COOPERATIVE PROBLEM SOLVING: A KNOWLEDGE PARTITIONING APPROACH

THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
OMTRI NALINI KUMARI

DEPARTMENT OF COMPUTER & INFORMATION SCIENCES
SCHOOL OF MATHEMATICS & COMPUTER / INFORMATION SCIENCES
UNIVERSITY OF HYDERABAD
HYDERABAD - 500 134
INDIA

December 1994

C E R T I F I C A T E

This is to certify that the thesis work entitled **"Cooperative Problem Solving: A Knowledge Partitioning Approach**" being submitted by **O. Nalini Kumari** in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy (Computer Science) of the University of Hyderabad, is a record of bonafide work carried out by her under our supervision.

The matter embodied in this thesis has not been submitted for the award of any other research degree.

Prof. B. E. Prasad

Dr. G. Uma

Supervisors
Dept. of Computer and Information Sciences
University of Hyderabad

13/12/94

Prof. A. K. Pujari
Head
Dept. of C. I. S.
University of Hyderabad

Prof. V. Kannan
Dean
School of M. C. I. S.
University of Hyderabad

DECLARATION

I, Omtri Nalini Kumari, hereby declare that the thesis work entitled **Cooperative Problem Solving: A Knowledge Partitioning Approach** is the bonafide work carried out by me, as per the Ph.D. ordinances of the university. The matter embodied in this thesis has not been submitted for the award of any other research degree to the best of my knowledge.

Omtri Nalini Kumari

To my parents

Smt T.C. Lalithamma
&
Sri O. Chenna Kesavulu

ACKNOWLEDGEMENTS

Though it is difficult to enunciate clearly all the concern expressed for me by several people, this is a small attempt to acknowledge their direct or indirect contribution and support in making me complete the research programme.

In all my humbleness, I express my sincere gratitude to my supervisors Prof. B. E. Prasad and Dr. G. Uma for their guidance and support. I have benefited from discussions with them and their valuable suggestions. I thank them for getting important literature and providing me with necessary facilities.

Further, Prof. Prasad's confidence in me and concern for me brought me back to this university for doctoral work. He has been very kind to sit for long hours and advise me inspite of his busy schedule and administrative responsibilities at UOH. I thank him for doing the needful even being overseas.

Dr. Uma has always been friendly, ready to discuss any time (even at home) and give helpful hints in many ways. She gave high priority to my work and has been very keen on the successful and fast submission of thesis. I specially thank her for patiently going through various versions of the thesis and making appropriate corrections.

I thank Dr T S Perraju for taking trouble to come over here and helping with corrections to earlier drafts of the thesis, for giving me aerospace application rules and other help. I thank Dr G Saraswati for being my domain expert for the medical diagnosis case study. Further, I thank Dr Ing-Ray Chen, Dr Klaus Fischer, Dr Denis Gagne, Dr K S Trivedi, Dr Narahari, Dr Sharma, and Dr Murthy for their papers and help.

I thank Prof AK Pujari, Head, Dept of Computer Science for his suggestions and encouragement. Further I express my thanks to Dr PS Rao, Mr Atul Negi, Dr PV Reddy and Mr KN Murthy who were always ready with their suggestions and help. I also thank Prof PG Reddy, Dr AS Reddy, Dr Arun Agarwal, Dr KC Reddy and Dr P N Girija and other faculty members for their concern and interest in my welfare.

Further, I wish to express my thanks to Prof V Kannan, Dean, School of MCIS, and Dr Upendra Rao, Director, Computer Centre for providing me with necessary facilities and encouragement.

My thanks are also due to Mr Kameswara Rao, Mr Balakrishna, Mr EA Vinod Kumar, and the staff of AI Lab, department, school, computer center, hostel and administration for their cooperation.

I thank my friends (and co-research scholars) Mr EVRC Mohan Reddy and Mr Vasudev Varma for the patient proof reading of the thesis and other suggestions. Mohan needs additional mention for books and kind help in various ways particularly during the last stages of thesis completion. I also thank Dr Phanindra Babu, Mr Prem Kumar and Ms Savita for sending the necessary papers, Mr Ananth Rao and Mr Sateesh for figures.

I thank my close friend Ms P UmaRani for her company at HCU and all the help she would be ready with even before I could realize the need. I also thank her family members Shri Ramgopal, Akhila, Smt Bhavani and Shri Radha Krishna Chetty, and

Dr Saraswati's family members Dr G S Raman and Arunkumar for their help and affectionate treatment.

Our family friends Dr. Mohan C. Vemuri and Mrs Aruna have been a source of encouragement and moral support and made me feel at home during my stay at Hyderabad. Their daughters Bindu and Sindu made me rejoice in their affectionate company. I thank all of them, other family members, Dr Mohan's students and Kishan.

Then, I thank Dr Ravindra Kumar for his kind help at HCU. My thanks are also due to the families of both Prof Prasad and Dr Uma.

Then, I take this opportunity to thank Dr M M Naidu, Mrs Satya and their family for their help and support at Tirupati ever since I joined SVU. I am lucky to be cared for by them. My thanks are also due to Dr K V Madhumurthy, Mr P Anajaneyulu, Dr Raman Rao, and many other professors at Tirupati.

I also thank my co-research scholars Shiny, Ravindra Sharma, Lakshmi Narayana, U.V.Ramanaiah, Raju and Bharadwaj for being kind to me. My stay in the hostel is made memorable in the company of Rukmini Devi, Nancy Sailaja, Nirmala, Hema, Rama Devi, Anuradha, Srikala, Indira(s), Poornima, Uma(s) and other hostel mates. My thanks also to Chakravarthy, Ramesh Babu, Bhanu Murthy, Vijaya Lakshmi, Vedavathi, Geetha, Saigeetha, Vijaya, Radha, Nirupama, Rekha, Jayaprabha, Saro and so many others.

I am grateful to my parents Smt Lalithamma and Sri Chenna Kesavulu for being my first teachers and for the careful planning of my studies and career. Along with my parents, my sisters Bhanumathi, Swarna Latha, Revathi, and my brothers Subhash Chandra Bose and Diwakar Babu are always with me with their abundant affection and support to see me succeed in all my endeavours. I acknowledge their direct help in the completion of my Ph D work as well. My (maternal) grand father Sri Chengalraya Naidu, and (paternal) grandmother Smt Govindamma deserve mention for their protection and affection. I thank Bhanu's husband Mr Surya Prakash, several of my aunts and uncles and other relatives who helped us. My neice Neeharika deserves special mention for all her love and good wishes for me.

Finally, I thank the Council for Scientific and Industrial Research for providing the financial support by awarding their Senior Research Fellowship and Sri Venkateswara University, Tirupati for being kind enough to grant me two years of study leave to complete my doctoral work.

I kneel before the Almighty for his kind blessings and helping me reach this stage, and mother Nature for all the inspiration.

ABSTRACT

Knowledge distribution plays an important role in Cooperative Problem Solving. It is very closely related to, and actually leads to task decomposition in some domains like monitoring applications. However, the problem of task decomposition has not been adequately addressed by Distributed AI (DAI) community. An appropriate knowledge distribution also balances the load, provides fast access to the knowledge base, reduces pattern matching time for reasoning, and reduces information exchange and inconsistency problems. Further, it facilitates distributed reasoning and modelling of agents.

This thesis proposes a knowledge distribution approach to Cooperative Problem Solving. The main objective is to partition domain knowledge and allocate the resulting subsets to agents in a distributed production system statically as well as dynamically. Both production rules and data are distributed to agents in order to speed up processing and minimize communication. The second objective is to provide mechanisms for distributed reasoning to seek data and partial results from other agents during actual problem solving.

The need for the present work arises due to the following reasons:

- Many distributed AI domains like Hearsay II adopt a general functional decomposition strategy which is domain dependent. Also, efficiency issues like load balancing and faster processing are usually not considered.
- Related work in single processor and multiprocessor systems is not suitable for DAI systems *per se*. Besides considering factors like distance, heterogeneous partitioning and working memory distribution, distributed reasoning has to be facilitated for effective problem solving. Moreover, the complex interdependencies among the knowledge subsets in DAI systems require special consideration while distributing knowledge dynamically.
- Since optimal partitioning is an NP-complete problem, heuristics which obtain a good partition quickly may some times be preferable to costly techniques that obtain a better solution. In case an optimal partition is also necessary, the partition obtained as above can be used as a good initial partition to generate the optimal one. Such a partition can speed up the process of obtaining optimal partition and/or improve the quality of the solution. However, there is no proper heuristic to do this.

In this thesis, an attempt has been made to solve these in three stages.

First, a linear time static knowledge base partitioning heuristic is proposed. Given a rulebase and the proportion in which rulebase subsets are to be obtained, it constructs a knowledge graph, generates a long spanning tree and partitions the rules in the given ratio by cutting along its chain. Rules are assigned to subsets such that interdependencies and data inconsistencies are minimized, and load is balanced. The metaknowledge maintained with each subset facilitates distributed reasoning.

The static partitioning algorithm is extended for distributing knowledge dynamically. Techniques for distributing knowledge dynamically for load balancing, i.e., with minimum local changes and by repartitioning the entire knowledge graph based on the run time behaviour, are proposed. Dynamic distribution for catering to changes to the knowledge base, and problem based dynamic allocation are also discussed.

Finally, a distributed reasoning heuristic is proposed for obtaining information from other agents. This is necessary when an agent cannot proceed with local inferencing and needs nonlocal information. The decision of what information has to be asked for is made based on rule firing likelihoods and dynamic occurrence of data. Requests are sent to agents with high utility value which is calculated based on the agent's past performance and the ability to generate or send that particular piece of information.

Working of these algorithms is tested with an aerospace vehicle checkout application and a medical diagnosis application.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the motivation and the need for research on knowledge distribution. Section 1.1 presents the motivation. Section 1.2 gives a brief introduction to Distributed Artificial Intelligence (DAI) and its subareas. Section 1.3 describes the styles, modes and forms of cooperation used in Cooperative Problem Solving (CPS). Section 1.4 discusses the issues involved in CPS and explains the role of knowledge distribution in each. Section 1.5 describes our model of the CPS system and section 1.6 defines the objectives of the research. Finally, section 1.7 presents the organization of the thesis.

1.1 Motivation

The area of Distributed AI has gained importance as a major paradigm for problem solving in computer science as well as in a variety of disciplines like robotics, linguistics, organization theory, biology and psychology [30, 50, 113]. This can be attributed mainly to the following reasons.

- Unlike traditional AI systems, DAI offers advantages like reliability, availability and speedup through parallelism.
- It concentrates on the aspects of communication and cooperation which are very important for solving complex real life problems. For example, diagnosis and treatment of a patient, military decision making and interpretation of geological data require cooperation among the participating experts [131]. Even a simple task such as diagnosis of a minor ailment often needs cooperation between a physician and a laboratory technician. DAI helps to understand the cooperation that is essential among (groups of) problem solvers.

- DAI helps to develop new theories, methods and tools that are needed to understand, design, construct and test such complex systems.

In order to exploit the advantages of DAI, tasks must be decomposed and distributed to the problem solving entities called agents. Task decomposition and allocation are, therefore, important issues in DAI. However, the difficulty in decomposing a given DAI problem into independent subproblems, the uncertainty present in the problem domain and the possible geographic separation of agents necessitate a lot of interaction among the agents. Since communication is costly and will remain so at least in the foreseeable future, it must be kept at a low level such that the improved performance due to parallelism does not get abrogated due to excessive communication. In this regard, it is observed that information exchange can be minimized by distributing tasks, knowledge and data with minimum interdependencies.

Once knowledge, data and tasks are distributed to agents, in the course of problem solving, the agents need to reason about what information they need to exchange in order to solve the problem, with whom, and when. Requesting for the most useful information from potential donors at the appropriate time, and acceptance of roles by agents permit smoother cooperation without much communication.

Proper distribution of knowledge and data helps in task decomposition and allocation, and facilitates distributed reasoning if metaknowledge about other agents is provided with each agent. Further, it improves the efficiency of the system in terms of faster processing, load balancing with less communication, minimizing data inconsistencies and sometimes the maintenance of the knowledge base.

However, in the earlier work reported in literature, knowledge partitioning is ignored as a tool for obtaining optimal task decomposition besides having advantages like efficient distributed inference. This thesis explores the knowledge partitioning approach for achieving these in Cooperative Problem Solving.

1.2 Distributed Artificial Intelligence(DAI)

DAI is concerned with the collaborative solution of a global task by a distributed group of entities. Entities range from simple processing elements to complex ones exhibiting rational behaviour. Problem solving is a collaborative process in the sense

that mutual sharing of information is necessary to allow the group as a whole to produce a solution, or to successfully accomplish the global task. The group of entities is distributed in that both control and data are logically, and often geographically distributed. According to Bond and Gasser [9], *DAI is concerned with distributing* ***and coordinating knowledge and actions in multiple agent environments.***

1.2.1 Arenas

Depending on the number of problems being solved, their nature and the way in which they are distributed, three arenas are identified: Distributed Problem Solving, Multi Agent systems and Parallel AI. Each of these is elaborated below.

Distributed Problem Solving (DPS)

A single problem is envisioned for the entire system of agents and typical problem solving involves *problem decomposition, subproblem allocation, subproblem solution* and *solution synthesis.* Depending on the application, the complexity and the importance of each of these phases vary. DPS mainly deals with interactions of groups of intelligent agents which act together.

DPS differs from Distributed Processing in many ways [115, 120]. In Distributed Processing, activities of agents carrying out independent and disparate tasks are to be synthesized. Major motivation is to reconcile conflicts arising from these activities for exploiting advantages of multiprocessing. Distribution implies spatial distribution of data, and usually there is no distribution of function or control. Further, most processing is done at a central site, while remote processors are basically data collecting entities. There is a lack of substantial cooperation in most Distributed Processing systems.

In DPS, available resources do not have predefined roles, agents are homogeneous and can solve any of the subproblems. The main issue is to develop frameworks for cooperation between willing entities rather than enforcing cooperation as a compromise between unwilling or incompatible entities. A DPS system may be adaptive to uncertainty in problem solving knowledge, but not to alternative problem contexts or to changing problem solving roles for modules [9].

Multi Agent Systems(MAS)

These are also called as Collaborative Reasoning Systems [109] and are concerned with coordinating the knowledge, goals, skills and plans of autonomous intelligent agents so that they can jointly take actions or solve problems [9]. Agents may work towards a single global goal or multiple goals that interact in some way. Like DPS system agents, these agents also share knowledge. Achieving coordination is quite difficult, for there may be situations as in open systems [60, 61, 62] where there is no possibility for having global control, globally consistent knowledge, globally shared goals or goal success criteria, or even a global representation of the system. A multi agent system may be able to form and restructure coordination frameworks based on emerging contexts and changing problem solving roles without the intervention of a programmer.

Parallel AI

This is concerned with developing parallel computer architectures, languages and algorithms which are primarily directed towards solving performance problems of AI. Examples of such systems include AGORA [8] and AF [56]. Parallel AI differs from the other two arenas because it does not aim to advance the conceptual understanding of the nature of reasoning and intelligent behaviour among multiple agents. For example, Connectionist systems comprise a collection of a large number of computation elements which need not be intelligent. These systems may be able to adapt to temporal uncertainty, but not to alternate solution paths and loss of problem solving knowledge [9].

However, DAI focusses mainly on approaches to the problems of distributing and coordinating knowledge and actions, and hence on DPS systems and Multi agent systems.

1.2.2 Distributed AI vs. Decentralized AI

A very closely related and much talked about area in 1990's, particularly in Europe is Decentralized AI (DzAI) [30]. It is concerned with the activity of an autonomous agent in a multi agent world. The term *agent* is used in a broader sense to designate

an intelligent entity which acts rationally and intentionally with respect to its own goals and the current state of the knowledge. Each agent has its own existence, which is not affected by the existence of other agents. Several autonomous intelligent agents coexist and may collaborate with other agents in a common world. Each agent may accomplish its own tasks, or cooperate with other agents to perform a personal or global task [30, 112].

