions based upon*, the intersection of each assumption's *contexts in* slot is taken. This list represents the list of contexts to which all assumptions in the set belong. The inference is added to the *inferences* slot of each context in this list, and the context is added to the inference's *contexts in* slot.

Note that when a problem solver adds an assumption to its belief set, no more than one context can be found inconsistent as a direct result of adding the assumption. Only the problem solver which added the assumption might be placed into an inconsistent context. However, when a problem solver registers an inference with the MATMS, many contexts might be found inconsistent. This implies that other problem solvers might suddenly be working with inconsistent belief sets as a result of one problem solver registering an inference. Each problem solver whose belief set becomes inconsistent must be notified and corrective alternatives must be supplied.

An inference can replace an assumption when a problem solver has derived something which either it or another problem solver had previously assumed. (This assumption could be either default or non-default.) This is a likely occurrence as a result of "normal" problem solving activity -- as a problem solver proceeds, it may reach a point at which it does not know the present (or any reasonable) value for a particular piece of knowledge necessary to continue working, so it "guesses" a value. Later either it or another problem solver may produce or recognize confirming evidence, which in effect replaces the assumption with an inference. Conceptually, replacing an assumption with an inference involves replacing all occurrences of the assumption in the knowledge base with the inference.

The first step in replacing an assumption in the MATMS knowledge base with an inference is to instantiate the inference using the procedure described earlier in this section. Instantiation is independent of the fact that the inference will be used to replace an assumption.

No new incompatible will be created nor will any be completed when an inference replaces an assumption, because incompatibles are ultimately dependent upon propositions, not beliefs. An assumption with this proposition as its *proposition* has already been created, so the incompatibles which are relevant to the problem solver datum in question have already been created or completed. They must be altered, though, because they refer to the assumption rather than the inference. All incompatibles which mention the assumption therefore have the assumption replaced by the inference.

Next, the *influences* slot of the inference must be adjusted to reflect the *influences*

slot of the assumption. Each inference which the assumption influenced must have its *immediate preconditions* and *assumptions based upon* recalculated. In general this could cause a fair amount of updating if the assumption influenced many inferences.

If the assumption was *non-default*, the contexts which contained the assumption must be killed, because they will never be referenced again. Killing a context means removing the context from *contexts in* of each belief in the context and deleting the instantiation of the context. If a problem solver is currently working in a context which is about to be killed, then it should be notified properly -- the problem solver will be told that one of its assumptions has been replaced by an inference. Generally, the problem solver will simply choose to accept the assumptions on which the inference is based. This way, the problem solver's belief set will continue to contain at least the same beliefs as its original belief set.

When an inference replaces an assumption, the MATMS records the details of the transaction so that it knows what to restore in case the inference is retracted.

An example illustrates how the MATMS operates to replace an assumption with an inference. Consider a simple two-agent system consisting of Agent₁ and Agent₂. The following transaction occurs:

1. Initially, each agent is working only with the default assumptions. This "null" context will be referred to as *C0*.
2. Agent₁ adds the assumption "there are many potholes on Route 56" (*A1*) to its belief set. This causes the MATMS to create a new context, *C1*. Agent₁ is then placed in *C1*.
3. Agent₁ adds the inference "Route 56 is slow" (*I1*) to its belief set. The inference is based upon only "there are many potholes on Route 56" (*A1*). The MATMS adds the inference to every context which contains *A1* -- only *C1*.
4. Agent₁ adds the assumption "Route 101 is fast" (*A2*) to its belief set. This causes the MATMS to create a new context, *C2*. Agent₁ is then placed in *C2*.
5. Agent₁ adds the inference "Route 101 is preferred over Route 56" (*I2*) to its belief set. The inference is based upon "Route 101 is fast" (*A2*) and "Route 56 is slow" (*I1*). The MATMS adds the inference to every context which contains *A1* and *A2* -- only *C2*.
6. Agent₂ adds the assumption "Route 101 is a four lane highway" (*A3*) to its belief set. This causes the MATMS to create a new context, *C3*. Agent₂ is then placed in *C3*.
7. Agent₂ adds the inference "Route 101 is fast" (*I3*) to its belief set. The inference is based upon only "Route 101 is a four lane highway" (*A3*). The inference is added to

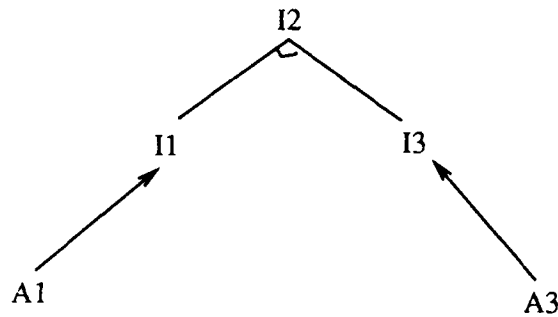


Figure 23: Inference Tree for *Inference Replaces Assumption* Example

	C0	C3	C4
environment	()	(A3)	(A1 A3)
inferences	()	(I3)	(I1 I2 I3)
incompatibles	()	()	()

	A1	A3	I1	I2	I3
contexts in	(C4)	(C3 C4)	(C4)	(C4)	(C3 C4)
influences	(I1)	(I3)	(I2)	()	(I2)
assumptions	*	*	((A1))	((A1 A3))	((A3))
preconditions	*	*	((A1))	((I1 I3))	((A3))

Table 8: Data Structures for *Inference Replaces Assumption* Example

each context which contains $A3$ — only $C3$. Because the inference $I3$ replaces the assumption $A1$, all contexts which include $A1$ must be killed — $C1$ and $C2$. Agent₁ is told that an assumption in its belief set, “Route 101 is fast”, is being replaced by an inference. In order to continue with its activity, it should accept the assumption “Route 101 is a four lane highway”.

8. Agent₁ adds the assumption “Route 101 is a four lane highway” ($A3$) to its belief set. This causes the MATMS to create a new context, $C4$. Agent₂ is then placed in $C4$.

Figure 23 shows the inference tree for this knowledge base. Table 8 represents the data structures of the MATMS at this point in the problem solving.

The algorithm for adding a justification is not very different than the one for replacing an assumption with an inference. To add a justification to an existing inference, the *assumptions based upon* of beliefs “above” the inference in the tree must be recalculated.

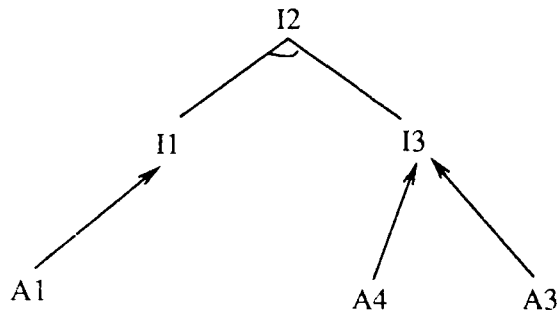


Figure 24: Inference Tree for Example Illustrating Multiple Derivations

In addition, beliefs below the inference could require updating in certain cases. Consider an additional step in the two agent system scenario presented a few paragraphs above.

9. Agent₂ adds the assumption “There aren’t many policemen on Route 101” (*A₄*) to its belief set. This causes the MATMS to create a new context, *C₅*. Agent₂ is then placed in *C₅*.
10. Agent₂ adds another justification for the inference “Route 101 is fast” (*I₃*). The justification is only “There aren’t many policemen on Route 101” (*A₄*). The inference is not added to any contexts because the only context which includes *A₄* already includes the inference.

Figure 24 shows the inference tree for this knowledge base. The relevant portion of the data structures of the MATMS at this point in the problem solving are presented in Table 9.

In general, adding a justification to an existing inference could be far more expensive than simply replacing an assumption with an inference because search must occur in both directions, instead of just up the tree.

4.5.3.3.4 Problem Solver Proposes Retracting Justification of Inference

When an agent proposes retracting a justification for an inference, it is asking to remove a certain list of beliefs from the inference’s *immediate preconditions*. If the justification exists, the MATMS must perform a potentially long series of operations. The easiest steps are the earliest. First, the justification is removed from the inference’s *immediate preconditions*. Next, the *influences* slot of each belief mentioned in the justification the problem solver wishes to remove is readjusted. More precisely, the inference is removed from *influences* of each belief in the justification which is no longer mentioned in any member of *immediate preconditions* of the inference. From there, the steps become more costly. The decision tree for this case is shown in Figure 25.

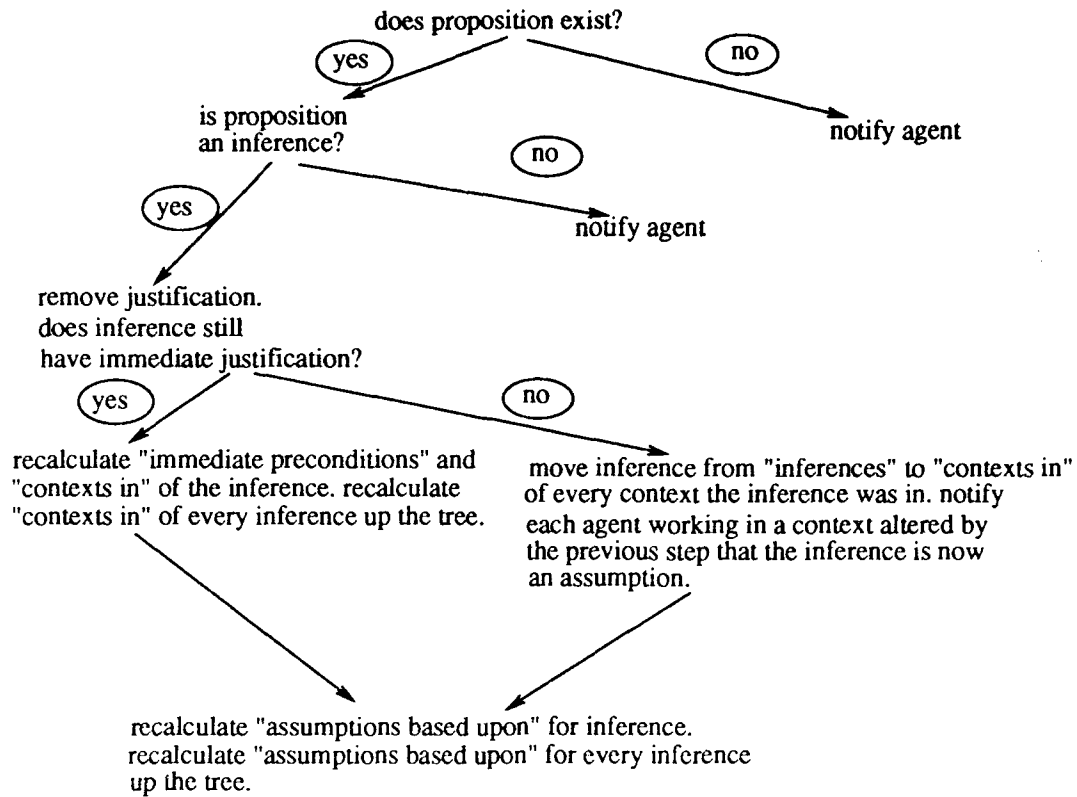


Figure 25: Decision Tree for *Problem Solver Proposes Retracting Justification*

If there is still at least one set of beliefs in the inference's *immediate preconditions*,

	C3	C4	C5			
environment	(A3)	(A1 A3)	(A3 A4)			
inferences	(I3)	(I1 I2 I3)	(I3)			
incompatibles	()	()	()			

	A3	A4	I1	I2	I3
contexts in	(C3 C4 C5)	(C5)	(C4)	(C4)	(C3 C4 C5)
influences	(I3)	(I3)	(I2)	()	(I2)
assumptions	*	*	((A1))	((A1 A3)(A1 A4))	((A3)(A4))
preconditions	*	*	((A1))	((I1 I3))	((A3)(A4))

Table 9: Data Structures for Example Illustrating Multiple Derivations

then the *contexts in* of every inference up the tree (including the inference itself) must be recalculated, because it may no longer belong to a context currently in *contexts in*. An inference is removed from a context by removing the context from the inference's *contexts in* and removing the inference from the context's *inferences*.

If there is not a set of beliefs in the inference's *immediate preconditions*, then, conceptually, the procedure detailed earlier concerning replacing an assumption with an inference must be reversed. The only deviation from simple inversion of that algorithm is that for every context to which the inference previously belonged, the inference is replaced by the assumption. Also, if a problem solver is currently working in one of these contexts, it must be notified that an inference it was working with is now an assumption.

Next, every inference "up" the tree (including the inference from which the justification has been removed) must have its *assumptions based upon* recalculated, as well as a possible adjustment to *contexts in*.

4.5.3.4 Read Operations The discussion of the MATMS would not be complete unless its knowledge access functions were discussed. The read operations, from the perspective of the MATMS, are simple. The problem solver has the much more difficult task of deciding what to ask, and how to ask it.

The most important (and usually the most difficult) feature of accessing MATMS data is the domain dependent mapping from the problem solver data implied in the query to the relevant set of beliefs. This mapping results in both knowledge which has been explicitly stated by a problem solver in the course of normal problem solving and the default knowledge. In Chapter 5, where an example implementation is outlined, a domain-specific mapping function which relates to a frame based knowledge base implementation will be discussed. Once the set of relevant beliefs has been determined, the rest is straightforward.

Read operations can be viewed as falling into one of three categories: is a particular problem solver datum contained in a particular problem solver's belief set? (problem solver dependent query); what problem solvers currently believe a particular problem solver datum? (all problem solver query); and describe all beliefs relevant to a particular problem solver datum (context independent query).

For a *problem solver dependent query*, after determining which beliefs are relevant to the problem solver data in question, the context of the problem solver to which the query refers is consulted. First, the non-default beliefs are considered. If any of these are contained in the context, the MATMS replies appropriately. The MATMS could respond with more than one belief, and the beliefs could be contradictory. If none of these beliefs are contained in the context, then the problem solver is considered "opinionless", and only the default knowledge is returned if it exists. If default knowledge is returned, it is identified as such in the response.

For the *all problem solver query*, a problem solver dependent query is performed for

all problem solvers.

An *all context query* is used when an agent requires all relevant beliefs concerning a particular problem solver datum. All beliefs, including default beliefs which are relevant, are returned. If the belief is an inference, then its derivation is returned. Assumptions are simply returned, identified as assumptions.

The problem with the responses to an *all context query* is that a problem solver may not understand many of the intermediate steps used to derive the inference. It may not even understand the assumptions the inference is based upon. Perhaps a more useful query is a variation of this — the problem solver simply gives the MATMS a set of beliefs, and then asks what a particular problem solver datum would be if the beliefs were “true”. In other words, the problem solver might ask something of the form “Suppose *A* and *B* were true. What would be the value of *C*?” If *C* can be derived from *A* and *B* either directly or indirectly, the MATMS responds accordingly. If *C* is an assumption in the knowledge base, then the MATMS would respond that it is an assumption and its validity is thus not connected to *A* or *B*. If no logical connection between *A*, *B*, and *C* has been registered with the MATMS, then it would reply only the default value for *C* if it exists.

4.5.4 The MATMS Interface

The MATMS was designed to be used by agents that “understand” a specific set of operating constraints. For this reason, unless it is used properly, some features of the MATMS might be lost. This section discusses the MATMS interface and shows how the MATMS should be used by agents without constraining the design of problem solvers. Problem solving can take place in a variety of forms, so a presentation of how exactly it should be done is impossible. Rather, this section details some basic aspects of problem solving and in particular what a problem solver should expect the MATMS to do and reply when the problem solver interacts with it. Whereas the previous section was written from the viewpoint of the MATMS, this section is from the viewpoint of a problem solver.

4.5.4.1 Designing an Appropriate Problem Solver A problem solver which would work well in the MATMS setting must be rational. In terms of read and write operations, it also must be aware of what to expect from the MATMS, and how to use the MATMS.

4.5.4.1.1 Rationality The problem solver can only expect rational results from the MATMS if its inference mechanisms are rational. That is, supplying the MATMS with inferences which contradict each other logically on the basis of the inference’s *immediate preconditions* will cause the MATMS to act irrationally. This should be expected, since the MATMS is only reflecting what it is supplied with.

The general rule is that a problem solver cannot produce inconsistent inferences from

the same set of assumptions. The concept of rationality can be illustrated by a number of examples.

1. $AB \Rightarrow C$
 $AB \Rightarrow D$
 $CD \Rightarrow \perp$

A problem solver possessing these rules would be clearly irrational. Two rules acting on the same set of preconditions cannot result in contradictory expressions.

2. $AB \Rightarrow E$
 $CD \Rightarrow F$
 $EF \Rightarrow \perp$

At the other end of the spectrum, a problem solver with these rules is rational. Two rules can act on completely different sets of beliefs and result in differing expressions. Comparisons of most problem solving rules fall into this category.