Both DAI and DzAI have common interest in the behaviour of distributed entities. However, in DAI, a global task is initially defined and the problem is then to design distributed entities to solve this global task. The main issue here is to study the distribution and collaborative solution of the given task. In contrast, DzAI is concerned with how a group of predefined decentralized autonomous agents are able to achieve tasks that may be of interest to a single agent or several agents. The main issue in DzAI is the study of the structure of the autonomous entities to provide insights into what kinds of problems these entities are able to solve.

1.3 Cooperative Problem Solving(CPS)

Problem solving in both DAI and DzAI can be divided along the axis of cooperation. Though agents cooperate most of the time, due to reasons like conflicting goals, agents may be noncooperative in some domains [106, 132]. In this thesis, we are concerned about agents that cooperate to solve a single problem, and therefore about Cooperative Problem Solving.

1.3.1 Cooperation

Cooperation is required for coordinating the actions of multiple problem solvers. Werner [122, 123] defines cooperation as *the process of mutual social action that leads to individual and social goals. It emerges out of the mutual adjustment of intentions of the participating agents which results from communication, from the proclivity to engage in a cooperation style, and from the cooperation style itself.*

Though cooperation is usually considered to be a form of interaction which results from communication, Genesereth, Rosenschein, Ginsberg et al. [52, 54] discuss about cooperation that is essential when communication is not possible between agents.

Their model is based on game theory techniques and makes use of payoff matrices. Certain assumptions about rationality of the agents helps to make reasonable choices without communication.

1.3.2 Styles of cooperation

Depending on the degree of cooperation extended to another agent, agents can interact in a variety of styles, viz., *totally cooperative, self-interested, antagonistic,* and *self-destructive,* or a combination of these [122]. These styles reflect the intentions and the compromise an agent is willing to make. However, these styles are agent and context dependent. They evolve based on the ranking and evaluations an agent makes of its own and other agents' goals.

1.3.3 Modes of cooperation

In an orthogonal direction, Zhang and Bell [131] classify cooperation among experts into four predominant modes based on their interdependence relationships: *horizontal cooperation, tree cooperation, recursive cooperation* and *hybrid cooperation.*

Horizontal cooperation is seen when each expert in the cooperative group can get solutions to problems without depending on other experts. But, if the experts cooperate, possibly using different expertise and data, they can increase confidence in their solutions. For example, the cooperation between doctors for diagnosing patients often illustrates horizontal cooperation (eg. the *second* opinion syndrome). Consultation and comparison of opinions add significantly to the value of the diagnosis.

Tree cooperation is demonstrated in situations where a senior expert depends on a junior expert in order to get solutions to problems. For example, a chief engineer's decision often depends on the work of junior engineers.

Recursive cooperation is seen in situations when different experts depend on each other in order to get solutions to problems. For example, in order to interpret geological data, geophysical experts and geochemical experts often depend on each other in a recursive way. There is a recursive dependence when a geophysicist asks a geologist to perform **a** subtask **who in** turn depends on the geophysicist **for** solving

some other (sub)subtask.

Hybrid cooperation is manifested in situations where different experts use horizontal cooperation at some level in an overall tree or recursive mode of cooperation. On the other hand, they could equally use tree or recursive cooperation at some point in an overall horizontal cooperation. An example of the former is where several opinions are obtained in order to optimize the quality of the final result coming from the engineers at a given level of seniority in the tree cooperation.

1.3.4 Forms of cooperation

According to Smith and Davis [115], cooperation among agents may be exhibited in the form of *task sharing* and *result sharing.* This refers to the content or what (information) is to be communicated between agents, and its use by an agent for cooperative problem solving.

In *task sharing*, the given problem (or task) is divided into suitable subproblems (subtasks) and allotted to different agents for solving. Agents cooperate by sharing the subtasks to achieve the main task. However, this requires that a problem be decomposable into more or less independent subproblems.

If the given problem cannot be easily decomposed into nonoverlapping subproblems, there may be a necessity to share the temporary partial results obtained by different experts. This is called *result sharing.* Recursive or hybrid cooperation may be necessary in that the individual agents must cooperate by exchanging partial results and other information. In Functionally Accurate/Cooperative approach to problem solving [38, 82, 83], network problem solving is structured so that nodes cooperatively exchange and integrate partial, tentative, high-level results **to** construct a consistent and complete solution.

While the exchange of intermediate results is the main thrust in result sharing, sharing of external input data may be treated as data sharing. In this thesis, we consider sharing of both intermediate results and data, and call it as *information sharing.*

***Joint** action* [73] is another form of cooperation when the actions of agents are intertwined, the problem is not decomposable into independent subproblems and

agents must share partial results and future plans. However, this differs from task and result sharing because it is a reciprocal process in which participating agents augment their actions to comply with those of others. Relevant examples include several agents lifting a heavy object, musicians in an archestra, driving in a convoy, and playing cricket. Jennings proposes *joint intentions* as a model of Multi-Agent cooperation [73].

1.4 Role of Knowledge Distribution in CPS

The important issues in CPS are task decomposition and allocation, coordination, coherence, communication, resolution of disparities and modelling of other agents. Knowledge distribution plays a crucial role in most of these issues as explained below.

1.4.1 Task Decomposition and Allocation

Task decomposition becomes necessary when a task requires more knowledge or more resources than what are available with an agent. A task must be properly described and formulated so that it can be decomposed and its subtasks are allocated to agents [9, 109]. *Task description* is the statement of the problem and the expression of dependencies among subtasks in a suitable language. *Task formulation* is a representation of the problem which decides on the boundaries of the problem and on what is known and what is not. Depending on the task description and formulation used, a task may give rise to different subtasks and different interdependencies. Dynamically changing patterns and contexts may require decomposition to undergo revision. Therefore, intelligent approaches to task decomposition must consider representation of tasks as well as dimensions of decomposition.

The dimensions of decomposition include knowledge, location, abstraction, time and available operators that can be applied to perform subtasks. Along these dimensions, decomposition can be done based on abstraction, data dependency and data partitioning, effective use of resources, division by function or product and other organizational and management criteria. Commonly used methods to obtain such decompositions are inherent decomposition, hierarchical planning,

decomposition by programmer, load balancing, minimally connected subgraphs and subtask aggregation [9].

Once the main task is divided into suitable subtasks, *task allocation* involves the assignment of tasks to agents that will actually perform them. Criteria for task allocation include bottleneck avoidance, fitness to specification, overlap in roles, uncertainty avoidance, reliability, urgency and resource consumption. Resource allocation is a related problem and is a way of prioritizing subtasks. An example is the Scientific Community [78] in which a sponsor based resource allocation is made.

There is little work reported in DAI literature that addresses automated problem formulation and decomposition. Greater effort has been put into task allocation mechanisms assuming *a priori* knowledge and task decomposition. For example, Contract Nets [115] and DVMT [36, 37, 84] address opportunistic allocation of tasks assuming the subtasks and knowledge partitioning are provided by the designer. Actors [78] treat the allocation decision as dynamic but task description and decomposition decisions are not addressed by them.

In this context, we see that all the conceptual distances that define distribution in a DAI system, as mentioned in [9], refer to the use of knowledge in some form or the other. For instance, computational cost, spatial distance and temporal distance define distribution based on the cost of using a piece of knowledge with respect to location or time; the others, viz., logical distance and semantic distance use logical dependency between portions of knowledge base and their practical use respectively.

Further, tasks that have strong knowledge production and consumption relationships may be grouped together. This introduces task coordination and precedence constraints that affect allocation decisions because a node cannot work until another has finished its task [36]. These task precedences are a type of task interdependency. Without adequate interaction capacity, matters of consistency, definition or direction should be addressed by the node with the most global view [124].

In addition to knowledge, data plays an important role in task distribution. Data dependencies among the tasks along the axes of semantics, logical dependencies or temporal dependencies can serve as bases for decomposition choices. Tasks can be decomposed by taking into account the natural or dependency related partitions in input data. In Distributed Sensor Nets (DSN) [83, 85], spatial distribution of data

provides a natural basis for task decomposition. It is also related to division by product or function and resource minimization. Further, metrics may be defined to measure these dependencies among data and to obtain a suitable task decomposition.

From the foregoing discussion, it can be observed that task distribution is closely related to distribution of knowledge and data (input data or intermediate results). Knowledge/data distribution may itself lead to task distribution. This is particularly true for some applications like data driven distributed expert systems and distributed data bases. Therefore, we approach task decomposition problem from the knowledge/data base partitioning perspective.

1.4.2 Coherence and Coordination

Coherence refers to how well the system behaves as a unit with respect to the solution quality, efficiency, clarity, or graceful degradation [9] of the whole system or only some part of it. Coherent behaviour requires satisfying three conditions namely *coverage, connectivity* and *capability*. *Coverage* must ensure that all necessary portions of the overall problem are included in the activities of at least one agent; *connectivity* implies that agents must interact in a manner which permits the solutions for the covering activities to be developed and integrated into an overall solution; finally, coverage and connectivity must be achievable within the network's *capability* such as communication and computation resource limitations.

Coordination is a property of interaction among groups of agents performing some task collectively [122, 123]. The degree of coordination exhibited is the extent to which agents can avoid extraneous activity in achieving their primary ends. Effective coordination implies some degree of mutual predictability and lack of conflict.

Coordination and coherence can be increased by reducing the dependencies among agents through a good task decomposition and increasing the supply of resources. Better coordination leads to greater efficiency in coherence through reduction in articulation work.

We facilitate better coordination and coherence by a good task decomposition which in turn is achieved through partitioning data and knowledge with minimum dependencies. Task decomposition and allocation done considering capacities of agents, communication links and distance between agents, with an aim to minimize

the interdependencies and the information exchange ensure *capability, connectivity and coverage.* A static allocation reduces the need for coordinating the task allocation. If the situation changes dynamically, dynamic distribution is the solution. We consider both static and dynamic distributions of knowledge and data, and reasoning about nonlocal information that is required for solving a subproblem.

In this context, it is both important and interesting to note that Malone and Crowston [90] also define *coordination as a process of managing interdependencies among activities.* They explain the possibility of managing the activities and monitoring solution progress by making use of dependencies like producer/consumer relationships, simultaneity constraints and task/subtask relationships.

1.4.3 Interaction

It is usually difficult to decompose a given DAI problem into nonoverlapping subproblems. Because of this, agents are required to interact so that they can combine their efforts and solve the problem. Interaction may be defined as a type of collective action in MA or DPS systems wherein one agent takes an action, or makes a decision that has been influenced by knowing about another agent [9]. Multiagent interaction needs to consider issues like

- among whom the interaction takes place
- when the interaction occurs (temporal or causal)
- what the context is
- how the interaction is accomplished
- why the interaction occurs
- what the basis of communication is.

Interaction for cooperation needs additional features like sharing of common knowledge and a communication protocol that allows for differences when agents possess disparate knowledge and use different knowledge representation schemes.

One of the major goals of knowledge distribution is minimizing interaction. In this thesis, we are mainly concerned with information exchange that is essential for

resolving the incompleteness of local knowledge or data. Seeking and sharing of partial results can be minimized by reducing the dependencies among the subtasks. In data driven expert systems, the communication required for task decomposition and allocation can be completely eliminated by a good partitioning of knowledge and data which takes care of such dependencies. In other systems, it leads to task decomposition with less communication. We provide strategies for reducing the communication for information exchange further and for better coordination.

1.4.4 Agent Modelling

Modelling of other agents (by an agent) is important for meaningful interaction, communication, coordination, control and task allocation. Agent models are useful for predicting the requirements and effects of events that cannot be directly sensed. This results in reduced communication by reducing the necessity of sharing conforming information, and leaving communication channels free for surprising, unpredictable information. Agent models are also useful for evaluating credibility, usefulness, reliability and timeliness of data.

Task allocation requires knowledge of what potential tasks the agents can perform. Knowledge of agent capabilities and responsibilities provide a way of reducing task-allocation overhead. Coordination requires an agent to reason about its own activity, its effects on other agents as well as those of others and thus have *network awareness* [36]. Knowledge about solution progress is important to detect deadlocks and liveness, and to predict whether it will be useful to exchange any information with another agent. Knowledge of beliefs, plans, goals and actions of others is necessary for reasoning about communication and synchronization of plans. For meaningful interaction, agents must have at least implicit knowledge about each other on communication protocols or languages and should know what reaction to expect from another on sending a message. Further, modelling requires knowledge about whether agents are self-organizing or adaptive, their interrelationships, default expectations, knowledge and beliefs of others, and about the communication links. However, completeness of models of other agents is difficult to achieve because it requires duplication of processing and consideration of several conflicting issues like communication, computation, speed and efficiency.

Examples of agent models include acquaintance data bases of MACE [49], partial

global plans [34, 35], production lattice models, process assembly networks and problem solution graphs [9].

As indicated earlier, agent models improve efficiency by focussing activity or by directing search. Knowledge of the data and resource requirements of other agents prevents unnecessary communication, and may engender early communication of important data. Agent models also help to know where to get particular information and about its availability [29, 65]. Focussed addressing reduces bidding delays and overheads. We achieve this by providing metaknowledge [65] with each agent and using utilities of agents for obtaining information. In addition, agents are defined by the knowledge and data assigned to them and hence the subtasks to be carried out.

1.4.5 Reconciling Disparities

Disparities that result from incomplete knowledge can be resolved by identifying and communicating the required knowledge. This may require reasoning about the knowledge state of separate agents [9]. Achieving common knowledge among agents, however, can be impossible in the face of communication unreliability. Provision of metaknowledge about the information requirements of other agents, and the knowledge of agents which could provide the required information help to resolve the incompleteness of local data or knowledge.

1.4.6 Knowledge and Structures for Task Decomposition

There has been very little research into what kinds of knowledge and what structures are required for automatic task description and decomposition. These are necessary if agents in a DAI system are to jointly construct and recognize their own problems. Integrating multiple perspectives and ideas from other disciplines like distributed computing is essential to solve the problem [9]. In this direction, we have developed graph-theoretic techniques and successfully used them for this purpose [98, 99].

1.4.7 Knowledge Base Access

In general, it is difficult for agents to have easy and fast access to large **and** possibly shared knowledge[9]. A few attempts have been made to integrate knowledge base access methods with **DAI** blackboard shells like GBB [23, 48] and BB1 [94] which incorporate high level knowledge structuring and knowledge access mechanisms, and pattern-directed retrieval of data. MACE [49] provides associative data bases within agents. For rule bases, about 90% of the time is consumed in pattern matching [44].

Knowledge base partitioning is an attempt in this direction to make access faster with smaller subsets. However, arbitrary partitioning will increase communication overhead. Therefore, techniques which can partition and allocate knowledge such that both access time and resulting communication overheads are reduced are of paramount importance. We make an attempt in this direction.

1.5 Our Model of the System and Agents

A Cooperative Problem Solving system may belong to any of the following categories: DPS, MA, Parallel AI or DzAI systems with cooperating agents. The assumptions about our model of agent and the Cooperative Problem Solving system are given below.

Agent

An agent has its own local memory and processor. Communication among agents is by message passing. Agents could be heterogeneous with respect to their capacities, and the knowledge possessed by each. Each agent is assigned a different portion of the knowledge base. However, the reasoning model and the knowledge representation scheme (production rules) used by all the agents are assumed to be same.

External input data may be obtained through sensors. The extent of duplication of data may vary from nil to full. Similarly, knowledge may or may not be duplicated. However, unless functional completeness is required, we do not prefer duplication of knowledge. Therefore, our agents can be considered as semi-autonomous problem solvers which cooperate by exchanging the required information.

Cooperation

As long as there are no other pending tasks, agents answer other s' requests for information immediately.

Mode of cooperation could be horizontal, tree or recursive, depending on the type of data or partial result exchange between agents. Both task sharing and result (information) sharing forms of cooperation are used. However, task sharing is made implicit by the distribution of knowledge and data, predefining the subtask to be carried out by each agent, particularly when knowledge is distributed statically. Even if knowledge is distributed dynamically, the task assignment remains fixed until knowledge distribution is changed again.