3. $AB \Rightarrow C$
 $AD \Rightarrow E$
 $CE \Rightarrow \perp$

This example falls in between the extremes characterized by the first two examples. These inference rules are rational when compared to each other.

4. $AB \Rightarrow C$
 $ABE \Rightarrow D$
 $CD \Rightarrow \perp$

While these rules also fall in between the extremes given by the first two examples, this set is irrational according to the manner in which the MATMS operates. The first rule states that in the presence of A and B , the MATMS should include C . The second rule states that in the presence of A , B , and E , include D . With these rules, the context of the environment ABD will include C and D , which is irrational.

Rationality as discussed here is conceptually not difficult to encode in a problem solver, for it requires only that the inference rules of a problem solver be rational as compared to each other. This property will be referred to as *self-rationality*.

4.5.4.1.2 Assumptions During normal problem solving activities, an agent can be expected to make or use assumptions. A problem solver makes assumptions when it is unsure of a particular piece of knowledge. There are three points which need to be made concerning assumptions.

First, consider the example situation. An agent has a particular rule:

$$ABCD \Rightarrow E$$

If the agent currently believes A , B , and C , then it could assume that E is valid, also. Even though an inference rule indirectly has produced E , E should not be registered as an inference. An inference is meant to represent that all preconditions of a rule have been met, which is not the case in this example.

This leads to the first point regarding assumptions. Problem solvers should internally record why a particular assumption was made. It is important to realize that the MATMS should be used to record the assumption itself, not the reasons why the assumption was made. A problem solver should simply record E with the MATMS, and internally maintain the knowledge that E was assumed because it has a rule ($ABCD \Rightarrow E$) which had most of the preconditions necessary for its firing believed.

Second, an agent should never attempt to add a default assumption to its belief set, because it is most likely already present. If it is not, because the agent has overridden the default, then the appropriate way of reasserting the default is by retracting the belief which overrides it.

Third, if the problem solver is present with contradictory default assumptions, the manner in which it should register with the MATMS that it believes one default in particular is by accepting beliefs which directly override the defaults which the agent does not accept. This is a little awkward, but overall the best procedure. In general, inconsistent default assumptions should be avoided.

4.5.4.1.3 Interacting with the MATMS When a problem solver changes its assumption set, it should always register the change with the MATMS so that the MATMS can either remove or add inferences as necessary and determine if the new belief set is consistent. The MATMS will always confirm the transaction with the problem solver in one way or another. This confirmation might include a message indicating that the problem solver may have made a mistake, such as when the agent attempts to add an assumption to its belief set which is already present. If the resulting belief set is inconsistent, the MATMS will inform the problem solver that certain subsets of the belief set are inconsistent. Each belief in each subset will be described. Each description includes the assumptions from which the belief has been derived. It is up to the problem solver to determine which assumptions it wishes to remove in order to make its belief set consistent.

When an agent registers an inference with the MATMS, the inference rule used to generate the inference should not be included (this was proposed in [8] as a method to record the control sequences used to generate knowledge.) This would be counter-productive to the overall system, because an inference of one agent should not be inherited by another unless it explicitly included the other's inference rules in its belief set.

Determining which read operation to perform in a given situation is difficult. When an agent queries its own beliefs of itself as well as other problem solvers, the *problem*

solver dependent query should be used. If a problem solver must assess the overall belief state of a particular piece of knowledge, the *all problem solver query* should be considered. To determine what any problem solver has ever believed concerning a particular piece of knowledge, the *all context query* should be used.

4.5.4.2 Designing a System around One MATMS Designing a problem solving system around one MATMS should be influenced by three topics: mutual rationality, inconsistency across problem solvers, and the utilization of system-wide default knowledge.

4.5.4.2.1 Mutual Rationality It has already been discussed that a problem solver must be self-rational. In addition, a problem solver must be rational as compared to the others for essentially the same reasons as were mentioned in the case of a single problem solver. This is true because inferences are not problem solver dependent. We say that two problem solvers are *mutually rational* if the inference rules of each problem solver could be used to create a self-rational problem solver.

When considering a single problem solver, requiring rationality is not unreasonable. If any problem solver in any system were not rational, it would probably not be very productive. However, the criteria that problem solvers be rational when compared to one another is a somewhat more restrictive and difficult to achieve. Two problem solvers could for instance be self-rational, but be irrational when compared to each other. To ensure mutual rationality, problem solvers must be designed in accordance with overall system goals; coordination of design is essential.

4.5.4.2.2 Resolving Inconsistency among Problem Solvers Mutual rationality suggests only that the inference rules of one problem solver be consistent with all other problem solving rules. For instance, given the same preconditions, two inference rules should not produce two pieces of knowledge which are contradictory. Mutual rationality includes nothing about what assumptions each problem solver can choose with which to work. This allows each problem solver to perform in a variety of problem solving activities, fairly independent of the activity of the others.

Two problem solvers which are mutually rational could seemingly present contradictory beliefs to the MATMS. For instance, suppose one problem solver had presented "it's 90° out" as an assumption in its current belief set, and had then inferred "it's a good day to go swimming because it's 90° out". Suppose that another problem solver had told the MATMS that its current context belief set includes "it's 40° out", and had then inferred "it's a bad day to go swimming out because it's 40° out". Clearly each problem solver is self rational, as well as mutually rational when compared to the other. In addition, the MATMS would "support" the context of each problem solver, because the belief set of each problem solver is self-consistent.

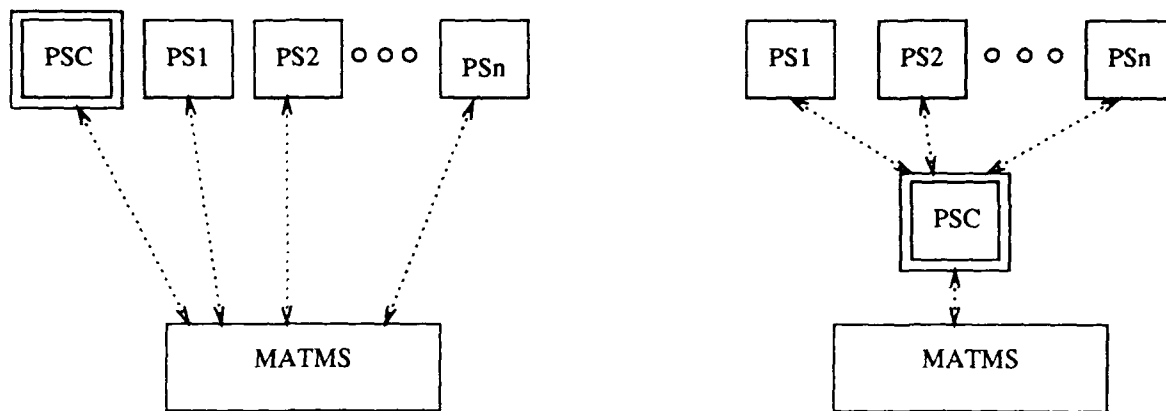


Figure 26: Proposed Interagent Communications Paths

However, there is clearly a problem with the overall problem solving. The individual problem solving is diverging if indeed the first problem solver believes that it is truly 90° outside, and the second believes that it is 40° (neither agent is involved in hypothetical reasoning). Certainly the MATMS recognizes that there are two assumptions in the knowledge base — 90° out and 40° out — which can not exist in the same context because they are directly contradictory. The conflicting beliefs are not present in the same context, so there is no problem from the viewpoint of the MATMS.

Any time the MATMS is used, a single problem solver should monitor the true world assessments of the others and recognize when inconsistencies between problem solvers arise. This problem solver is essentially part of the domain independent MATMS, except when considering that the rules necessary to resolve the conflicts must be domain dependent.

Two architectures could be investigated, depending upon the domain. These architectures are shown in Figure 26, with the agent responsible for resolving inconsistency labeled as PSC. The architectures differ primarily in the interagent communication paths utilized. In the first, each agent interacts directly with the MATMS to get its beliefs. This would be faster for the individual problem solvers, but would also make the job of the agent resolving the inconsistencies difficult. In the second, each problem solver interacts through the agent resolving the inconsistencies to communicate with the MATMS. Monitoring beliefs is thus much easier.

The agent responsible for maintaining consistency across local problem solvers can certainly recognize inconsistencies between problem solvers. To resolve them, either of two methods could be used. The PSC could itself choose one value over the other without

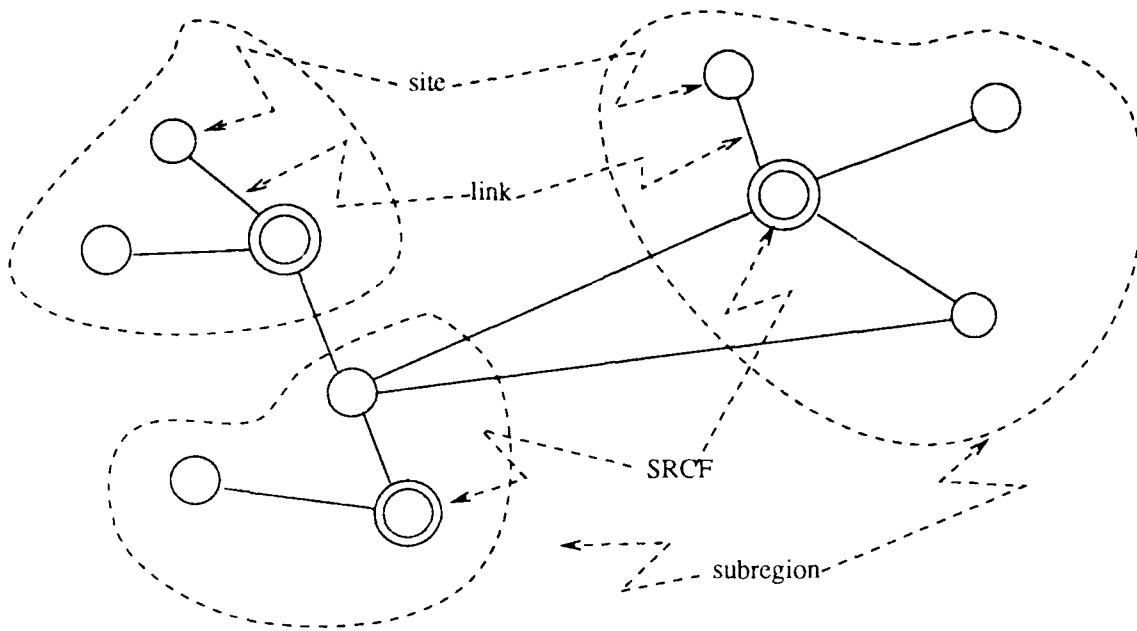


Figure 27: Sample Communications Network

consulting the two problems solvers from which the beliefs originated. Alternatively, the PSC could tell the problem solvers that they conflict with one another, leaving resolution up to the inconsistent problem solvers. Both methods require further research.

4.5.5 Implementation for Communications Network Management

4.5.5.1 The Domain The domain for which the MATMS has been implemented is a distributed knowledge based system for managing a large-scale communications network. The communications network provides telecommunications service for people as well as machines. The communications system can be described on three levels.

On one level, the communications network can be viewed as a sparsely interconnected array of transmission facilities called *sites*. Each site is generally only connected to one or two other sites. The interconnections between sites (*links*) provide a transmission medium over which to send communications signals between sites. For control purposes, sites are grouped into non-overlapping sets called *subregions*. Each subregion contains one SubRegion Control Facility (*SRCF*) to which each site in the subregion reports the operational status of its equipment, availability of resources, etc. A portion of a "typical" communications network is presented in Figure 27.

Another level is the equipment configuration at each site. This level includes the equipment and connections necessary to originate and switch communications signals.

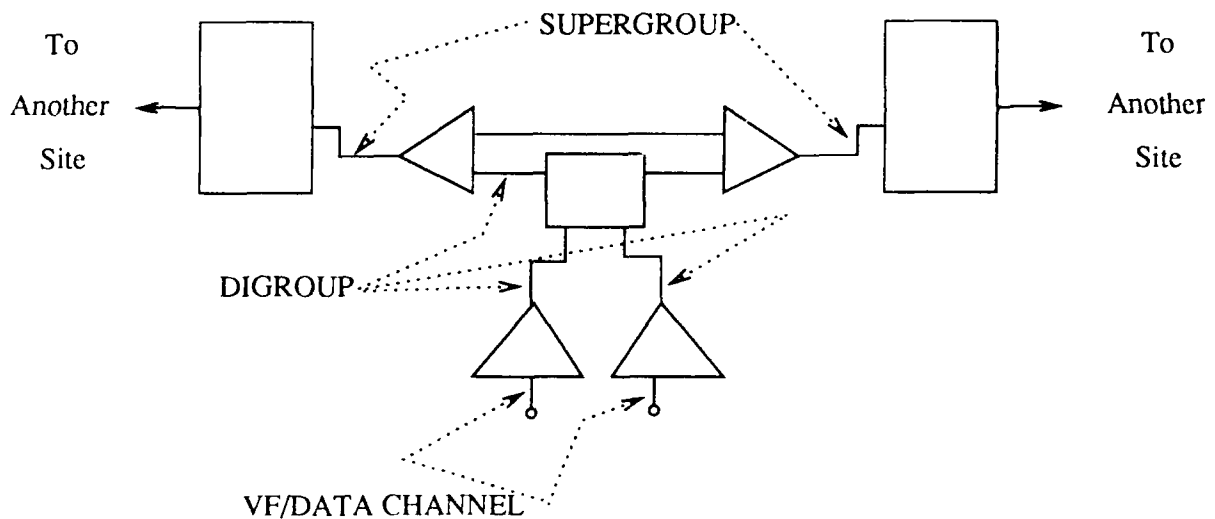
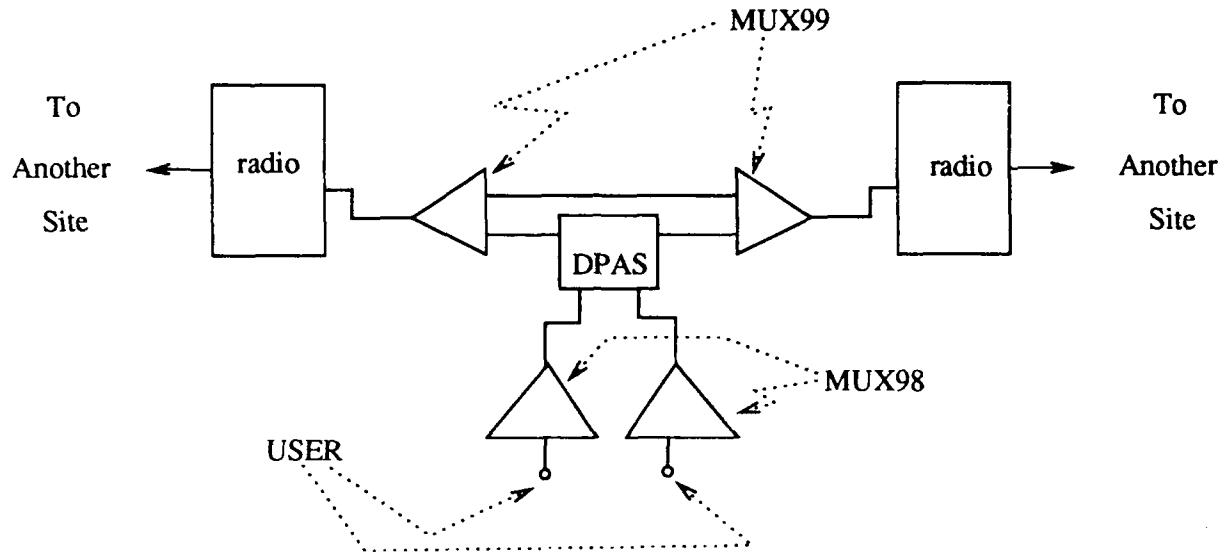


Figure 28: Sample Equipment Configuration

An example equipment configuration is given in Figure 28 (adapted from [4]). Equipment and their interconnections are constrained by a variety of rules.