A single problem is considered for solving by the entire system of agents. There is no duplication of processing by the agents. But, the questions to be answered are: How is the main task decomposed and allocated? What criteria need to be considered in this? What is the objective? How can tasks be distributed evenly among the agents with minimum communication? When should the agents be interacting with each other? How should their activities be coordinated? Knowledge partitioning provides solutions to these and leads to effective cooperation.

Subarea

Since our interest is in solving a single task, the goal of all agents is the same. Each agent attempts the subtask assigned to it while cooperating with others. Futher, as knowledge partitioning helps to decompose the main problem, knowledge partitioning may be considered as a DPS problem.

Besides making task decomposition and allocation easier, in our approach, coordination is facilitated by a proper distribution of knowledge and data, providing metaknowledge and distributed reasoning algorithms to resolve incompleteness of information. Coordination, communication, cooperation and resolution of disparities are achieved by exchanging useful information. The distributed reasoning and coordination problems are applicable to MA and DzAI systems as well.

The proposed knowledge distribution techniques for fast access to knowledge and data, and the methods for reducing communication are a step towards improving

the performance of the system. This aspect is specific to Parallel AI systems.

Similarly, knowledge distribution can be considered as a DzAI problem as it helps to define agents by allotting portions of knowledge, data and some metaknowledge about the information requirements of agents. This is also true as we assume a fixed number of agents to be already existing in several cases of distribution.

Since we assume that agents attempt to solve a single problem in a cooperative fashion, it is basically a contribution to Cooperative Problem Solving.

1.6 Objectives of the Research

The main objectives of this research are to develop

- a fast domain independent, heterogeneous knowledge base partitioning technique which doesn't duplicate knowledge, balances load, reduces communication, facilitates reasoning, task decomposition and reduces inconsistency problems; the partition obtained can be used as it is or serve as a good initial partition for generating an optimal partition
- techniques for dynamic distribution of knowledge and data
- a distributed reasoning strategy for further effective information exchange by seeking useful information from satisfactory and potential donors.

1.7 Organization of the thesis

Chapter 2 surveys related work on knowledge distribution. It gives an account of the types of knowledge, importance of kncwledge distribution in general, and tasks involved in knowledge distribution. Further, it discusses various types of distributing knowledge and data and their effect on exchange of partial results. Lastly, it presents a survey of earlier work in DAI as well as related areas like parallel production systems and single agent systems.

Chapter 3 describes domain independent techniques for statically distributing knowledge based on data dependencies. It proposes a linear time knowledge partitioning heuristic for a connected knowledge graph and another for dealing with

disconnected components. Further, obtaining functional decomposition is discussed. Then, another method is presented for allocating the resulting subsets to different agents.

Chapter 4 presents two case studies, viz., an aerospace vehicle monitoring application, and a medical diagnosis application.

Chapter 5 deals with dynamic distribution of knowledge. It presents techniques developed for balancing the load dynamically by limiting the necessary changes to neighbouring agents as far as possible, by repartitioning and reallocating the knowledge base, and for adaptive reorganization to accommodate the changes to the knowledge base as a result of newly acquired knowledge. Finally, problem based dynamic knowledge distribution is discussed using examples from medical diagnosis domain.

Chapter 6 presents the distributed forward reasoning strategy to resolve the incompleteness of knowledge which arises due to distribution. Distributed forward chaining production systems are considered, and examples from the above domains are presented.

Finally, chapter 7 summarizes the main contributions and concludes with directions for further research.

Chapter 2

Knowledge Distribution

This chapter introduces the terminology and concepts, and discusses the related work on knowledge distribution. Section 2.1 presents the different types of knowledge. Section 2.2 discusses the tasks involved in knowledge distribution and explains the various types of knowledge distribution with emphasis on communication aspects. Section 2.3 presents the techniques used for partitioning and allocating knowledge. Section 2.4 gives an account of the related work on knowledge partitioning in single agent problem solving systems, parallel production systems, and Distributed AI systems.

2.1 Types of Knowledge

Knowledge can be broadly classified into two types: *domain specific* knowledge and *control* knowledge. While domain specific knowledge is useful in solving the actual problems posed to the system from the external world, *control* knowledge is used for task distribution, result synthesis, monitoring, coordinating, organizing or improving the efficiency of the system [88]. Lun and Mac Leod [88] describe several classes of agents based on the specific activities carried out by them, viz., domain specialists, knowledge managers, knowledge facilitators, knowledge transformers, knowledge monitors, and interface agents. Similary, knowledge can also be divided into several types based on the specific activity it is intended to perform.

- *Domain knowledge* is application specific and is usually static. The other types of knowledge are general.
- *Management knowledge* is used to control the problem solving cycle. Typical activities include creation and termination of agents, and scheduling and coordinating their activities. It can also aid in negotiations, decomposing **and**

allocating tasks, synthesizing results, and intelligent deployment of resources.

- *Monitoring knowledge* is used for monitoring and analyzing intelligent behaviour by tracing and debugging. It is important for identifying bottlenecks and inefficiencies. However, this is not part of problem solving.
- *Facilitating knowledge* facilitates and expedites problem solving by reconciling conflicting information, merging disparate perspectives, resolving uncertainty and managing voting schemes.
- *Transformation knowledge* converts information from one form to another acceptable to user agents. This is done by maintaining continuous flow of information across various data levels that arise from multiple levels of abstraction in problem solving.

Ishida [71] proposes another type of knowledge called *organization knowledge* necessary for interactions among agents. It is used to dynamically decompose or combine agents based on the utilization of the (idle) resources, message traffic and communication overheads for better performance of the system. It is a combination of *management knowledge* and *monitoring knowledge.*

In a different direction, knowledge is also classified as *procedural knowledge*, and *problem solving knowledge* [126].

- *Procedural knowledge* is intended to maximize the use of data processing technology to provide efficient processing of the activities. It contains well-thought out, well-tested and well-structured organizational knowledge. In MOAP [126], this is described in terms of tasks which can be broken down into a hierarchy of subtasks.
- *Problem solving knowledge* is intended to maximize the use of AI technology to provide reasoning capabilities. It describes the strategies and serves as advice to the activity coordinator when procedural knowledge is insufficient to perform an organizational activity. This consists of knowledge related to facts and knowledge related to the missing parts of procedural knowledge.

As mentioned in chapter 1, *modelling knowledge* [9], i.e., knowledge required by an agent for having models of other agents can also be considered as another

type of knowledge. This, in turn, can be classified as knowledge about capabilities, resources, solution progress, beliefs, plans, goals and intentions of agents, their self-organizing or adaptive nature, the interagent relationships and the communication protocols.

Knowledge may exist in various forms and representations [126] requiring transformation for use by different agents. For example, knowledge may exist in the form of C programs, Pascal programs, rules and cases. To support organizational activities, an agent may need to contain knowledge in more than one type and/or form.

Our interest is in domain knowledge represented in the form of rules and other information in the form of facts, external input data and partial results (derived data) obtained during the problem solving. We chose rules to represent knowledge as these are the most widely used knowledge representation scheme in expert systems. In order to differentiate domain knowledge in the form of rules from the information available in the working memory (data and partial results), we refer to the former as *knowledge* or *expertise*, and the latter as *partial results* or *information.*

2.2 Knowledge Distribution

Knowledge distribution plays an important role in problem solving in several domains.

- In Distributed AI systems dealing with massive data and/or knowledge, data and knowledge must be distributed such that interdependencies are minimized among subsets for minimizing communication and efficient problem solving. In some cases, an inherent decomposition may be possible while in others an explicit partitioning strategy is necessary.
- In parallel production systems, a good partitioning facilitates faster pattern matching and parallel exploration of the search tree by many processing elements.
- In single agent problem solving systems, when the knowledge base is too large to be accommodated at once in the main memory, suitable portions have to be dynamically loaded.

Further, when appropriately done, knowledge distribution helps in task decomposition, load balancing, agent modelling, distributed reasoning with less information exchange and knowledge base maintenance.

2.2.1 Tasks involved in Knowledge Distribution

Distribution of knowledge involves two tasks: *knowledge (base) partitioning* and *knowledge (subset) allocation.*

Knowledge partitioning refers to obtaining suitable subsets/parts that can be given to agents while *knowledge allocation* refers to actually allotting these subsets to individual agents. Either or both partitioning and allocation may be done based on the agent's capacity (memory size, cpu speed, etc.), its distance from other agents, and problem specific factors, if any. Knowledge partitioning can also be referred to as *knowledge decomposition* or *knowledge grouping.* Similarly, the terms *assignment* and *allocation* are synonymous.

Some of the other terms used in this context are *knowledge organization*, and *indexing* [114]. In the Contract net frame work proposed by Smith [114], *knowledge organization* component comprises of *partitioning, indexing* and *distribution.* While the term *partitioning* is used in the same sense as ours, i.e., the way in which knowledge is broken up into modules, *indexing* is the provision of handles placed on the knowledge modules for fast access. The term *distribution* is used here to refer to *allocation.*

2.2.2 Types of Knowledge Distribution

Knowledge distribution depends on several factors like organization of the agents, distance between them, distribution of tasks and data, use of global memory or local memories, and the extent of duplication of data or knowledge allowed.

The communication required for distributed reasoning in turn depends on how data and knowledge are distributed, viz., whether statically or dynamically, with or without duplication, and the criteria used for distribution. Considering the model of agents described in chapter 1, the possible types of knowledge distribution are explained below.

1) Full (static) duplication of knowledge

A copy of the whole knowledge base is kept with each agent. Any subproblem can be allotted to any agent as its knowledge is sufficient to solve it. Therefore, communication for knowledge importation is nil if the subproblems are independent. Depending on the degree of overlap and the availability of the required data, there may be requests for data and partial results. The distribution of knowledge is static and the issue of dynamic distribution with full duplication does not arise. Distributed Hearsay [40] is an example of this.

2) Static distribution without duplication of knowledge

Knowledge is partitioned and allocated to agents statically. Usually, the given problem is also partitioned to suit the knowledge distribution and the subproblems are allocated to agents possessing the most suitable knowledge (expertise) subset. The average number of requests for partial results (measured over a number of problems solved) could be high if knowledge is distributed arbitrarily. Alternatively, the partially solved subproblem itself, with its temporary solution (partial result) can be sent to the agent with the required knowledge. For example, in Distributed Knowledge Model [86], knowledge is distributed statically, without any overlap. There is no sharing of knowledge and inference is distributed to the agents with the required knowledge. Static distribution is useful for efficient reasoning and statically balancing the load on the agents. If desired, an *a priori* estimate of dynamic aspects like the frequency of use of a knowledge base subset can be used in the static distribution[127].

3) Static distribution with partial duplication of knowledge

Knowledge is distributed statically, as in case (2), but common knowledge is duplicated at appropriate places. The most frequently used data is also kept in memory reserved for global or shared information, or duplicated and kept with each agent. However, data updation overheads and inconsistency problems will be more if data is distributed. Medical knowledge [55] for diagnosis can **be** organized in this way.

4) Dynamic knowledge distribution

The given problem is divided into subproblems and then the chunks of knowledge, probably complete with respect to the subproblem being solved, are dynamically allotted to agents. Reasonable amount of duplication of knowledge is expected as these chunks can have overlapping portions. Communication for knowledge importation is minimal. Both knowledge decomposition and allocation are dynamic. As in case(l), requests for partial results may be required to a small extent. However, dynamic distribution is costly because of the computation and communication required from time to time. Contract net [115] considers such problem based dynamic distribution of knowledge if the contractor for a subproblem does not possess the required knowledge already.

Redistribution of knowledge at run-time is necessary for dynamic load balancing also. In this case, however, knowledge need not be duplicated.

5) Semi-dynamic distribution with partial duplication of knowledge

After initial distribution of knowledge, knowledge base of an agent may undergo changes from time to time. New knowledge may be acquired as a result of learning. Knowledge which is being frequently requested from another agent may be added to an agent's local knowledge base (subset). Conversely, knowledge which is not being used by it may be deleted and transferred to the appropriate agent. When this process continues over a period of time, a stable state (with few requests for information exchange) that closely resembles case (3) can be reached. Otherwise, small changes to the initial distribution for dynamic load balancing result in semidynamic distribution resembling case (4).

6) Completely random distribution of knowledge

Knowledge may be distributed without considering the extent of duplication or relevance to the subproblem. Distribution could be done statically or dynamically, and with or without any regard to the suitability of agents. Problem decomposition and allocation may also be random. Dynamic changes to the local knowledge base may or may not take place. As it leads to total chaos with requests for knowledge

importation and information exchange, more on an average than in any other case, this may be adopted only in the initial stages of problem solving and in a very uncertain environment.

We can see from the above discussion that, in Cooperative Problem Solving, it is difficult to come up with a distribution in which both the information exchange and duplication of data or knowledge are eliminated. However, duplication of data should be kept at a low level for minimizing the associated concurrency and inconsistency problems. It is desirable to keep the duplication of knowledge also to a minimum for advantages like faster access (pattern matching time), less communication, a good task partitioning, and load balancing as mentioned in the beginning of this section and in chapter 1. Therefore, we consider a distribution in which knowledge is not duplicated unless functional decomposition is the criterion and data duplication is kept at a minimum level.

We discuss the cases (2) and (3) further in chapter 3, and cases (4) and (5) in chapter 5. Obviously, there is no need for explicit techniques for cases (1) and (6). A strategy for information exchange is the subject of chapter 6.

2.3 Partitioning/Allocation Techniques

Use of decomposition as a line of attack against complexity, and for advantages like performance, maintainability and understandability is ubiquitous in systems design and implementation. Though the literature is sparse, recognition of decomposition as a valuable concept in AI is long-standing. According to Stefik and Conway [11, 118], the idea was first quantified with Minsky's planning islands which when strategically placed are intended to reduce search in combinatorial problems.

Associated with partitioning are the cost of partitioning, problems of how to partition, what degree of partitioning is best and how to measure the effectiveness of the partitioning process. Some of the criteria for partitioning are maximum independence, minimum interface complexity, minimum subsystem complexity, minimum overall complexity and maximum comprehensibility [103].

Information-theoretic approach, cluster analysis, probabilistic and graph based

approaches are the commonly used partitioning and allocation techniques. Simulated Annealing is a more recent technique.

2.3.1 Information-theoretic and Probabilistic Approaches

Information-theoretic approach aims at minimizing coupling and maximizing cohesion [11] by considering

- local and global information flow among system components through procedures and global data structures respectively [59],
- cost, benefit, relative complexity of partitions at program specification level to guide decomposition [24], and
- common coupling, content coupling, and information paths to measure the strength of connections for testing modifiability of partitions [95].

Probabilistic approaches are used for both static and dynamic partitioning and allocation. Myers [95] describes first order and complete dependency matrices based on high probability paths to calculate the expected number of subsets that must be changed when some subset undergoes a change.

In the context of dynamic load balancing, Evans and Butt [42] indicate that service and arrival rates result in poor load balancing for long update interval lengths whereas queue lengths provide better estimates for load balancing and give an improved load balancing performance for a wide range of interval lengths. Shin and Chang [17, 110] use probability and queueing theories for dynamic load sharing in a distributed real-time system.

However, determination of exact probabilities is usually difficult.

2.3.2 Clustering

A clustering problem is simply that of separating or partitioning a finite collection of objects into subsets so as to satisfy some criteria [117]. Clustering algorithms are broadly classified as *exclusive* and *nonexclusive* (overlapping) schemes. Exclusive

technidues are further divided into extrinsic (supervised) and intrinsic(unsupervised) types. Intrinsic techniques are in turn divided into *hierarchical* and *partitional* types.

In order to use a clustering algorithm, a measure of *distance* or *relatedness* between the given objects must be defined. Measuring the similarities between all pairs of objects, closest pair is selected and merged to form one cluster. The procedure is repeated grouping objects with other objects or possibly with already formed clusters. A stopping criterion is necessary to halt the process when optimal number of clusters are obtained.