The third level – the communications path level – is probably the most important. It can be considered the fundamental view of the network. The two primary objects

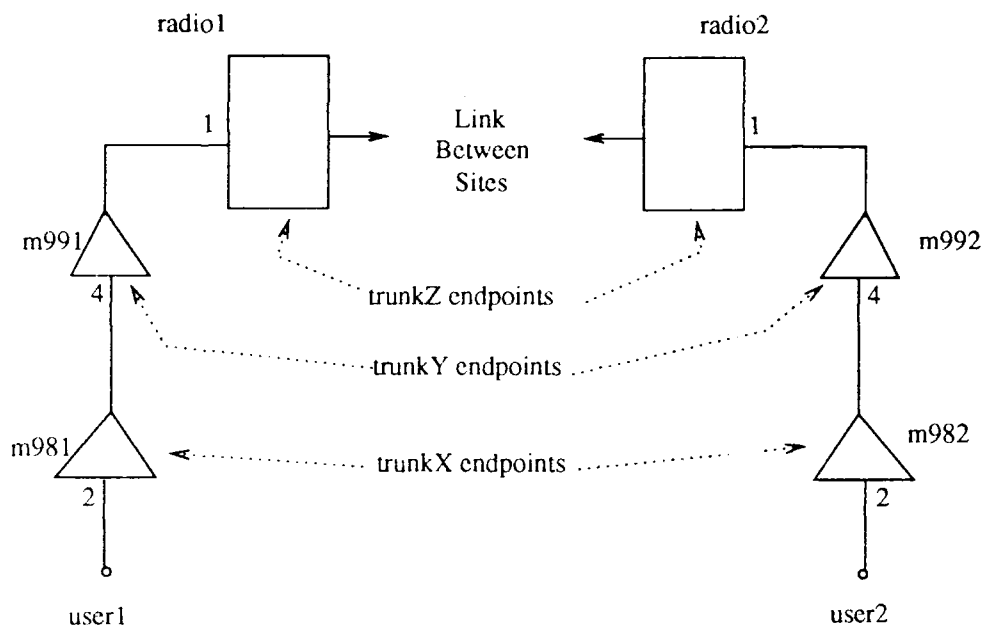


Figure 29: Sample Trunk and Circuit Configuration

present at the communications path level are *trunks* and *circuits*. A circuit is the complete elementary path between two pieces of terminal equipment by which two-way telecommunications service is provided. A trunk is a group of equipment and connections which establishes telecommunications connectivity by providing a resource for circuits to ride. Circuits ride on channels of a trunk; there is typically a capacity for several channels per trunk. A useful analogy for channels on a trunk is to imagine one big pipe (the trunk), which contains a number of small pipes (channels) running the entire length of the big pipe. With this analogy, it is easy to see that a trunk can ride channels of other trunks. The trunk exists with or without the circuit, but this is not symmetric; a circuit cannot exist unless it rides a trunk, or a list of trunks connected in series. The trunk refers to “physical” connectivity, whereas the circuit refers to “logical” connectivity. For more details, see [31].

In order to understand how the MATMS operates in this domain, one should understand the general concepts of the communications path level. Thus, an example of two sites partially configured is presented in Figure 29 with careful attention to trunks and circuits. The description of Figure 29:

- *ckt1* (connecting *user1* and *user2*) rides *trkx* which rides *trky* which rides *trkz*.
- *trkx* starts at *m981* and ends at *m982*. It has 24 channels.
- *trky* starts at *m991* and ends at *m992*. It has 8 channels.

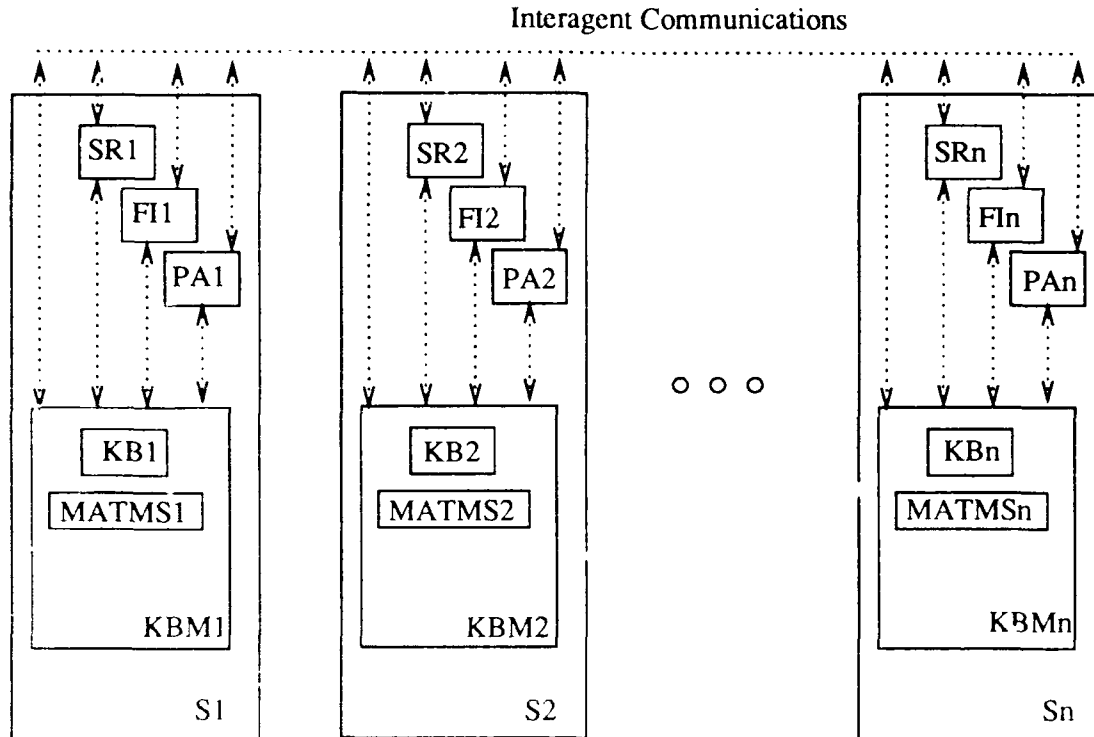


Figure 30: Communications Network Knowledge-Based System Architecture

- *trkz* starts at *rad1* and ends at *rad2*. It has 2 channels.

If any piece of equipment should fail or degrade to the point of causing any of *tkx*, *tky*, or *trkz* to fail, then the two end users of *ch1* would be disconnected. The circuit would be said to be *disrupted*.

Maintaining communications service is the primary objective of the system. If any circuit fails, then the circuit must be restored often at the same time as the problem is being diagnosed. Restoration could proceed by reconfiguring equipment, or selecting existing alternative trunks, or a combination of both.

4.5.5.2 Knowledge-Based System Architecture The knowledge base system architecture was briefly discussed in the introduction. The purpose of this section is to expand on that description. In particular, the role of the KBM will be discussed further.

The architecture of the knowledge-based system is shown in Figure 30. The interagent communications paths have been included in the figure to convey the general operation of the system. Although agents can communicate with other agents of the same type at different subregions, most communication is with the local Knowledge Base Manager (KBM).

The KBM has many responsibilities. It grants access to the knowledge in the local knowledge base, must process the transactions from the SR, PA, and FI agents in a logical order, must know where knowledge requested by the agents resides (if it not present within the local knowledge base), and most importantly it has to maintain a consistent local view of the state of the world by monitoring the beliefs of the individual agents in the node in combination with other KBMs in the network.

To aid in maintaining a consistent local view, the KBM includes the MATMS. The MATMS is used to keep the belief sets of each of the agents consistent, as well as to recognize inconsistency when comparing belief sets. It is the KBM, however, which must attempt to recognize and resolve the inconsistency. The role of the KBM after recognizing discrepancies is to advise the problem solvers as to how to resolve the inconsistencies, often by consulting other KBMs where appropriate. For example, if PA believes a certain trunk is not operable, and FI believes it is, the KBM might ask another KBM what it believes the state of the trunk to be. This of course would only work if the trunk crossed subregion boundaries (so that another KBM *would* have knowledge of it), and the KBM knew which other KBM to ask.

4.5.5.3 Architecture Implementation The global knowledge base is created using the Graphical User Interface for Structural Knowledge (GUS[18]) on a Symbolics 3670. An option in GUS divides the knowledge base along subregion boundaries in order to create the knowledge bases for each problem solving system discussed in the previous sections. Division of knowledge can be modified to test different distributed knowledge representation schemes. The knowledge representation scheme in GUS is frame-based, and this representation is also utilized in the distributed knowledge bases.

A distributed simulation environment (SIMULACT[24]) is used to test the knowledge based system. It provides a parallel simulation environment for any number of agents. Interagent communications support is provided in a highly flexible format.

The Knowledge Base Manager (KBM) comprises a single agent in SIMULACT. At this time, it contains query processing functions, the knowledge base itself, and the MATMS. Knowledge provided to the KBM from GUS is treated partially as constants and partially as default assumptions to the MATMS. For example, a particular radio's name is constant, as is its operational definition, while its initial status is treated as a default assumption. In the absence of information to the contrary, the status of a radio is assumed to be the value provided by GUS.