A drawback to algorithms of this type is that on each iteration, the algorithm makes the best possible agglomeration of two groups, but it never backtracks even when a better grouping is possible [72].

2.3.3 Graph Partitioning

In this approach, the entire knowledge base is first represented as a graph, and then the graph is partitioned into subsets [18]. Graph partitioning requires the number of subsets, *k,* the sizes of subsets $s_1, s_2, .., s_k$ and other constraints, if any, to be given.

A procedure for obtaining an optimal partition is NP-hard. Supposing G has n nodes of size 1 to be partitioned into *k* subsets of size *p* where *kp = n,* the number of cases to be evaluated is $(1/k!)\binom{n}{p}\binom{n-p}{p}..\binom{2p}{p}\binom{p}{p}$ which yields a very large number for most values of n, k, p. For example, if n= 40, k = 4, and p = 10, this value will be greater than 10^{20} [74]. Because of such inordinate amount of computation required in direct approaches, fast heuristics are required.

A well known heuristic due to Kernighan and Lin [74] starts with an arbitrary partition A,B of the original set S (of vertices and the associated edges in the graph G). By a series of interchanges of subsets of A and B, the method tries to decrease the external cost corresponding to the weights of edges that connect the vertices in different subsets of the partition. The first pair of vertex subsets from A and B is chosen such that their interchange results in maximum gain representing the decrease in external cost. Eliminating these subsets from further consideration, the interchange process continues with a new subset pair whose interchange results in the next highest gain until no further improvement is possible.

The process is repeated for other arbitrary starting partitions (or ones obtained as above) to obtain as many locally minimum partitions as we desire. Examination of all pairs of sets for exchange, and evaluation of the costs requires time proportional to $(n^2/2)4^n((1/\pi \cdot n)^{1/2}$ for large n resulting in a complexity of $O(n^{3/2}4^n)$. fc-way partitioning can be done either by repeated bisection, or by making an initial fc-way partition and optimizing it using the generalized (KL) fc-way partitioning heuristic.

Another technique used for partitioning graphs, viz., Simulated Annealing, is discussed in the next subsection.

Dunlop et al. [33], Fiduccia [43] et al., and Bui et al. [12] proposed improvements to these basic methods.

2.3.4 Simulated Annealing

This method is first introduced by Kirkpatrick et al. [77]. It is based on statistical mechanics and is used to solve combinatorial optimization problems like physical design of computers, graph partitioning, and Travelling Salesman problem. This can actually both partition and allocate the resulting subsets onto multiple processors [127].

Simulated Annealing (SA) consists of four elements : *system configuration, cost function, generating mechanism* and *cooling policy.* The scheme begins with an arbitrary configuration and an initial value of the cost function at a certain temperature. A new configuration generated from the previous one is accepted based on Metropoli's criterion [77] which occassionaly accepts new configurations with higher cost for escaping local minima. At each temperature, the generation process is repeated to produce a sequence of configurations representing states in a Markov chain until the probability distribution of the system states approaches Boltzmann distribution. If the temperature T is set sufficiently high and is decreased slowly enough, the Boltzmann distribution tends to converge to a uniform distribution on the set of globally minimal states.

However, in any implementation of the algorithm, Markov chain is of finite length. Therefore, asymptotic convergence can only be approximated and Simulated Annealing is not guaranteed to find a global minimum with probability 1. Moreover, both quality of the final solution and speed cannot be achieved in practice

Though Genetic Algorithms [6] and Annealing Genetic approaches [87] are also used for solving combinatorial optimization problems, these too are costly and consume lot of space to store populations of solutions.

2.3.5 Knowledge Distribution Vs. Program Distribution

Though specific to software modules, program (task) partitioning is related to knowledge base partitioning. This has been extensively studied in the areas of parallel and distributed computing, and software engineering. Following are some of the methods used in task partitioning and allocation.

Stanfel's [117] partitioning method using cluster analysis involves finding the shortest path between system components by elementary graph theory and optimization techniques. For process assignment, Arora et al. [3] reduce a graph of process and processor nodes connected by edges whose weights represent the cost to be minimized. A process node connected to another process (or processor) node by an edge having maximum weight is merged with the latter node. The reduction process continues with the new graph (with fewer edges and process nodes) until no process nodes are left in the graph.

Sadayappan et al. [39, 107, 108] propose a *nearest neighbour* approach, and a *recursive clustering* approach based on clustering and KL-graph partitioning heuristics. The nearest neighbour strategy [39, 107] is found to be more effective on hypercube systems with high message startup times, especially for finite element graphs; the recursive partitioning heuristic [39, 108] is generally better on hypercubes with lower message startup times and more effective on random task graphs.

In the dynamic load sharing method of Shin and Chang [17, 110], when a node becomes fully loaded or underloaded, it broadcasts this change to a set of its neighbouring nodes called a *buddy set,* and selects the first available node from its *preferred list,* an ordered set of nodes in its buddy set. Cybenko [25] discusses a dimension exchange method for dynamic load balancing.

For distributed online scheduling of periodic tasks, Ramamritham et al. [105] propose methods in which a task is sent to a node that is randomly selected *(random*

scheduling), a node estimated to have sufficient surplus resources to complete the task before its deadline *(focussed addressing*), based on the bids received for the task from nodes in the system (*bidding*), or based on a technique that combines both bidding and focused addressing *(flexible algorithm).*

Xu and Hwang [128] propose four heuristic methods for sending a newly arrived task from a heavily loaded node to its neighbours based on the range *(location policy)* viz., *local* or *global,* and heuristic discipline *(transfer policy)* viz., *least recently migrated node* (LRM)or a *minimum load maintained node*(MLM) combinations.

Alok Choudhry et al [20] propose a heuristic for dynamic remapping of data parallel programs where the task graph is a chain of modules and only the adjacent modules interact. After assigning the first few modules whose total load approximately equals the global average load μ on a processor, a new average load *avg* considering only the unassigned modules and the rest of processors is computed. The procedure is repeated with the new average *avg* instead of μ.

However, most of these techniques assume the processors or agents to be having equal capacity and hence are for homogeneous partitioning and allocation [17, 25, 39, 66, 75, 107, 108, 127, 128]. Usually any task can be executed on any processor without depending much on the execution of other tasks. Otherwise, the interaction pattern is simple as manifested in between adjacent modules in chain-like task graphs for data parallel programs [15, 17, 75] or finite element graphs [39, 108]. However, CPS problems exhibit more complex interaction patterns because of the interdependencies among subproblems. The problems associated with knowledge (rule) base partitioning are more complicated than those encountered in task partitioning for data parallel programs and other domains due to the grain size also. The techniques used for partitioning tasks are not adequate and directly suitable for knowledge base partitioning. Therefore, more general mechanisms are required for CPS systems.

Finally, data base partitioning and allocation has similarities with knowledge base partitioning and allocation. Normalization ensuring lossless join decomposition and dependency preservation keeps database redundancy under control and maintains semantic integrity. Depending on the application, horizontal fragmentation, vertical fragmentation or a mixture of these is used to partition and distribute

data base files [27, 79]. For more complex and large databases, file allocation problem is generalized to file allocation problem, viz., for the given queries, updates, and the sites where the results have to be sent, one should determine i) the fragments to be allocated, ii) allocate these possibly redundant fragments and the operations on them to the sites of the computer network such that a cost function is minimized. MINDS [65] is an example for such system for intelligent retrieval and reallocation of documents.

However, knowledge base partitioning is more complex than data partitioning. Data partitioning deals with relatively static data representing the physical world objects. In contrast, knowledge base partitioning has to consider both static data and temporary data (intermediate or partial results). Moreover, it needs to deal with the data as well as the code that acts upon data.

Therefore, knowledge base partitioning can be considered a sort of superset of program partitioning and data partitioning, and new techniques taking care of data distribution, homogeneous partitioning and distance along with the complex interdependencies need to be developed.

2.4 Related work on Knowledge Partitioning and Allocation

To accrue the advantages of distributing knowledge as mentioned in section 2.2, a good partitioning strategy has to be selected to minimize replication and related inconsistency problems. This requires complete knowledge of the domain and a study of the dependencies among the subsets. Also, a balance must be struck between the number of subsets and the size of each subset so that total access overhead is minimum. In a rule-based system, this refers to the balancing of *process-level* and *rule-level* overheads [18] which are associated with the selection of a rule group (when too many rule group processes are involved), and the selection of a rule within that group (when too many rules are present in it) respectively. Therefore, techniques which can partition knowledge and data to meet these requirements, independent of the domain, are of great help. However, literature on such techniques is very sparse.

A brief account of the relevant work in **DAI** as well as in other related areas like AI and parallel production systems is given below.

2.4.1 Knowledge Partitioning

Broadly, knowledge base partitioning techniques can be categorized as *(bottom-up) synthesis approaches* and *(top-down) decomposition approaches.*

Synthesis approach

In the *synthesis approach*, knowledge subsets are obtained by grouping individual rules together based on some criteria. Use of common facts, distance between rules, rule context and rule spaces are examples of such criteria to *cluster* or group rules [11]. Groups of rules with a common fact are referred to as *rooted trees.* Those groups formed on the basis of distance between rules (measured as the number of different condition or action elements) are called *concepts. Rule context* is the specific circumstance under which a rule set gets activated. *Rule space* is a collection of rules pertaining to functional characteristics like phase of activity, goal and system type.

The work under synthesis approach in single agent AI systems, parallel production systems and Distributed AI systems is presented below in that order.

Single Agent systems

Jacob and Froscher [72] describe a software engineering methodology to facilitate knowledge base maintenance. After separating control variables, their method partitions rules into a collection of *rooted trees* based on *relatedness.* Relatedness is a function of the non-monotonic facts shared by rules and a weight factor given based on the type of sharing. Rule pairs with a shared fact B as in

- if A then B, if B then C,
- if A then B, if C then B, and
- if B then A, if B then C

are assigned weights of LO, 0.75 and 0.5 respectively. Rule pairs without any common fact are assigned a weight of -0.25. Starting with each rule as a separate

group, each time a pair of groups with largest relatedness is combined into a single group until either the relatedness is negative or a predetermined number of groups are obtained.

Cheng and Fu [19] also use a clustering based partitioning approach where knowledge in the form of rules is clustered into a higher level construct termed *concept*. Concepts are either *tangible* or *intermediate* ones. Tangible concepts represent manifestations and treatments that are inputs or outputs of the system. Intermediate concepts simplify the knowledge structure to accelerate reasoning procedure and to extend the applicable domain of knowledge either generated by the system or given by experts. Distance between two rules R_i and R_j is calculated based on the number of (condition or action) elements that are different between rules.

In the context of knowledge acquisition, Davis' TIERESIAS [28] assembles rules interactively based on the rules already present in the knowledge base. It constructs *rule models* which are tree structures having all rules that conclude about the same attribute at the root level. Models of rules which conclude affirmatively and negatively about the attribute appear below the root, and under each of these are models of rules that deal with specific values of the attribute. Each model has pointers to models of more general and more specific subsets of rules.

In the context of knowledge engineering (for transferring inferential knowledge from an expert), Gaines and Shaw [47] use logical-cluster-analaysis techniques for deriving and encoding inference relations from fuzzy sets which represent decision making processes of experts. An information theoretic measure of uncertainty reduction due to a hypothesized relation is used to compare hypotheses and determine the optimum tradeoff between fuzzy truth values and the probability of being correct. Gaglio et al. [46] also use fuzzy sets for encoding knowledge from multiple experts viewing the construction of the expert system as a multiperson decision process.

Niizuma's successive problem decomposition method [101] parallelizes partitioning the knowledge base with solving an abstracted version of the given problem. The abstracted problem is called the quotient problem, and is constructed through certain information about relationship between differences and means. As the intermediate subproblems (subgoals) may not be simpler than the original one, and may not be solvable, methods have been suggested to make them so. The simplest problem is the one in which only one difference exists between the initial state and

the goal. AND/OR tree type problem trees and difference equations are used for doing this.

Parallel Production Systems

In parallel production system environment, Sohn and Gaudiot [116] partition the set of patterns namely condition, action or working memory elements. They use, in decreasing order of importance, the number of attribute value pairs (AVPs) in a pattern, similarity in attribute part and similarity in value part of patterns. Patterns that have the same number of AVPs and are similar in attributes form a cluster in a two-dimensional feature space and can be trained to achieve 0(1) pattern matching time.

Basu et al. [4] partition the rulebase into clusters named *rule spaces* depending on the functional characteristics derived from the *phase* of activity, *goal* to be established and the *type* of the system. Clustering criterion of an individual rule space is defined as a triplet belonging to the cartesian product of the phase, goal and type sets. The rule spaces are also called as *rule blocks* [5]. These rule blocks store the corresponding inference flow graph and are used for fast pattern matching and parallel rule firing.

On a message passing computer, instead of partitioning production rules, Acharya and Tambe [1] partition and distribute hashtables of tokens (incoming data and newly generated data) into hashbuckets for parallel processing of the tokens destined for different hash buckets. Their system consists of a set of match processors and a control processor. The control processor performs all conflict resolutions and broadcasts one packet containing all the working memory elements to all match processors. Match processors perform all constant tests (in the premise parts of rules), hash the new tokens generated and send to the processors which own the corresponding hashbuckets. Hash function uses the node-id of the destination two-input node and the values bound to the variables being tested for equality at the destination node as key. Produced instantiations **are** sent back to the control processor and the control processor starts the match phase of a new cycle when the current cycle ends.

Distributed AI

In Distributed AI domains like speech understanding and medical diagnosis, a domain specific functional approach to decomposition of knowledge base is considered. Decomposition of overall task into various *knowledge sources* (KSs) is regarded as natural. A KS has three parts: the conditions under which it is to be activated (in terms of the conditions in the blackboard in which it is interested), kinds of changes it makes to the blackboard, and a procedural statement of the algorithm which accomplishes these changes. A KS usually deals with one or a few levels of the blackboard to apply its knowledge. These levels of abstraction (along which the blackboard is also partitioned) hold different representations of the problem space. Examples of levels in the speech problem are syntactic, lexical, phonetic and acoustic [40, 41].

Smith's contract net frame work [114] also deals with knowledge organization. They divide the organization task into partitioning, indexing and distribution (allocation in our terminology). In this framework, partitioning is made by trial and error, and handles are provided for accessing the knowledge subsets. Knowledge distribution is either static or dynamic. Dynamic distribution is necessary when an agent requests another directly for knowledge transfer, when the task of knowledge transfer is announced, or when bidder on award of the task transfers the knowledge. There are no explicit partitioning techniques mentioned.

Gomez and Chandrasekaran [55] distribute medical knowledge through a hierarchy of *concepts* which are clusters of production rules pertaining to diseases, their causes or other notions that are relevant to diagnosis. The rules under each concept are further organized into three groups : *exclusionary, confirmatory* and *recommendation* rules. While the first two refer to rules that establish, postulate, or rule out the possibility of a disease and hence invoke a concept, the third type of rules are applied to manifestations found by confirmatory rules and anticipate subconcepts or superconcepts to suggest the possibility of related diseases.

Their more recent use of decomposition involves hierarchies of simple and complex knowledge structures to model generic categories of expert reasoning processes [16]. These generic tasks involve hierarchical classification, hypothesis matching,

knowledge-directed information processing, abductive assembly of hypotheses, object synthesis by plan selection and refinement, and state abstraction. While some are specific to diagnostic reasoning, all six can be considered as building blocks for the construction of knowledge based systems for other types of problem solvers, eg., routine design.

Lenat [81] organizes knowledge as a community of interacting modules called *Beings,* to simulate a particular expert in some domain. A Beings module possesses a corpus of specific facts and strategies for its designated speciality and can recognize when it is relevant. In the domain of automatic programming, as a result of its interaction with other Beings, final code of a Being reflects the knowledge of the expert member it is representing. This is similar to a *concept formation.*

In Distributed Knowledge Model(DKM) [86], agents are organized as a hierarchy with possible lateral connections among agents in different subtrees. Their knowledge in the form of Prolog predicates is classified as *local*, *group* and *global*. Knowledge is distributed, not duplicated, and not shared among agents. Inference is distributed to agents with required knowledge. However, method of partitioning knowledge is not discussed.

Adler [2] proposes a framework, called OMNI, for integrating existing, heterogeneous, knowledge-based systems that are deployed in a distributed computing environment. It integrates the otherwise disjoint, problem-solving components without altering them before incorporating into the framework. Integration is achieved by associating a *broker* with each existing problem-solving component, called the *specialist*

Weihmayer et al. [121] discuss about the issues in cooperation when agents with dissimilar domain knowledge and knowledge representation schemes are involved. Knowledge sources are partitioned into a two-level agent structure: local expertise/planning level and metaknowledge/agent control.

Intelligent Agent (IA) of Pan and Tenenbaum [102] supports a clearly discernible task or job function, automating what it can and calling on the services of other IAs when necessary. Complex enterprise operations are divided into a collection of elementary tasks or activities, which, after modelling in cognitive terms, are entrusted to IAs for execution. One of their goals is to integrate independently developed

software packages into the framework so that they inter-operate seamlessly and are easily used and maintained. I As interact directly via a message bus or through a distributed, shared knowledge base called MKS. MKS serves as a repository for shared knowledge and a center for information exchange among agents. This common knowledge is represented once and shared by all applications that need it.

These approaches, however, have no provision for balancing the process-level overhead and rule-level overheads.

Decomposition Approaches

In the *decomposition approach*, a partition is obtained by considering the entire knowledge base as one unit and splitting it into smaller ones. Usually, this corresponds to a graph partitioning problem where nodes of a graph G with costs on its nodes and edges, are partitioned into *k* subsets of specified sizes $s_1, s_2, \ldots s_k$ so as to minimize the total cost of the edges cut. Minimizing the total cost of edges cut when the graph is partitioned into *k* subsets corresponds to minimizing the message flow between these *k* subsets, thereby reducing the process-level overhead. However, determination of the best $k, s_1, s_2, \ldots, s_k$ is important to balance both process-level overhead and rule-level overheads [18]. However, as mentioned earlier in section 2.3, optimal partitioning is an NP-complete problem.

Harvey et al. [58] differentiate task-level parallelism from match-level parallelism [70, 116]. Task-level parallelism can be obtained by a high-level decomposition of the production system along three dimensions, viz., implicit vs. explicit, synchronous vs. asynchronous production firing, and distribution of production rules and working memory elements vs. no distribution at all. It is suggested that speed-ups obtained from task-level parallelism multiply with speed-ups obtained from match level parallelism. However, the choice of which to partition depends on whether one can readily identify a partitioning and whether its subsets have enough uniformity in their processing times to achieve a good parallelism.

Chen et al. [18] use KL graph partitioning heuristic as a subroutine to obtain *k*-way balanced partition for uniprocessor environment. In the graph for the knowledge base, each node corresponds to a rule with its cost proportional to the size of the rule. Each edge going from rule R_i to rule *Rj* is assigned a cost of one if R_j uses a

fact generated by rule i, or zero otherwise. Varying *k*, these parts are evaluated to determine the best partition using a performance equation derived from a Markov model of an event driven, message passing real-time expert system.

2.4.2 Knowledge Allocation

Knowledge allocation approaches can be classified into *one phase approaches* and *two phase approaches.* In one phase approach, knowledge allocation subsumes knowledge partitioning, whereas in a two phase approach, explicit knowledge base partitioning precedes knowledge allocation. Both these approaches have relative merits and demerits.

All the work related to knowledge decomposition discussed above comes under *two phase approach.*

Under the *one phase approach,* Dixit and Moldovan [32] formulate the problem of allocation of production systems onto a multiprocessor as a 0-1 linear-quadratic programming problem and reduce it to a 0-1 linear programming problem. They present techniques to detect parallelism and communication requirements among rules. A heuristic is proposed to solve the allocation problem using the A^* algorithm. However, their method introduces an overhead, which increases rapidly with the number of rules in the system [127].

Xu and Hwang propose a method that achieves a nearly optimal solution with reduced overhead using Simulated Annealing [127] for a balanced processing of rule-based expert systems on multicomputers (could be DAI systems also). Their main purpose is to maximize parallelism by distributing the work load evenly and to minimize communication cost in message passing among nodes. Starting with a random configuration in which all processors get almost equal number of rules, new configurations are generated by making trial changes to the previous ones. The cost function comprises of loss of parallelism, load imbalance and internode communication and is optimized to reach a global minimum.

Ishida et al. [51, 71] reorganize a collection of problem solvers to track changes in response requirements, problem solving requests and resource requirements. Agents are created and destroyed dynamically, and domain knowledge is continually reallocated using *decomposition* and *composition.* Decomposition divides an agent into

two, and composition combines two agents into one. These force reorganize actions by modifying the distribution of *problem solving knowledge* and *organizational knowledge* in the organization, and modifying the particular association between resources and problem solving knowledge. The decomposition and composition are performed by arbitrarily halfing the rules of an agent (to create two agents), and clubbing two agents (to make one agent) respectively. However, the need for better partitioning techniques has been emphasized.

Tout and Evans [119] propose a model in which production rules are distributed at run-time. Rules in a processor do not migrate to other processors during execution. If the queue of tasks to be investigated is not empty, each idle processor requests and gets a predefined number of rules, and investigates to find the applicable rules. From among the applicable rules, some or all the rules may be selected and fired resulting in updates to the database which further initiate a new cycle and define a new task queue. Their performance analysis indicates that the design of parallel expert systems with local working memories improves speedup as well as efficiency and the results are slightly better when the size of the rulebase is larger.

2.5 Conclusions

Many distributed AI domains like Hearsay II adopt a general functional decomposition strategy which is domain dependent. Moreover, efficiency issues like load balancing and faster processing are usually not considered in these systems.

The work of Chen et al. [18]and Xu et al. [127] for static partitioning of rulebases aims at a balanced fc-way partitioning in which all subsets are of equal size. We refer to this as *homogeneous partitioning.* The dynamic scheduling of rules in equal numbers to processors by Tout et al. [119] again refers to homogeneous partitioning. Similarly, the adaptive decomposition and composition by Ishida et al. [71] can be considered as homogeneous partitioning.

Homogeneous partitioning is applicable to systems in which all agents have the same capacity. However, if agents have different computing capabilities, homogeneous partitioning necessarily limits performance. Since cooperating agents need to exchange results, a faster agent may have to wait till a slow communicating partner finishes. Also, there could be probably a better partition with different component

sizes that results in less information exchange. This problem can be solved by *heterogeneous partitioning* where a task is divided according to performance capabilities of individual agents.

Further, as both KL [74] and SA [77] are computationally intensive, a sub-optimal solution obtained quickly can be of great use if optimality is not that important and cpu time is at a premium. A good starting partition reduces the time taken in both these methods for obtaining the final solution or improves its quality. Also, there is no explicit support for distributed reasoning to resolve incompleteness of local information in both the algorithms.

In the next chapter, techniques for obtaining fc-way heterogeneous partitions statically and allocating the resulting subsets are presented. The partition so obtained can also be used as good starting partition for the KL and SA techniques. Metaknowledge is abstracted by the technique so that distributed inferencing can be effectively performed.

Dynamic distribution of knowledge is considered in Chapter 5.

Chapter 3

Static Knowledge Base Partitioning and Allocation

Our objective in this chapter is to develop a graph-based heuristic for a fc-way heterogeneous partitioning of the knowledge base which overcomes the problems mentioned in the last chapter by using data dependencies. We follow the decomposition approach to obtain a given number of subsets in the specified proportion of sizes. In addition, methods for obtaining functional decomposition and allocating the resulting subsets are also discussed and developed.

The chapter is organized as follows. Section 3.1 gives an introduction and section 3.2 explains the issues involved in cutset based knowledge base partitioning. Section 3.3 discusses the k-way partitioning of a connected knowledge graph. Section 3.3.6 gives the linear time heuristic for static partitioning, analyzes its time complexity and section 3.4 illustrates the heuristic with a few examples. Sections 3.5 and 3.6 deal with disconnected components in the graph, and functional decomposition respectively. Section 3.7 presents a heuristic for allocating the rulebase subsets obtained as above to k agents with a given topology and interagent distances. Finally, the last section presents the conclusions.

3.1 Introduction

Earlier work on static rulebase partitioning for load balancing used techniques like KL graph partitioning [18] and Simulated Annealing [127] for obtaining a homogeneous partition. Considering graph-based approaches, literature on graph partitioning usually refers to vertex partitioning, particularly two-way partitioning or bisection [12, 43, 74]. The techniques are originally developed for VLSI circuit design and even the improvements are oriented towards the same. If more than two subsets, say k subsets are required, it is done by repeated bisection of the graph.

There is no direct procedure to perform fc-way heterogeneous partitioning. However, in distributed (AI) systems, agent capacity and speeds may not be equal. If all the agents are allotted subsets of the same size, the slowest agent will be a bottleneck for the entire network [15].

Further, earlier work on rule partitioning for load balancing considers a single processor or multiple processors with shared memory, as only rules were to be distributed [18, 127]. Data is either centralized or fully duplicated in local memories. But when data is also to be distributed and kept in the local memories of agents which are possibly geographically separated, data updates and inconsistencies associated with rapidly changing data (eg. in a monitoring system) become important. Often the techniques used in other domains ignore data distribution, resulting in run time communication delays while accessing data at remote sites [111]. To remedy these problems, it is necessary to distribute both rules and required data.

We aim at obtaining a heterogeneous partition without repeated bisection such that both rules and data are partitioned using a graph-theoretic approach [98, 99]. Data are represented as vertices and rules are represented as labels on the edges connecting data. As rules are a kind of semantic constraints and can be treated as functional or multivalued dependencies [27, 79], this representation allows us to exploit the dependencies and the adjacency of data and rules in the graph. Our objective then becomes that of partitioning both vertices and edges such that rules are in the given proportion with less communication overhead and data duplication. Further, as part of the partitioning process itself, metaknowledge directories are generated to facilitate distributed reasoning.

For developing formal methods covering various cases of static partitioning and allocation, we shall first give few definitions and discuss how cutsets can be used to obtain partitions from the knowledge graph representing the rulebase.

3.2 Cutset Based Knowledge Base Partitioning

We shall use the following example rulebase with six rules to explain the possibilities and issues in partitioning the knowledge base using cutsets.

R1.	AB → C	R2.	BDE→ F
R3.	BG → H	R4.	I → J
R5.	FG → K	R6.	AG → I

The left hand side (LHS) of each rule is called the premise part, and the right hand side (RHS) the action part. For simplicity, we have shown only attribute names and boolean *and* relationship among the attributes in all rules. However, there is no loss of generality because, irrespective of the values of the operands (even if they are not boolean) and the relationship between them, the expressions must be evaluated to proceed with rule enabling and firing. Therefore, it is enough for our purposes to know in which rules an attribute participates in the RHS part and in which rules it participates in the LHS part.

Def 1.1 Knowledge Graph

A knowledge graph G — (V,E,L) is a directed, acyclic, labelled graph which consists of a set V of data elements (attributes or objects) as vertices, a set E of edges such that E : V → V, and a set L of edge labels corresponding to the rule identifiers to which the edge belongs.

To construct a knowledge graph, a directed arc is drawn from each of the attributes in the LHS part of a rule to (each of) the attributes in its RHS part. The rule id is given as the corresponding edge label. The knowledge graph for the above rulebase is shown in figure 3.1.

In an acyclic graph, the *indegree* of a vertex v_i is the number of incoming edges incident on v_i. Similarly, the *outdegree* of a vertex v_i in an acyclic graph is the number of outgoing edges incident on v_i. The *degree* or *incidence* of v_i is the number of edges incident on the vertex. Table 3.1 gives the *indegree, outdegree* and *incidence* pertaining to each attribute in the knowledge graph.

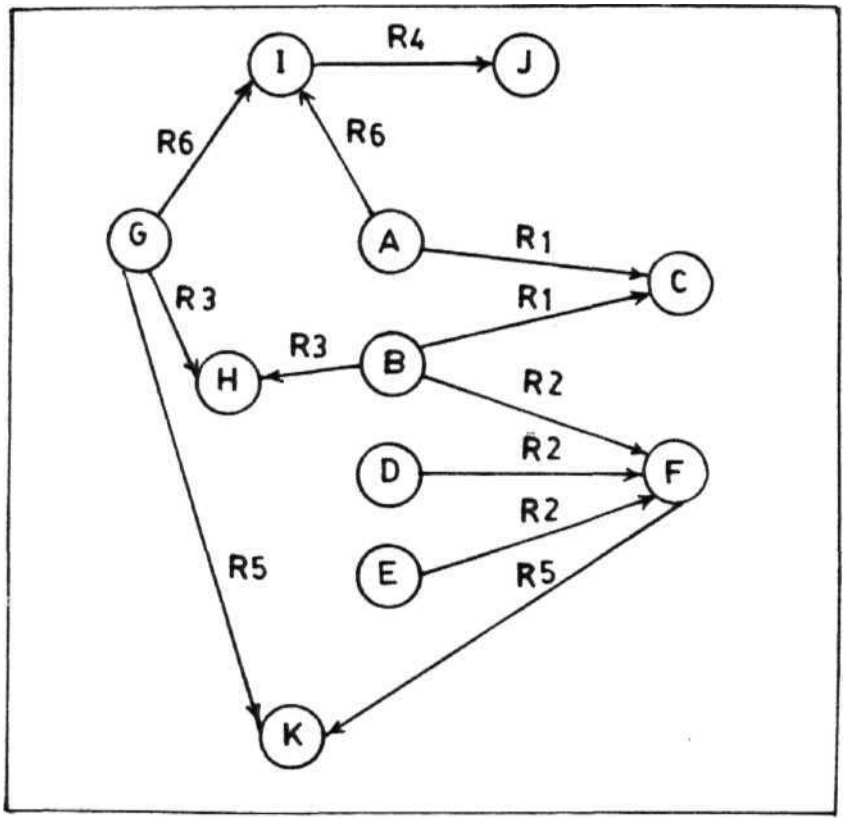

Figure 3.1: Knowledge Graph for Example Rulebase 1

	A	B	C	D	E	F	G	H	I	J	k
Indegree	0	0	2	0	0	3	0	2	2	1	2
Outdegree	2	3	0	1	1	1	3	0	1	0	0
Degree	2	3	2	1	1	4	3	2	3	1	2

Table 3.1: Degree information for the Knowledge Graph for Example Rulebase 1

Def 1.2 Cutset

In a connected graph G, a cutset is a set of edges whose removal from G leaves G disconnected, such that no proper subset of these edges disconnects G.

Since a cutset is the minimum set of edges whose removal leaves a connected graph disconncted, it is also called as *minimal cutset.*

For instance, the set of edges $\{GI, BC\}$ forms a cutset and divides the vertices into two sets {I, J, A, C} and {B, D, E, F, G, H, K} shown in figure 3.2a. The only rule that is completely associated with the first subgraph is R_4; the rules associated with the second subgraph are $R_2, R_3,$ and R_5. R_1 does not belong to either of them completely as neither has all the data required by it. Similarly R_6 also does not appear in either of the subgraphs.

Def 1.3 Rule Completeness

If $R = \{R_1, R_2, ..R_n\}$ *is the set of rules associated with the knowledge graph G corresponding to the complete knowledge base , then the rule subsets* RS_i *associated with each of the subgraphs* G_i *must ensure that* $\backslash R \backslash = |\cup_{i=1}^{k} RS_i|$, *where* $\backslash R \backslash$ *stands for the cardinality of the set R.*

In order to achieve *rule completeness* in the above example, we need to duplicate attributes B and G with the first component also. Now, as shown in figure 3.2b, the resulting subgraphs have their component data sets as {I,J,A,B,C,G} and {B,D,E,F,G,H,K} and rules sets $\{R_1, R_4 R_6\}$ and $\{R_2, R_3, R_5\}$ completely associated with them. This duplication is similar to the duplication of attributes in files for database *referential integrity*, and *dependency preservation* [27, 79). It indicates the need for keeping copies of data in working memories of agents.