The MATMS exploits the knowledge representation scheme (frames) and to some extent the domain itself. In particular, the task of finding all the beliefs which are relevant to a particular problem solver query is handled by realizing that most queries will access a particular slot in a frame. Therefore, beliefs which are relevant to a given slot are kept within the slot along with the default. This will be illustrated in the section which follows.

4.5.5.4 Examples of MATMS Usefulness

4.5.5.4.1 Example 1 The purpose of this example is to illustrate the way data structures are accessed.

Suppose that there is a frame for a particular instance of a trunk:

id:	trk9
is-a:	trunk
type:	digroup
.	.
.	.
.	.
status:	» MATMS default frame«

The default frame is:

default:	UP
beliefs:	()

The default frame can be interpreted as "Trk9 is up by default. There are no beliefs presented by problem solvers to override the default."

Now suppose that PA asks the KBM for its (PA's) belief concerning the status of trk9. The KBM would invoke the MATMS by attempting to access a slot of a frame which is controlled by the MATMS. In other words, the KBM would attempt to access the slot, which automatically invokes the MATMS. Because there are no beliefs in the context which PA is currently working in (this must be the case because the *beliefs* slot of the default frame is empty), the response would be "UP, by default."

If PA subsequently proposed the assumption "the status of trk9 is down", the MATMS would change the status frame to be:

default:	UP
beliefs:	(A1)

where *A1* corresponds to "the status of trk9 is down."

If PA asked again what the status of trk9 is, the KBM would reply "down" (the KBM would invoke the MATMS to determine the status of trk9, which would find *A1* present in the environment of the current context of PA). However, if FI asked, it would still get "UP, by default." Note that if for some reason FI proposed that the trunk is down, the status slot of trk9 would remain the same. FI would just have *A1* added to its current belief set. Incidentally, if these were the only transactions which the MATMS had made, then the end result is that FI would be placed in the same context as PA.

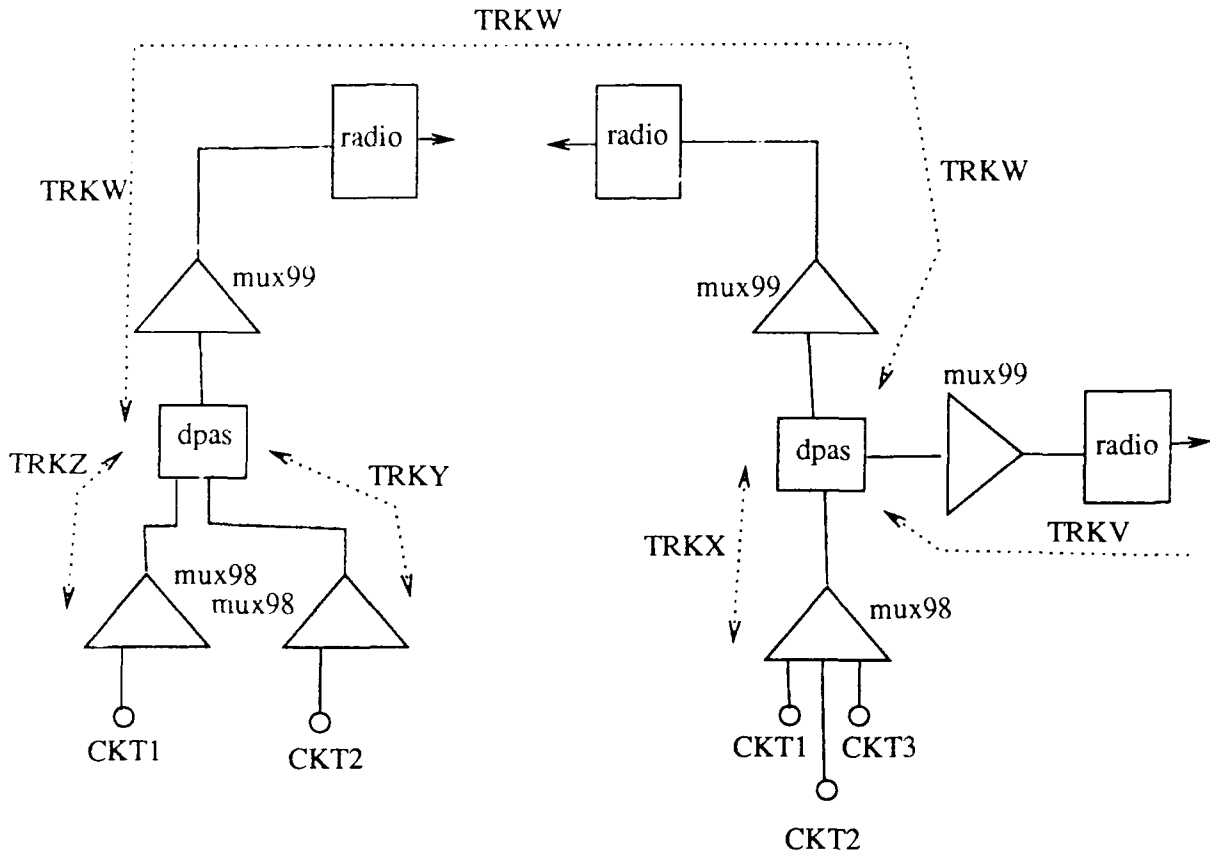


Figure 31: Communications Network for Example 2

4.5.5.4.2 **Example 2** Example 2 shows more of the potential of the system and is tied more closely to the domain.

Suppose that the following configuration exists for a subregion (Figure 31). A general description is that *ckt1* rides *trkz*, *trkw*, and *trkx*. *Ckt2* rides *trky*, *trkw*, and *trkx*. *Ckt3* rides *trkx*, and then *trkv* to another subregion.

Initially, FI, SR, and PA have not made any assumptions — suppositions — about the communications network. Each problem solver is working in C1 (context 1). (Each time a new context is created by the MATMS, the context counter is increased by one. For instance, the next context created will be named C2, and so on.) The knowledge base consists of the defaults (though only the defaults relevant to the discussion are mentioned here):

trkw is up

[A1]

<i>trkv</i> is up	[A2]
<i>trkz</i> is up	[A3]
<i>trky</i> is up	[A4]
<i>trkx</i> is up	[A5]
<i>ckt1</i> is up	[A6]
<i>ckt2</i> is up	[A7]
<i>ckt3</i> is up	[A8]

In addition, the following premises have been entered into the MATMS: a trunk cannot be up and down (P1), a circuit cannot be up and down (P2).

Now, at time t_0 , suppose PA is notified of a user alarm concerning *ckt1*, at the same time that it is notified of a user alarm concerning *ckt2*. (A *user alarm* is when the user of a circuit notifies a technical control facility to complain that he has lost service in a particular circuit.) PA views user alarms as assumptions in the form of new observations of the world, and immediately registers the assumptions with the MATMS.

user alarm <i>ckt1</i> at $t=0$	[A9]
user alarm <i>ckt2</i> at $t=0$	[A10]

Adding the two assumptions [A9] and [A10] results in contexts C2 and C3, respectively.

As PA continues to work, it eventually registers the following inferences with the MATMS, through the KBM. Even though there was not a user alarm, PA concluded that *ckt3* was down through the application of the rule "If multiple circuits on a trunk fail at the same time, assume the trunk has failed." Therefore, PA asserts that *trkw* and *trkx* are down. Asserting that *trkx* is down leads PA to infer that *ckt3* is also down.

<i>ckt1</i> is down because	user alarm <i>ckt1</i> at $t=0$	[I1]
<i>ckt2</i> is down because	user alarm <i>ckt2</i> at $t=0$	[I2]
<i>ckt1</i> , <i>ckt2</i> fail at same time because	user alarm <i>ckt1</i> at $t=0$	[I3]
	user alarm <i>ckt2</i> at $t=0$	
<i>trkx</i> is down		[A11]
<i>ckt3</i> is down because	<i>trkx</i> is down	[I4]
<i>trkw</i> is down		[A12]

Note that *ckt1 is down* does not represent a contradiction of beliefs for PA in the MATMS, because *ckt1 is up* was a default assumption.