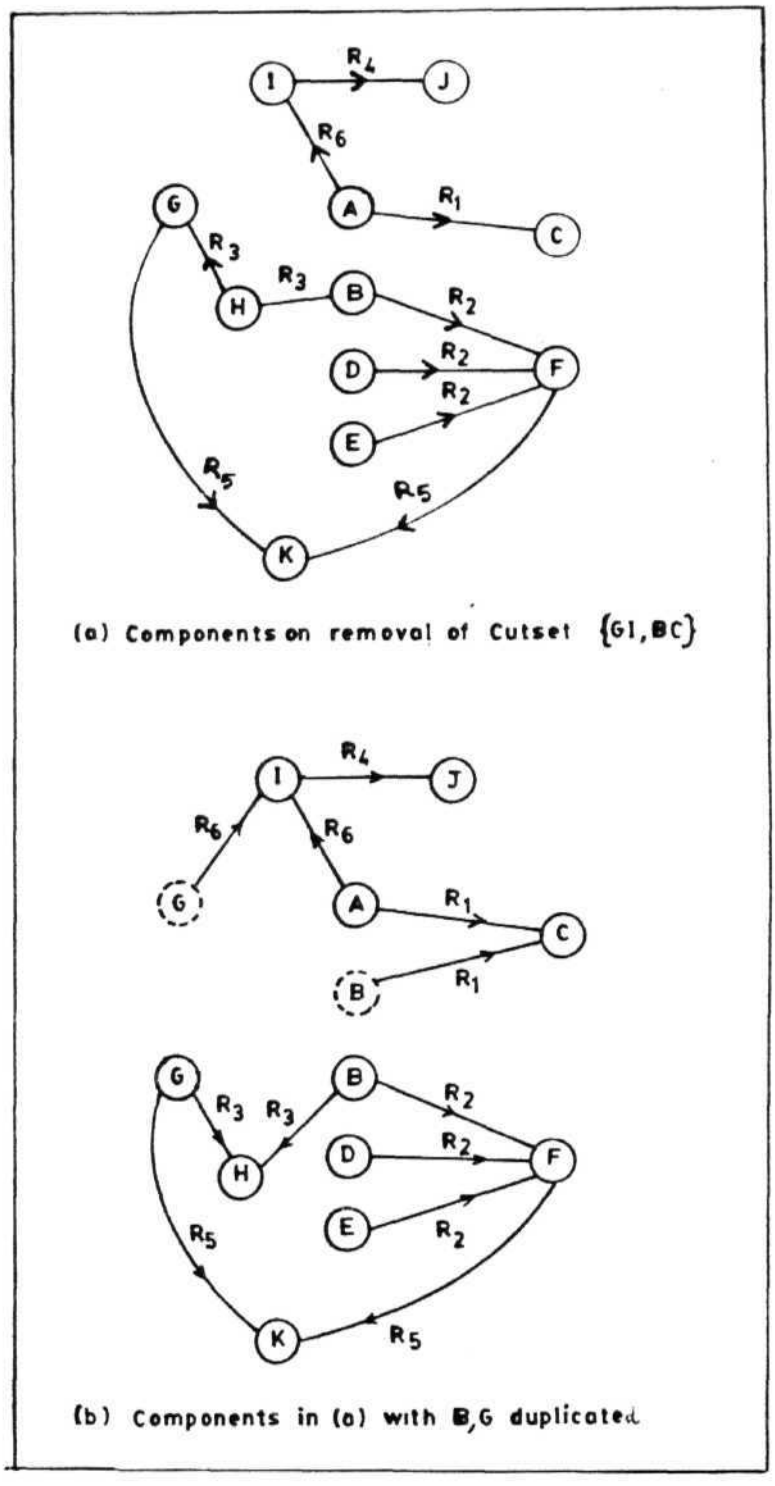

Figure 3.2: Components obtained with cutset {GI,BC} for Example Rulebase 1

However, for avoiding the data inconsistency problems, if rules are to be assigned to subsets without duplicating data, some more constraints need to be considered. In this case, after the rule is assigned to some subset, the corresponding agent has to receive the required data from other agents for enabling and firing that rule. Selection of a suitable subset (and hence an agent) for the assignment of a rule is discussed in the next section.

We choose the later option and, hereafter, instead of showing the duplication of attributes in the graph, the information regarding the required data will be stored in the directories (after assigning the rule to a suitable subset). As we shall see in the forthcoming chapters, this metaknowledge is useful to dynamically redistribute the knowledge and to facilitate distributed reasoning.

Mathematically, computation of a cutset is closely associated with a spanning tree.

Def 1.4 Spanning Tree

A tree T is said to be a spanning tree of a connected graph G if T is a subgraph of G and T contains all vertices of G. (It is also called a skeleton, scaffolding, or maximal tree (subgraph) of G.)

A spanning tree has n — 1 edges where *n* is the number of vertices in the original graph. Removal of any edge from a spanning tree separates it into two disconnected parts. The set of solid arcs in figure 3.3 is an example for a spanning tree. The dashed arcs are those which are present in the original graph but not in the spanning tree and are called as *chords* of that spanning tree. (The meaning and use of integer markings given on edges will be explained in the next section.)

It is quite possible that a knowledge graph is not connected and has components in it. However, we postpone the discussion of disconnected components in the graph till section 3.5 and assume that the knowledge graph is connected till then.

A cutset can be obtained by removing an edge belonging to the spanning tree giving two disjoint vertex (node) sets, and selecting the chords, if any, which connect the vertices in the two disjoint vertex sets. The spanning tree edge and the chords together form the cutset. For example, removal of edge BC from the spanning tree

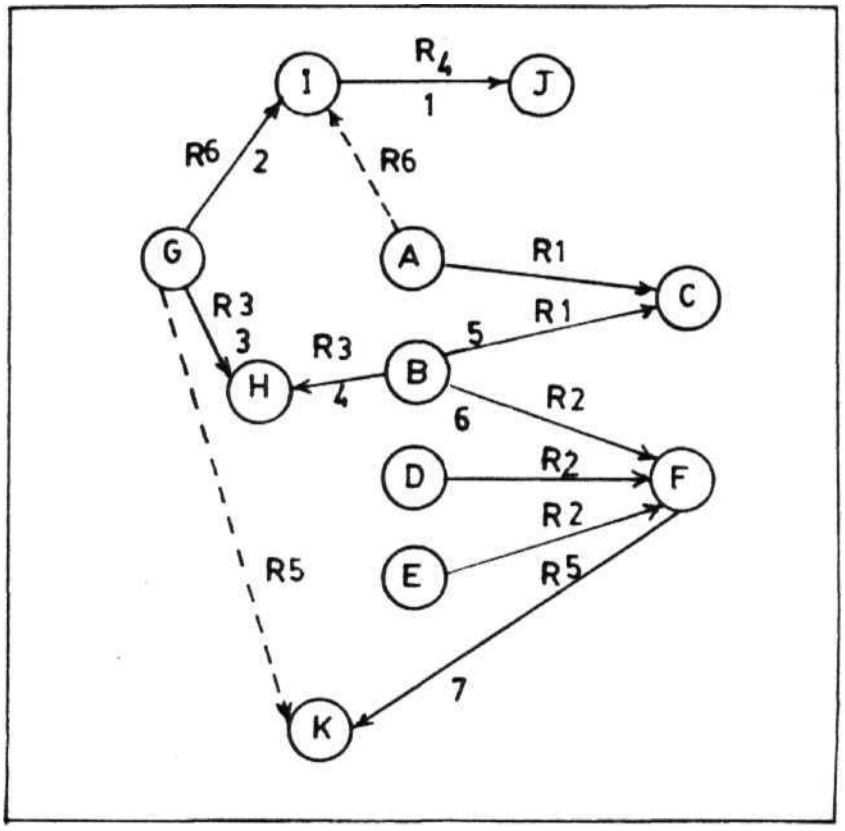

Figure 3.3: A Spanning Tree for the Knowledge Graph of Example Rulebase

shown in figure 3.3 yields two vertex sets {A,C}, and {B,D,E,F,G,H,I,J,K}. To obtain disconnected components with the same partitioning of the vertices, chord AI also must be removed. Therefore, the set of edges AI and BC forms a cutset. This is also called a *fundamental cutset.*

Def 1.5 Fundamental Cutset

A cutset S containing exactly one brunch of a spanning tree T is called a fundamental cutset with respect to T.

Though the number of subsets (parts) obtained as above using cutsets is two, this method of obtaining cutsets [31] does not bear any relationship with the number of rules required in each subset. Moreover, obtaining *k* subsets in the required ratio is not straight forward. In order to develop such a method for knowledge graph partitioning (in the next section), we shall first define the terms semipath, length of a semipath, pendant vertex, spandegree and chain.

Def 1.6 Semipath

A semipath from vertex v_i to vertex v_j is a path from v_i to v_j in the corresponding undirected (knowledge) graph.

In the knowledge graph shown in figure 3.3, JIG is a semipath from J to G. JIGHBFK is another semipath, i.e, from J to K.

Def 1.7 **Length** of **a** semipath

Length of a semipath is the sum of weights of edges incident in the semipath.

We assume unit weights for edges unless otherwise explicitly mentioned. Length of the semipath JIGHBFK is 6.

Def 1.8 **Pendant vertex**

A vertex with degree 1 is a pendant vertex.

J is a pendant vertex in the above graph.

Def 1.9 Spandegree

Spandegree of a vertex w.r.t. a spanning tree is the degree of a vertex considering the edges in that spanning tree only.

Def 1.10 Chain

A Chain is the longest semipath in the spanning tree.

The semipath JIGHBFK is the chain for the spanning tree shown in the figure 3.3.

Now we shall classify the spanning tree edges into two types: those which lie on the chain, and those which do not. We call the former type of edges the *chain edges*, and the latter type of edges the *branch edges.* A branch emanates (in the corresponding undirected graph) from a vertex whose spandegree is greater than 2. Semipath BCA is a branch on the chain JIGHBFK.

A branch in turn may have subbranches in it. The number of branches of a branching vetex is called its *branching factor.* Branching factor of vertex B is 1. For a branching vertex lying on the chain, branching factor is 2 less than its spandegree. For example, if a vertex has a spandegree of 4, then it has two branches.

3.3 *k*-way Partitioning of the Knowledge Graph

The proposed *k*-way partitioning consists of two phases: *initial decomposition* and *boundary refinement.* In the first phase, knowledge graph is initially cut into *k* disjoint components by cutting the graph at *k* - 1 places using spanning trees and cutsets. This involves determining the position of the cut and an approximate partitioning of the rules and data. In the second phase, boundaries are smoothened

to get a required partitioning leaving the inner portions of the subsets undisturbed as far as possible.

To enable partitioning, initial decomposition phase requires

- the generation of a spanning tree from the knowledge graph (constructed as described in section 3.2) and
- marking of its edges suitably.

The same spanning tree, chain, and edge markings can be used to obtain partitions in any desired proportion. However, partitions obtained using different spanning trees may be of different sizes and exhibit different characteristics. Therefore, for obtaining a particular partition, selection of the spanning tree and determination of the cutsets are non-trivial. Since, exhaustive enumeration is not practical, we propose a heuristic for this.

3.3.1 Generation of a Spanning tree

Since a spanning tree encompasses all the vertices, it has a large number of rules on its edges. A spanning tree with a long chain facilitates partitioning due to its linearity. Therefore, we consider generating a spanning tree with a long chain.

Selection of starting vertex

We choose either a pendant vertex or a vertex with zero indegree or zero outdegree as a starting vertex, to generate a spanning tree with a long chain. We shall call this the *root* of the spanning tree.

There will be at least one candidate root in any knowledge graph as all external input data items will have zero indegree, and all final result attributes will have zero outdegree. A vertex with degree 1 may be preferred as candidate root because this will not have another edge incident on it, and hence it is quite possible that this may be forming one end of the chain.

Inclusion of edges

Since every spanning tree has a chain, identifying a spanning tree with the longest chain is computationally intensive. Hence the following heuristic is adopted. Starting with the root, edges are included in the spanning tree such that spandegree of vertices is kept less than or equal to 2 as far as possible. This heuristic gives a spanning tree with a reasonably long chain. Considering the root as the current vertex, an edge incident on it and the other end vertex (adjacent vertex) are included in the spanning tree if the adjacent vertex is not already in the spanning tree vertex list. The newly added vertex now becomes the current vertex and the process is repeated with the new current vertex until n — 1 such edges covering all the vertices in the spanning tree are included.

The following procedure gives the detailed steps involved.

procedure *generate_spanning _tree()*;
(* generate a spanning tree with a long chain for a connected knowledge graph *)
begin
initialize *spandegree* of all vertices to zero;
select a vertex v_i as root;
add v_i in *open list* at the end;
add v_i in the *spanning tree vertex list*;
k := 2;
while *(k <= n)* and *(open list* not empty) do
begin
if 3 an edge v_iv_t or v_tv_i in the knowledge graph such that
v_t is **not** in the *spanning tree vertex list* already then
(* if v_t is already in the *spanning tree vertex list,* mark the edges
unsuitable for further consideration *)
include the edge in the *spanning tree edges list;*
include ruleset RS_e on the edge in the *spanning tree rule set;*

include v_t in the *spanning tree vertex list;*

$k = k + 1$;

increment *spandegree* of v_i and v_t;

if *spandegree* of v_i is equal to the degree of v_i then

delete v_i from *open list;*

endif;

if *spandegree* of v_t < degree of v_t then

add v_t in *open list;*

endif

else

delete v_i from *open list;*

endif;

if *open list* not empty then

let v_i be the (most recently added but undeleted)

vertex at the end of the *open list;*

(* else if *open list* empty and $k < n$ then there are

disconnected components in the graph

select a new v_i as root from the rest of the vertices in the knowledge graph;

mark the previous component with its rules and number of rules for indexing;

endif;*)

endif;

end; (* end of while*)

end; (* end of procedure *)

3.3.2 Finding Chain

Though a spanning tree has n-1 edges of the original graph, there need not be only one single semipath in the spanning tree. Also, the (longest) semipath from the root

to the last vertex (included during spanning tree generation) need not be the chain in the spanning tree.

Finding the chain is done in two steps:

1. finding the longest semipath from root, and
2. finding the chain using the longest semipath obtained in step 1.

The longest semipath from root may be found using depth first search starting from root. Lengths of semipaths from a vertex to leaf vertices (which are not in the longest semipath) in spanning tree are stored while backtracking in the search process in order to find the chain in step 2.

The spanning tree can now be divided into three parts, viz., the longest semipath portion sp_1 from root to the first branching vertex, portion *mp* from the first branching vertex to the last branching vertex including all the branches incident on each branching vertex, and the portion of (the same) longest semipath sp_2 from the last branching vertex to the last vertex in the longest semipath. Each branching vertex will have first and second longest branches bp_1 and bp_2 if its branching factor is at least two. Let l_1, l_2, b_1 and b_2 be the lengths of sp_1, sp_2, bp_1 and bp_2 respectively. Now, the longest semipath in the entire spanning tree, i.e., chain, may be found as follows:

If the longest semipath portion till the branching vertex under consideration sp_1 is shorter than its first longest branch, the branch becomes part of the chain (longest semipath), and vice versa. If the new branch is shorter than the second longest branch, then these two are interchanged. Adjoining the portion of the semipath from this branching vertex to the next branching vertex, if any, with sp_1, the process is repeated for all the branching vertices. The new sp_1 adjoined with sp_2 will be the longest semipath in the entire spanning tree and hence the chain.

The procedure for finding the chain using the longest semipath from the root is given below.

branch point bv_1 and the next branch point *bv* to l_1;

make *bv* the new bv_1;

endif;

until the last branch point bv_2 is encountered;

chain is the semipath *sp*\ adjoined with sp_2;

length of the chain $l_c = l_1 + l_2$;

let RS_c be the set of rules present on the chain edges;

end; (* of the procedure *)

We have already mentioned that many of the rules in rulebase appear on the edges of the spanning tree. Even if there are any rules which do not belong to the spanning tree rule set, this does not affect the final decomposition and the rule completeness. These rules will be included when chords are considered in the cutset — chain is used only to facilitate decomposition.