The end result of this is that PA is placed in a context which is defined by:

name:	C5
environment:	(A9 A10 A11 A12)

	A9	A10	A11	A12
contexts in	(C2 C3 C4 C5)	(C3 C4 C5)	(C4 C5)	(C5)
influences	(I1 I3)	(I2 I3)	(I4)	()

	I1	I2	I3	I4
contexts in	(C2 C3 C4 C5)	(C3 C4 C5)	(C3 C4 C5)	(C4 C5)
influences	()	()	()	()
assumptions	((A9))	((A10))	((A9 A10))	((A11))
preconditions	((A9))	((A10))	((A9 A10))	((A11))

Table 10: Data Structures for Example 2

inferences: (I1 I2 I3 I4)
incompatible belief sets: ()

The knowledge base is briefly described in Table 10.

Because it has no reason to disbelieve PA at this time, the KBM accepts the assessment of the current state of the world by PA as correct. In other words, it adopts the belief set of PA by asking the MATMS to place it in the same context as PA.

When PA is finished assessing the user alarms on the circuit operation, it tells FI to begin work. FI begins by asking the KBM for its current assessment, which happens to be solely PA's assessment. Therefore, FI is placed in context C5. Thus, it inherits the beliefs which assert that *ckt1* is down, *ckt2* is down, *trunkx* is down, etc.

FI performs measurements on each of the circuits or trunks in question and determines that *ckt3* is actually up, not down. FI enters the following beliefs into the knowledge base:

tests of <i>ckt1</i> at t_1		[A12]
tests of <i>ckt2</i> at t_1		[A13]
tests of <i>trkx</i> at t_1		[A14]
<i>ckt1</i> is down because	tests of <i>ckt1</i> at t_1	[I6]
<i>ckt2</i> is down because	tests of <i>ckt2</i> at t_1	[I7]
<i>trkx</i> is up because	tests of <i>trkx</i> at t_1	[I8]

A more careful, step by step analysis of the steps involved when FI changes its belief set is necessary to understand the operations of the MATMS.

1. FI adds A12 to its belief set, which causes C6 to be created. The KBM accepts A12 into its belief set.
2. FI adds A13 to its belief set, which causes C7 to be created. The KBM accepts A13 into its belief set.
3. FI adds A14 to its belief set, which causes C8 to be created. The KBM accepts A14 into its belief set.
4. I6 adds another justification for "ckt1 is down". It serves as confirming evidence, as does adding I7. Because the KBM has accepted the beliefs upon which both I6 and I7 are based upon, it automatically inherits I6 and I7.
5. When FI attempts to add I8, the MATMS responds that its belief set is inconsistent, because it currently believes that *trkx* is up, and *trkx* is down. The MATMS marks the current context of FI (C8) inconsistent. The MATMS informs FI that it can remove "trkx is down" directly, because it is an assumption, and it can remove "trkx is up" by retracting "tests of *trkx* at t_1 ". For FI, there is no great difficulty in deciding that "ckt3 is down" should be removed from its belief set, because "ckt3 is up" is an inference which it just made.

However, the KBM is faced with maintaining consistency. At this point, FI believes that "trkx is down", and PA believes that "trkx is down". In this situation, the KBM clearly believes FI, because PA is prone to errors due to time constraints. That is, PA is forced to make a fast, rough estimate, while FI must be certain of its work before it enters beliefs into the knowledge base. For that reason, the KBM follows FI and retracts "ckt3 is down". Note that FI still believes that "ckt3 is down" until it is told by the KBM that its beliefs are outdated, or it consults the KBM for a new set of beliefs. Further work by FI would suggest that *trkx* is operating properly.

At this point, FI tells SR that it is finished. SR begins work to restore circuits *ckt1* and *ckt2*.

4.5.6 Areas for Future Research

4.5.6.1 Current Limitations There are limitations to the current design of the MATMS which should be investigated further. Specifically, premises need to be extended, and overall system efficiency should be investigated further.

The current premise structure allows for two pieces (or classes) of problem solver data to be considered inconsistent. For instance, "a trunk cannot be up (operational) and down (inoperable) at the same time." This causes difficulties when one attempts to model a more complex premise structure which involves more than two objects. A one-of-three

situation (choosing one route from three choices) or perhaps a two-of-three situation (having enough money to buy any two of a fishing pole, a softball glove, and a tennis racquet, but not all three) cannot be handled currently. This has not been attempted yet because there is no immediate need for it; the premises necessary for our application demand a simple binary situation.

Although the design clearly addresses the importance of an efficient system, the degree of efficiency is still questionable. It is not obvious whether the inefficiencies present are a result of the implementation or the design. It is clear that the system is often forced to recompute justifications for inferences. This could involve a large amount of time in order to keep the inference trees as compact as possible.

There was always some debate about including default assumptions explicitly in the environment of contexts. If they were included, creating contexts could be performed more quickly than in the present design, but there would be more pointers between objects in the knowledge base. On the other hand, not including default assumptions explicitly in contexts increases the ease of comparing belief sets and in general makes the operations of the MATMS more efficient. If, upon testing the system more, it is determined that the MATMS spends most of its time creating contexts, an argument could be made for a redesign which would put default assumptions explicitly in contexts. This modification would not be difficult to make.

Many of the questions involving the efficiency of MATMS have not been addressed because the problem solvers (FI, PA, SR) are still being developed. Interactions involving the MATMS have only been tested with the SR problem solver, so such concerns as multiple derivations, context switching, and to some extent multiple context maintenance have only been tested using "typical" cases which have been fabricated. As the knowledge based system matures, the MATMS will surely be refined, although the design appears to be sufficiently flexible to handle most changes and was designed to be adaptable.

4.5.6.2 Future Pursuits The four primary issues for future investigations involving the MATMS are: creating a third type of belief, grading assumptions with certainty factors, performing distributed truth maintenance, and studying techniques for resolving inconsistency across problem solvers.

Example 2 of the previous section illustrates the possible need for a third type of belief. In that example, PA had reasoned that a trunk (*trkx*) was down because the majority of circuits on the trunk were also down. In particular, the actual rule PA was basing its reasoning upon was:

If *ckt1* is down
 ckt2 is down
and *ckt3* is down then *trkx* is down

Upon closer analysis of PA's reasoning, it is clear that "*trkx* is down" does not fall cleanly into either belief category.

This belief is not strictly an inference because not all of the left-hand-side of the rule used to produce the belief was believed at the time the rule was utilized. That is, PA did not believe that *ckt3* was down at the time it used that rule to assert that *trkx* was down. In analyzing the example, if "*trkx* is down" were called an inference and the scenario continued, in order to remove the inconsistency which would arise concerning the status of *trkx* by retracting an assumption, FI would eventually be forced to retract either "tests of *trkx* at t_1 " or one of "tests of *ckt1* at t_1 " or "tests of *ckt1* at t_1 ." It could not easily remove the inconsistency by retracting an assumption because it believes each of the three assumptions equally. FI could remove the inconsistency by retracting the justification to an inference, but again it would have a difficult decision in selecting the inference for which it would retract a justification.

The belief "*trkx* is down" is not strictly an assumption because its validity is dependent upon other beliefs. For example, if "*ckt1* is down" were retracted, then "*trkx* is down" should also probably be retracted. The problem in treating "*trkx* is down" as an assumption is that the MATMS will not automatically retract it when an agent removes "*ckt1* is down" from its belief set. The current design insists that the problem solver explicitly retract each belief in this case. Therefore, clearly part of the the MATMS's purpose is defeated when "*ckt1* is down" is treated as an assumption.

Overall, the problem is that there will be beliefs which are difficult to categorize. A third category of beliefs -- perhaps called *reasoned assumptions* -- should be pursued. Reasoned assumptions should be handled like assumptions in some ways, and like inferences in other ways.

Another future pursuit involves the general notion of making assumptions, and is perhaps more relevant in the absence of the proposed third type of belief. When a problem solver's belief set becomes inconsistent, it must usually remove one or more non-default assumptions in order to correct the problem. The decision as to which assumption(s) should be removed is very difficult to make. For example, if the MATMS provides a problem solver with the knowledge that two assumptions in its belief set -- A_1 and A_2 are inconsistent, how does the problem solver decide which assumption to remove?

At this time, assumptions provide a purely black-and-white world -- either they are present within a problem solver's belief set, or they are not. The reasons why a particular non-default assumption is made is kept solely within each problem solver making the assumption. For example, in the last example of the previous section, PA used the rule "If multiple circuits on a trunk fail at the same time, assume that the trunk has failed" to conclude that "*trkx* is down." (As discussed earlier in this section, the best manner in which to treat the belief "*trkx* is down" is to consider it an assumption.) Clearly PA knows why it made the assumption; however, this knowledge is not kept within the MATMS. The MATMS is designed only to keep track of which inferences depend upon the assumption.

not why the assumption was originally made by the problem solver. There is no real means by which the problem solver can compare assumptions, except by chronological backtracking to determine *why it made the assumption.* Even then, the choice may not be evident.