To determine the actual cutsets, the spanning tree edge to be removed first must be found out for each cutset. In order to identify these, spanning tree edges need to be marked suitably.

3.3.3 Marking edges for decomposition

Integer markings on edges indicating their position in the spanning tree are useful in determining the positions where the graph is to be cut to obtain an approximate partitioning in the given proportion. We give below the steps for marking (labelling) the edges to enable suitable decomposition.

procedure *mark_edges()*;
begin
Mark the edges with 1,2,... to indicate their position
from one end of the chain to the first branching vertex on the chain;
Let the rules corresponding to edges in the chain be denoted by RS_c;
repeat
If a branching vertex is encountered then
for all branches of the branching vertex
mark a branch edge with the next integer;
include RS_e in RS_c
only if the rule labels RS_e on it do not belong to RS_c;
endfor;
endif;
continue marking chain edges with the next integer without
looking for the corresponding rule label
till another branching vertex is encountered
until the other end of the chain is reached;
Let / be the highest label (i.e., number of edges labelled in this way);
end;

The value of l indicates the number of marked edges in the spanning tree. It can be seen that the set of rules on the marked edges is nothing but the spanning tree rule set. The value of / can be used now to determine the position of the edges to be cut and thus find the subsets.

The vertices and the edges numbered this way are stored in an array of size n to facilitate decomposition. Starting from the edge marked 1, all edges including the branch edges without any markings, and the associated vertices along with the integer marking information in increasing order are stored in the spanning tree. For the spanning tree and its chain shown in figure 3.3, the linear representation would be the set of edges {JI, IG, GH, HB, BC, CA, BF, FD, FE, FK}, with the associated

vertex ordering {J,I,G,H,B,C,A,F,D,E,K}.

3.3.4 Initial Decomposition

For the initial decomposition, we must compute the size of each subset, determine whether balanced partitiong, i.e., partitioning exactly in the given ratio, is possible, and then determine the cutsets to be used for decomposing the graph.

Size of subset

Let N be the total number of rules, and $p_1 : p_2 : .. : p_k$ be the proportion in which rulebase subsets are to be obtained. We can obtain size of each knowledge subset *z;* as $\lceil p_i * (N/\sum_{j=1}^{k} p_j) \rceil$.

procedure *determine_size_of_subset()*;

```
begin
for i := 1 to k do
        z_i = ⌈p_i * (N/∑_{j=1}^{k} p_j)⌉;
endfor;
end;
```

Is balanced partitioning possible?

If $N mod \sum_{j=1}^{k} p_j = 0$ then, it may be possible to obtain a balanced partitioning in the given ratio. Otherwise there may be a small difference in the desired sizes calculated for the given ratio and the actual sizes of the subsets obtained. The following procedure does the same.

```
procedure check_if_balance_possible(bal_possible);

(* check if required partitioning is possible *)
begin
   if N mod ∑_{j=1}^{k} p_j = 0 then
      bal_possible = true
      else bal_possible = false;
   endif;
   return(bal_possible);
end;
```

Determining the position of the edges to be cut

If / is the number of marked (labelled) edges of the spanning tree, we can divide it approximately in the given ratio by cutting at edges $e_1, e_2, .., e_{k-1}$ where

$$e_i = \lceil (\sum_{j=1}^{i} p_j / \sum_{s=1}^{k} p_s) * l \rceil;$$

If an edge to be cut, e_i, happens to be a branch edge, and if the number of rules on the chain RS_c happens to be small compared to the total number of rules N, then e_i is taken as it is. Otherwise, the following possibilities exist depending on whether $p_i >= p_{i+1}$, or $p_i < p_{i+1}$.

- In the first case, i.e., if $p_i >= p_{i+1}$, we skip the branch, go in the forward direction, i.e., in the direction of increasing integer markings, and make e_i the first edge in the chain immediately after the branch so that rules on the branch edges are added to P_i.
- Otherwise, if $p_i < p_{i+1}$, then we make e_i the last edge in the chain just before the branch so that branch rules are added to P_{i+1}.

This is to retain the sizes of subsets (to be obtained) closer to the proportion **given.** Further, if any two edges e_i and e_{i+1} have no marked edges in between

them, then we must increment or decrement the number appropriately depending on whether $p_i >= p_{i+1}$ or not. However, this is done only if the new edge does not belong to a branch again and violate the branch criterion described above. The branch skipping keeps the resulting cutset as a fundamental cutset and can reduce information exchange between subsets.

procedure *determine_edges_to_cut()*;

(* determine the position of edges e_i to be cut to make the graph disjoint *)

begin

i) for each subset do

$e_i = \lceil (\sum_{j=1}^{i} p_j / \sum_{s=1}^{k} p_s) * l \rceil$;

endfor;

ii) if RS_c is close to N then

(* the number of rules on the chain edges is close to N *)

for i := 1 to k-1 do (* for each subset *)

if $p_i >= p_{i+1}$ then

e_i is the first edge in the chain immediately after the branch

else

e_i is the last edge in the chain just before the branch;

endif;

endfor;

endif;

iii) for i := 1 to k-1 do (* for each subset *)

if (e_{i+1} - e, - 1) = 0 then

if ($p_i >= p_{i+1}$) and (e_i + 1 does not belong to a branch) then

$e_i = e_i + 1$

else if (p, < p_{i+1}) and (e_i - 1 does not belong to a branch) then

```
                    e_i = e_i - 1;
                endif;
            endif;
        endfor;
end;
```

Once the edges in the spanning tree to be cut are identified for including them in the cutsets, we can determine the actual cutsets by including the chords (and other spanning tree edges) also. Removal of these cutset edges leaves the graph as *k* components. The data and rules which belong to the individual components form an approximate partition. Retaining the same partition of data obtained by the spanning tree decomposition, rules may be assigned to the subsets based on the data available with the subsets and the data required for the rules. We first find a *proposed* set of rules for each subset and do the actual assignment in the *boundary refinement* phase. We shall denote the data (vertex) set and the proposed rule set of a subset by VS_i and *PRSi* respectively. Steps for finding these are given below.

```
procedure find-data_and_proposed_rule_sets();
(* Form disjoint vertex sets VS_i's and *)
(* proposed rulesets PRS_i's for the k subsets *)
(* by cutting at e_is along the chain; *)
begin
   for i := 1 to k do (* for each subset do *)
      VS_i = { v / v is the second end vertex of e_(i-1) or
         v is the first end vertex of e_i or
         v is a vertex incident on edges between e_(i-1) and e_i};
         if i = 1 then e_0 is the very first edge in the chain;
         if i = k then e_k is the last edge in the chain;
      PRSi = { RS_e / RS_e is the set of rule labels on the edges between e_(i-1) and c,};
   endfor;
end;
```

The set of rules present on the cutset edges is called the Conflict Rule Set and is denoted by *CRS*. These rules correspond to the spanning tree edges as well as the chords in all the k - 1 cutsets. Since these are the likely candidates for inclusion in more than one subset, the exact subset in which a rule in CRS is to be placed has to be determined in the boundary refinement phase.

Rules corresponding to a chord (or to those edges other than the e,.s calculated for cutting) and found to be using and producing data belonging entirely to one subset p_i are included in the PRS_i of that subset; otherwise, if the end vertices of the chord (or another spanning tree edge of cutset) are in two different components, the rule is included in the *CRS*. The following procedure does this.

procedure *find-proposcd^and.cutseijrules()*

```
(* Compute the cutset rules (Conflict Rule Set) CRS, and associated subsets; *)
begin
for i := 1 to k - 1 do (* for each subset *)
   for each vertex v_j of the set VS_i do
      for each edge v_j v_t with rule set RS_e do
      (* in the corresponding undirected graph *)
         if v_t ∈ VS_m where m <> i then (* m not equal to i *)
            store RS_e and the the subset ids i and m in CRS
         else PRS_i = PRS_i ⋃ {RS_e};
         endif;
      endfor; (* edge *)
   endfor; (* vertex *)
endfor; (* subset *)
end;
```

It may be observed that (all) the cutsets formed like this are not necessarily fundamental cutsets with respect to that spanning tree. If an edge e_i belongs to a branch, it will not form a fundamental cutset. The vertices for the first subset are separated first, those for the second subset next and so on, until all vertex subsets are identified. Then the cutset edges are those which have end vertices in two different vertex sets.

3.3.5 Boundary Refinement

In this phase, we determine the subsets to which the rules in the CRS are to be assigned. Among the rules in CRS, a rule with largest number of attributes is considered first. To resolve the conflict and actually assign the rule, selection of the appropriate subset can be done by calculating the number of data elements (pertaining to this rule alone) available in each candidate subset. Let us call this its *attribute count.* Now, a rule is assigned to a subset with highest attribute count and still has not got its share of rules z_i. However, if all the candidate subsets have got their share of rules, the rule is assigned to a subset with highest attribute count. The partition obtained above ensures rule completeness.

Metaknowledge Directories

Once a CRS rule is assigned to a subset, it is necessary for the agent (to which that subset is allocated) to know what other relevant data is required from other agents (having other subsets) in order to fire that rule. Alternatively, data generated by firing this rule may be used by some other agent also. Hence, it is necessary that the agent knows which other agents require this data. For each subset P_i, we maintain two areas in the directory, viz., *NRFi* and MRB_i for this purpose.

NRF_i represents the data that Needs to be Requested From other agents. It has the details of data name and the id of the subset (agent) from which the data is to be requested (obtained). Similarly, MRB_i represents the data that May be Required By other agents. The details of data name and the id of the subset (agent) which may be requiring this data are stored in MRB_i. Assuming a rule in subset *Pi* requires some attribute v_a in subset P_j ($v_a \in VS_j$, i.e., owned by agent having subset P_j), *NRFi* has an entry (v_a, j), and *MRBj* has an entry (v_a, i).

The *NRFs* and MRBs represent the coupling between agents and thus help in the allocation of knowledge subsets to agents, redistributing knowledge dynamically and requesting for nonlocal information. This will be discussed in section 3.7 and the forthcoming chapters.

The procedure *assign_rules* assigns the rules in CRS to the subsets as described above and updates the NRF and MRB parts of each subset.

procedure *assign_rules()*;

(* Assign rules to subsets and store directory information *)
(* NRF_i:Needs to be Requested From other agents, a list (attribute, agent) *)
(* MRB_i:May be Required By other agents, a list (attribute, agent) *)
begin
i) for i := 1 to k do (* for each subset *i* *)
    nonconflicting rule set $RS_i = PRSi - CRS$;
    $NRF_i = \{\}$;
    $MRB_i = \{\}$;
    if $\backslash RSi \backslash >= z_i$ then
        mark it *okay* and add it to *okay list;*
    endif;
  endfor;
ii) sort the c rules in CRS in decreasing order of rule size,
  (* i.e., on the number of attributes in the rule *)
  for each rule in CRS do
    sort the candidate subset ids involved based on number of
      attributes of this rule (in that subset)
    for i := 1 to s-1 do (* each subset id i in the sorted list*)
      if $\backslash RSi \backslash < z_i$ then

include the rule in *RSi* (• of that subset *);

RSi\ = *RSi*\ + 1;

endif;

if *RSi*\ = z_i then

mark it *okay;*

add the subset in *okay list;*

break;

endif;

endfor;

endfor;

if the rule is not allotted to any of these subsets and bal_possible = false then

allot the rule to the first subset with an id i at the beginning of the list;

$|RS_i| = |RS_i| + 1$;

if *RSi*\ >= z_i then

mark the subset *okay* and add it in *okay list;*

endif;

endif;

let P_i be the subset to which the rule is allotted;

for each attribute v_a of the rule,

such that $v_a \in V_j$ where $j <> i$

(* P_j is a candidate subset which did not get the rule *)

$NRF_i = NRF_i \bigcup (v_a, j)$;

$MRB_j = MRB_j \bigcup (v_a, i)$;

(* update NRF of the subset to which the rule was allotted

and MRBs of the remaining candidate subsets *)

endfor;

end;

The procedure *check_if_balance_obtained()*checks whether the partitioning obtained is exactly in the given ratio. If a balanced partitioning is possible, but is not obtained with the above spanning tree edges cut, $e_1, ..e_{k-1}$,then it shifts some of them so as to shift few of the extra rules from the subsets whose sizes are greater than their desired sizes.

procedure *check_if_balance_obtained(balance_obtained)*:

```
(* Check if required partitioning is obtained *)
begin
        balance_obtained := false;
        for each subset i do
                if |RS_i| = z_i then
                        put the subset in the balanced subset list;
                endif;
        endfor;
        return(balance_obtained);
end;
```

After the final vertex sets and rule sets are found, the following procedure finds the exclusive vertex sets EVS_is and dupliate vertex sets DVS_is. While VS_i represents the data owned by the subset P_i, EVS_i represents the data that belongs to subset P_i and is used only by the rules in P_i. However, DVS_i represents all the data required by the agent to fire the ruleset RS_i. If data duplication is allowed, updates should be propagated to all the places.

The following procedure calculates the *EVSiS* and DVS_is and completes the filling of slots in the metaknowledge of agents.

procedure *create_metaknowledge()*;

(* Compute the attributes which are exclusively owned by this agent(EVS_i),
and all the attributes needed (DVS_i)for firing its local ruleset RS_i *)
begin
for i := 1 to k do (* for each subset do *)

EVS_i = VS_i – vertex set pertaining to MRB_i;

DVS_i = $VS_i \bigcup$vertex set pertaining to NRF_i;

endfor;
end;

It may be seen that the number of entries in the union of all NRF_is is equal to the number of entries in the union of all MRB_is and gives the total number of attribute duplications required for the partition, if data duplication is allowed. Otherwise it represents the total *communication coupling* between agents. The number of entries of the form (v_a, j) in NRF_i and those of the form (v_b, i) in MRB_j indicates the data to be exchanged between the agents having P_i and P_j, and hence the communication coupling between them. The communication coupling is useful for reasoning as well as dynamic distribution of the knowledge.

3.3.6 Partitioning Algorithm

The complete algorithm is given below.

Algorithm *connected_static_partitioning*

Assumptions:

The algorithm requires the knowledge graph to be connected

Inputs:

Rulebase consisting of N rules

Proportion $p_1 : p_2 : \ldots : p_k$ in which rulebase subsets are to be obtained

Outputs:

Rulebase subsets $P_1, P_2, .., P_k$ with rules in the given ratio

Directories for the rulebase subsets with attributes details - owned or shared

Steps:

1 Construct the knowledge graph in the form of adjacency list with attributes as nodes, and arcs connecting all input attributes of a rule to each of its output attributes as edges with rule number(s) as the label(s) on each edge.

Compute indegree, outdegree, and degree for each attribute.

2 (* spanning tree generation and marking *)

(a) *generate_spanning_tree();*

(b) *find.chain();*

(c) *mark_edges ();*

3 (* check whether balanced partitioning is possible *)

(a) *determine _size_of_subset();*

(b) *check_if_balance_possible(bal_possible);*

4 (* partitioning *)

(a) *determine_edges_to_be_cut();* (* initial decomposition *)

(b) balance_obtained = false;

Her := 0;

(c) repeat

i *find-data.and.proposed.rule.sets();* (* initial decomposition *)

ii *find-proposed.and-CutscLrules();* (* initial decomposition *)

iii *assign_rules();* (* boundary refinement *)

iv *check_if_balance_obtained(balance_obtained);*

if (balance-obtained = false) and (iter < maxiter) and (baLpossible = true) then

(* maxiter is a constant defined by the user *)

for subsets that do not balance do

shift e_i by one such that

it is in a subset P_i whose $|RS_i| > z_i$;

endfor;

Her := iter + 1;

endif;

until (balance_obtained = true) or (bal_possible = false) or (Her > maxiter)

5 (a) *create_metaknowledge()*;

(b) for i:= 1 to k do (* for each subset*)

print the lists RS_i, VS_i,

NRF_i, MRB_i, EVS_i and DVS_i;

6 stop;

Complexity of the Partitioning Algorithm

Construction of the adjaceny list for the graph corresponding to the rulebase requires $N \bullet$ a time where TV is the number of rules and a is the average number of attributes in a rule. However, this is trivial for any partitioning algorithm and need not be considered in the calculation of complexity of the algorithm.