An alternative design would suggest keeping the reason why an assumption was made with the assumption itself in the MATMS knowledge base. (This is not the approach taken by conventional truth maintenance systems.) That way, the MATMS could automatically resolve inconsistencies — the assumption with the “lower degree of certainty” gets thrown out. Of course this creates another problem, precisely how to assign certainty factors to assumptions. It is exceedingly difficult to place a single measure upon an assumption in order to allow comparisons between assumptions. It is much easier to simply compare a set of assumptions as the inconsistencies arise to determine which to discard. If certainty factors could be reasonably assigned to assumptions, then choosing between two viable default assumptions — an action which is almost impossible now — could be resolved.

Distributed truth maintenance has always been of long-term concern. Although not particularly evident (and not presented in this thesis because distributed truth maintenance is not the immediate purpose of the MATMS), the MATMS incorporates the basic framework for distributed truth maintenance.

Distributed truth maintenance is necessary for any distributed knowledge base in which at least some of the knowledge is replicated. In our domain, trunk and circuit information is replicated in certain knowledge bases. When an agent in one subregion determines that a particular trunk is down and enters this knowledge into its MATMS, then the KBM of the subregion should inform all other KBMs with knowledge of the trunk that it believes that the trunk is down. Mason and Johnson describe the fundamentals of distributed truth maintenance in [26].

In much the same way that the KBM resolves inconsistency among the problem solvers in its local system, the KBM must resolve inconsistency between itself and other KBMs. Distributed truth maintenance is especially interesting because it necessitates distributed control. A single KBM resolving conflict within a single knowledge base implies central control. When KBM agents and the knowledge are distributed, many interesting strategies could be investigated.

Strategies for inconsistency resolution must be further investigated, both in a single agent and across agents. Deciding which assumptions are correct, and which to discard, is the single most important issue facing a problem solving system using the MATMS. As stated earlier, the MATMS efficiently recognizes inconsistencies, but cannot be expected to resolve them in the most general cases.

References

- [1] A. Barr and E. Feigenbaum (Eds.), *The Handbook of Artificial Intelligence, vol. 1*, William Kaufmann, Inc., Los Altos, California, 1981.
- [2] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of Cooperation in Distributed Problem Solving", in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 767-770, August 1983.
- [3] S. E. Conry, R. A. Meyer, and V. R. Lesser, "Multistage Negotiation in Distributed Planning", to appear in *Readings in Distributed Artificial Intelligence*, A. Bond and L. Gasser (eds.), Morgan Kaufman Publishers, California.
- [4] S. E. Conry, R. A. Meyer, J. E. Searleman, "A Shared Knowledge Base for Independent Problem Solving Agents", in *IEEE Proceedings of the Expert Systems in Government Symposium*, October 1985.
- [5] D. D. Corkill, "Hierarchical Planning in a Distributed Environment", in *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 168-175, August 1979.
- [6] J. de Kleer, "An Assumption-based TMS." *Artificial Intelligence* **28**, 1986, pp. 127-162.
- [7] J. de Kleer, "Extending the ATMS." *Artificial Intelligence* **28**, 1986, pp. 163-196.
- [8] J. de Kleer, J. Doyle, G. L. Steel Jr., and G. J. Sussman, "AMORD: Explicit Control of Reasoning." *SIGART* **64**, pp. 116-125.
- [9] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs", in *Numerical Mathematics*, vol. 1, pp. 269-271, 1959.
- [10] J. Doyle, "A Truth Maintenance System," *Artificial Intelligence* **12**, 1979, pp. 231-272.
- [11] E. H. Durfee and V. R. Lesser, "Using Partial Global Plans to Coordinate Distributed Problem Solvers", in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 875-883, August 1987.
- [12] G. W. Ernst and A. Newell, *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press, 1969.
- [13] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", in *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189-208, Winter 1971.

- [14] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton, New Jersey: Princeton University Press, 1962.
- [15] R. G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation", in *IEEE Transactions on Communications*, vol. COM-25, no. 1, pp. 73-85, January 1977.
- [16] J. J. Garcia-Luna-Aceves, "A New Minimum-Hop Routing Algorithm", in *Proceedings of IEEE INFOCOM '87*, San Francisco, CA, April 1987.
- [17] M. Georgeff, "A Theory of Action for Multiagent Planning", in *Proceedings of the Fourth Annual Conference on Artificial Intelligence*, pp. 121-125, August 1984.
- [18] B. R. Hogencamp, *GUS: A Graphical User Interface for Capturing Structural Knowledge*, M. S. Thesis, Clarkson University, Potsdam, NY, January, 1987.
- [19] J. M. Jaffe and F. M. Moss, "A Responsive Routing Algorithm for Computer Networks", in *IEEE Transactions on Communications*, vol. COM-30, no. 7, pp. 1758-1762, July 1982.
- [20] J. Jubin, "Current Packet Radio Network Protocols", in *Proceedings of IEEE INFOCOM '85*, Washington, DC, March 1985.
- [21] A. L. Lansky "Behavioral Specification and Planning for Multiagent Domains", in Technical Note 360, Artificial Intelligence Center, SRI International, Menlo Park, California, November 1985.
- [22] A. L. Lansky "Localized Event-Based Reasoning For Multiagent Domains", in Technical Note 423, Artificial Intelligence Center, SRI International, Menlo Park, California, January 1988.
- [23] G. Ludwig and R. Roy, "Saturation Routing Network Limits", in *Proceedings of the IEEE*, vol. 65, no. 9, Sept. 1977.
- [24] D. J. MacIntosh and S. E. Conry, "SIMULACT: A Generic Tool for Simulating Distributed Systems," *Eastern Simulation Conference*, Orlando, FL, April, 1987 (also available as NAIC Technical Report TR-8714).
- [25] J. Martins and S. Shapiro, "A Model for Belief Revision," *Proc. Non-Monotonic Reasoning Workshop*, AAAI, 1984, pp. 241-294.
- [26] Cindy L. Mason and Rowland R. Johnson, "DATMS: A Framework for Distributed Assumption Based Reasoning," 1988 Workshop on Distributed Artificial Intelligence, Lake Arrowhead, CA, May 22-25, 1988.

- [27] D. McAllester, "A Three-valued Truth Maintenance System." Technical Report Memo 473, MIT AI Lab, 1978.
- [28] J. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks". Bolt Beranek and Newman Inc., in BBN Report 2831, May 1974.
- [29] J. M. McQuillan, I. Richer, E. C. Rosen, "The New Routing Algorithm for the ARPANET", in *IEEE Transactions on Communications*, vol. COM 28, no. 5, pp. 711-719, May 1980.
- [30] P. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol", in *IEEE Transactions on Communications*, vol. COM-27, no. 9, Sept. 1979.
- [31] R. A. Meyer and S. E. Conry, Communications, in: *Expert Systems and Artificial Intelligence*(Thomas C. Bartee, ed.), Howard W. Sams and Company, Indianapolis, 1988, pp. 61-96.
- [32] R. A. Meyer and Charles Meyer, "The Role of Knowledge-Based Systems in Communications System Control," NAIC Technical Report TR-8715, Rome, NY.
- [33] R. Reiter, "A Logic for Default Reasoning," *Artificial Intelligence* **13**, 1980, pp. 81-132.
- [34] J. S. Rosenschein and M. R. Genesereth, "Deals Among Rational Agents", in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 91-99, August 1985.
- [35] Reid G. Smith "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver", in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-10, no. 12, December 1980.
- [36] W. D. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network", in *Communications of the ACM*, vol. 20, pp. 477-485, 1977.
- [37] J. Westcott and J. Jubin, "A Distributed Routing Design for a Broadcast Environment", in *Conference Record of IEEE Military Communications Conference*, Boston, MA, vol. 3, pp. 10.4.4-10.4.5, October 1982.
- [38] D. E. Wilkins, "Domain-Independent Planning: Representation and Plan Generation", in *Artificial Intelligence*, vol 22, pp. 269-301.
- [39] *Baseline Conceptual Design for the DCOSS Data Base*, Defense Communications Agency, July 1985.