By using an adjacency list for representing the knowledge graph, step 2(a), i.e., generation of a spanning tree (also an adjacency list) can be done in $O(n)$ time where n is the number of attributes. By storing the semipath lengths for each intermediate vertex from leaf vertices in step 2(a) itself, each of steps 2(b) and 2(c) requires $O(n)$.

Steps 3(a) and 3(b) together require $k + 1$ computations where k is the number of subsets required. Hence step 3 is *0(k).*

Step 4(a), i.e., determination of edges to be cut, requires k iterations and is of $O(k)$, 4(b) requires two computations.

Step 4(c)(i), finding the vertex sets (and the initial proposed rule sets) for the subsets requires examination of all the n vertices. Hence this is $O(n)$.

Step **4(c)(ii)** for finding the complete proposed rule set and conflict rule set requires k iterations as there are k vertex sets. For each vertex all the edges incident on it are to be examined. Assuming the average number of vertices is *n/k,* average

number of edges for a vertex is $\backslash E \backslash / n$ where E is the set of edges in the knowledge graph, for k iterations in the outermost loop, the time complexity is $O(|E|)$.

Step 4(c)(iii) involves finding the nonconflicting rulesets and then assigning the CRS rules to subsets. Determination of nonconflicting rulesets is $O(k)$. Assuming that there are c rules in CRS, each rule usually has 2 to 3 subsets as candidates to which it can be assigned. In the worst case, which usually does not occur in practice, the number of candidate subsets would be k. Updation of NRFs and MRBs requires time $O(k)$. Therefore this step is $O(ck)$. However, since c « TV, c << n, and $c << |E|$overall complexity of step 4(c) is the maximum of $O(n)$ and $O(|E|)$. The number of iterations in the repeat loop is constant, usually about two or three, and hence need not be added to the complexity.

So, the overall complexity of step 4 is $O(max(n,\backslash E \backslash))$.

Steps 5(a) and 5(b) are of $O(k)$; 5(b) is in fact mere outputting of the results.

Therefore, the overall complexity of the algorithm is $O(max(n,\backslash E \backslash))$.

3.4 Examples

We shall illustrate our approach with the the help of a few examples using the rulebase we considered in section 3.2.

3.4.1 Case 1: Two subsets in the proportion 2 : 1

1. The knowledge graph is shown in figure 3.1 and the degree information is given in table 3.1.

2. (a) One of the attributes with zero outdegree is J. It is also a pendant vertex to serve as a root. The spanning tree generated starting with it is shown in Figure 3.3. This has the edges JI, GI, GH, BH, BC, AC, BF, DF, EF and FK.

   (b) The chain is computed as the set of edges JI, GI, GH, BH, BF and FK.

   (c) The highest edge marking in this spanning tree / is computed as 7. The edges belonging to branches are BC, AC, DF and EF. While the first

two belong to the same branch emanating from B, the last two are two different branches emanating from F. Edge BA is marked 5 to indicate its position in the semipath. Arcs AC, DF and EF are not labelled because they do not belong to new rules.

3 (a), (b) Required rule partitioning with sizes $z_1 = 4$, $z_2 = 2$ is possible;

4 **(a)**. $e_1 = 5$; Arc 5 happens to be a branch. Since the number of rules on the chain $|RS_c|$ (=5) is close to N (=6), and since $p_i > p_{i+1}$, e_i is advanced by one more edge. This is equivalent to adding one more edge to the bigger subset, $e_1 = 6$;

(b) The variable balance_obtained is initialized to false, and the number of iterations, iter = 1.

(c)

Disjoint vertex sets VS, = {A,B,C,G,H,I,J}; VS_2={D,E,F,K};

Proposed rule sets PRS_1 ={R1,R3,R4,R6}; PRS_2={R2,R5}.

Cutset edges are BF,GK; Associated rules make the conflict rule set CRS ={ R2, R5};

Initial rule sets for the subsets RS_1 ={ R1,R3,R4,R6} (with corresponding rule count $|RS_1| = 4$) and RS_2 ={} (with $|RS_2| = 0$). All the four rules belong exclusively to subset P_1. P_1 is marked okay as it has the required number if rules. P_2 does not have any rules in it so far.

Of the rules in CRS, for R2, of the atttributes B,D,E and F, only B is needed by subset P_2 from P,. Hence this rule can be allotted to P_2.
Hence, RS_2 ={R2}; NRF_2 ={B(1)}; MRB_1={B(2)};

For R5, of G,F and K, only G is required by P_2 from P_1. Assigning this **rule** to P_2 makes RS_2 ={R2,R5}; NRF_2 ={B(1), G(l)}, and MRB_1={B(2), G(2)}. Now, P_2 has got 2 rules and the agent with P_2 may need to request for attributes B and G from agent with P_1. P_2 is marked okay.
Since both the subsets are in the required proportion, balance is obtained.

5 **Final Partition:**
After filling the remaining metaknowledge slots, the final partition is shown

P_1	P_2
RS_1 ={R1,R3,R4,R6};	RS_2 ={R2, R5};
VS_1 ={A,B,C,G,H,I,J};	VS_2 ={D,E,F,K};
EVS_1 ={A,C,H,I,J};	EVS_2 ={ D,E,F,K};
DVS_1 ={A,B,C,G,H,I,J};	DVS_2 ={ B,D,E,F,G,K};
NRF_1 ={};	NRF_2 ={B(1), G(1)};
MRB_1 ={B(2), G(2)};	MRB_2 ={};

Table 3.2: A 2:1 Partition of Example 1 Rulebase

in table 3.2.

If no duplication is allowed in the working memories of agents with these subsets, each of the disjoint vertex sets VS_i represents the data owned by the agent having subset P_i. NRF_is and MRB_is give the information about the data that needs to be requested from other agents (attribute and from whom to request), and the attributes that may be requested by others (attribute and from whom the request may be sent). There is a possibility that an item generated (as a result of firing the corresponding rules) by one agent may actually be owned by some other agent. Then the first agent may have to send the value immediately to the owner on generating **it.**

If duplication is allowed to some extent, DVS_is represent the attribute set required by agents for firing the rule sets. MRB_is can be used to send copies to the other agents that require the item as soon as it is generated, and NRF_is can be still used to request in advance if necessary.

We can see that the agent with subset P_1 does not need any information from others and the agent with subset P_2 does not have any attributes that may be requested by other agents.

3.4.2 Case 2: Three subsets in proportion 1:1:1

Steps **1** and 2 are same as for case 1.

3 **(a),(b)** Required rule partitioning with sizes $z_1 = z_2 = z_3 = 2$ is possible.

4 **(a)(b).** $e_1 = 3$, $e_2 = 5$; As e_2 happens to be a branch edge, and since all the subsets are to be in the same proportion, we simply advance forward making $e_2 = 6$. The variable balance_obtained is initialized to false, and the number of iterations, iter is made 0.

(c)

$VS_1 = \{G,I,J\}$; $VS_2 = \{A,B,C,H\}$; $VS_3 = \{D,E,F,K\}$;
$PRS_1 = \{R4,R6\}$; $PRS_2 = \{R3,R1\}$; $PRS_3 = \{R2,R5\}$;

Cutset edges are AI,BF,GH,GK;

Corresponding rule set CRS = {R2, R3, R5, R6};

$RS_1 = \{R4\}$; $RS_2 = \{R1\}$; $RS_3 = \{\}$;

For R2, of B,D,E,F, only B is required by P_3 from P_2. Assigning it to P_3, RS_3 = {R2}; $NRF_3 = \{B(2)\}$; $MRB_2 = \{B(3)\}$;

For R3, of B,G,H, only G is required by P_2 from P_1; $RS_2 = \{$ R1,R3$\}$;
$NRF_2 = \{G(1)\}$; $MRB_1 = \{G(2)\}$; Since $|RS_2| = 2$, P_2 is marked okay.

For R5, only G is required by P_3 from P_1; $RS_3 = \{R2,R5\}$;
$NRF_3 = \{B(2), G(1)\}$; $MRB_1 = \{G(2),G(3)\}$; Same attribute G is required by subset P_2 as well as subset P_3. Since $|RS_3| = 2$, P_3 is marked okay.

Lastly, for R.6, of A,G,I, only A is needed by P_1 from P_2. Therefore, RS_1 $= \{R4, R6\}$; $NRF_1 = \{A(2)\}$; $MRB_2 = \{B(3), A(1)\}$;

All the subsets are in the specified proportion; hence balance is obtained.

5 **Final Partition:**

After filling the metaknowledge slots, the algorithm exits. The final partition is shown in table 3.3.

Rulebase partitions obtained for an aerospace vehicle checkout system and medical diagnosis application using our approach are encouraging. These will be discussed in the next chapter.

P_1	P_2	P_3
RS_1 ={R4,R6};	RS_2 ={R1, R3};	RS_3 ={R2,R5};
VS_1 ={G,I,J};	VS_2 ={A,B,C,H};	VS_3 ={D,E,F,K};
EVS_1 ={I,J};	EVS_2 ={C,H};	EVS_3 ={D,E,F,K};
DVS_1 ={A,G,I,J};	DVS_2 ={A,B,C,G,H};	DVS_3 ={ B,D,E,F,G,K};
NRF_1 ={A(2)};	NRF_2 ={G(1)};	NRF_3 ={B(2), G(1)};
MRB_1 ={G(2),G(3)};	MRB_2 ={A(1),B(3)};	MRB_3 ={};

Table 3.3: A 1:1:1 Partition of the Example 1 Rulebase

3.5 Disconnected Components in the Knowledge Graph

Knowledge graphs can also have disconnected components in them. The disconnectedness implies the absence of communication among the agents when each component (or a group of components together) is treated as a subset in some partition and is given to one agent. This actually represents a *functional decomposition* where each component pertains to some subsystem in the whole system, or the components respresent procedures which do not interact at all. Therefore, it is important to be able to identify such components and make use of them appropriately.

Identification of a Component

While generating the spanning tree, if we encounter a vertex that doesn't have another new reachable vertex (from it in the corresponding undirected graph), and none of its predecessors in the spanning tree generated so far have, but there are still some vertices of the knowledge graph which are not included in the spanning tree, it means there are disconnected components in the knowledge graph. These may be indexed with useful information like number of rules in the component for easy retrieval and efficient processing. We shall call the number of rules in a component as the *size of the component,* and the number of rules required for a subset (in the partition to be obtained in the given proportion) as the *size of the subset.*

Identification of Knowledge Graph Components forming Balanced Subsets

A partition with perfect balancing and zero communication is possible in either of the following two cases:

- number of subsets = number of components, and each subset has one component of the same size
- number of subsets < number of components, and each subset has one or more (groups of) components together equalling the subset size.

In all other cases, one or more components need to be broken to get partitioning in the required ratio. In particular, if the number of subsets is more than the number of components, some components must be cut to obtain rule base subsets whose sizes are in the given ratio. However, (even if perfect partitioning is not possible,) it is desirable to identify the subsets and the group of components, if any, with matching sizes. Let us call a subset which is assigned rules satisfying its size requirement a *balanced subset.*

These balanced subsets and the components assigned to them should be separated from the rest of their respective groups and should not be considered for further partitioning. This is because communication among such components is nil. The new proportion (with fewer subsets when compared to the number of subsets in the original proportion) representing the remaining subsets and the unassigned components should be calculated for giving it as input to our partitioning heuristic. We propose a heuristic to identify such balanced subsets and the corresponding components when the knowledge graph is not connected. Partitioning in the modified proportion can be obtained by making minor modifications to the connected_static_partitioning heuristic discussed in section 3.3.6.

The procedure *balanced_subset_components()* identifies the balanced subsets which can be formed from components.

procedure *balanced_subset_components()*;

begin

let k be the number of rulebase subsets required in the proportion $p_1, .. p_k$;

let c be the number of components in the knowledge graph;

let $z_1, z_2, .., z_k$ be the sizes of (rulebase) subsets, sorted in decreasing order;

let $g_1, g_2, .., g_c$ be the component sizes, sorted in decreasing order;

let bs be the number of balanced subsets which have the exact number of rules as its desired size;

let rc be the number of remaining components whose rules have to be assigned to subsets still;

$i, j, sum, diff$ and *temp* are temporary variables;

let *open list* represent the unassigned components;

let *temp list* be a temporary list of components for assignment to a particular subset.

begin

$bs = 0$;

rc = c;

for i := 1 to k do

j := 0;

diff := z_i;

temp := 0;

initialize *temp list* to null;

while *(j* < rc) and *(diff* > 0) do

J := j + 1;

if (g_j <= $diff$) then

diff := $diff - g_j$;

temp := *temp* + 1;

copy the jth component in the *open component* list to *temp list*;

endif;

endwhile;

if *(diff* = 0) then

rc := *rc* — *temp;*

bs := *bs*+ 1;

delete components in *temp list* from open component list;

assign the rules and vertices in *temp list* to RS_i, and VS_i respectively;

$NRF_i = \{\}$;

$MRB_i = \{\}$;

endif;

endfor;

if *(bs* < *k)* then

proceed from step 2h of *connected_static_partitioning* algorithm

considering the remaining subset ratios and the remaining components

endif;

end;

3.5.1 Partitioning Disconnected Knowledge Graphs

Following are the changes required to the static partitioning heuristic developed for a connected knowledge graph, to accommodate multiple components.

Spanning tree generation

If a new vertex is not reachable, in the corresponding undirected knowledge graph, from any of the vertices in the spanning tree forest generated so far, there is another component in the knowledge graph. A new vertex may be chosen as the root for the spanning tree of the next component in the graph, and its spanning tree generation may be continued in the same fashion in the same loop.

Marking of edges

Though chain of the spanning tree in each component has to be identified separately, marking of edges in the spanning tree proceeds continuously with consecutive integers as though there is only one spanning tree present in the entire graph. Both marking of edges and partitioning commence from the largest unassigned component and continue with the next largest until there are no more components in the graph.

Computation of the edges to be cut, and desired sizes of rulebase subsets

Since edges of a component are given integer labels continuing with those given for the earlier component, edges to be cut can be determined in the same way as we did for partitioning a connected graph.

Complexity of the augmented version of partioning algorithm

Besides the additional checking required for identifying the components that form balanced subsets, changes are required to the procedures

- Spanning tree generation
- Finding chain
- Marking of edges

so that partitioning heuristic can be applied on the knowledge graph with multiple components.

Time required for spanning tree generation is the same except that when the open list becomes empty, a new root vertex is chosen for the next component before proceeding with the inclusion of edges in the spanning tree for the new component. Since degree information is stored with vertices while constructing the knowledge graph itself, this doesn't require extra time. Extra time is required only for the selection of new root and to store the component itself. This, however, is negligible as backtracking is minimized by deleting the vertices from the open list the very

first time the vertex is found not to be having new vertices on its edges. Therefore, the time complexity does not increase for this step.

Chains have to be found for each spanning tree separately. However, as both the depth first search for finding the longest semipath from the root and finding the chain from it are of *0(n)* as mentioned earlier, the total time required for finding all chains will also be of $O(n)$.

Storing the spanning forest information in a linear array form after finding the chains (as done for a single spanning tree) keeps its time complexity *0(n)* only.

The extra step *balanced_subset_components* for finding the components that form balanced subsets requires $O(kc)$ time. However, the number of components *c* will certainly be less than the number of data elements n, infact c should be less than *n/2*. Therefore, the overall complexity of the modified version of our heuristic to deal with disconnected components remains the same, i.e., $O(max(n,|E|))$.

3.6 Obtaining Functional Decomposition

Instead of concentrating on the load balancing aspect (by obtaining rulebase subsets in the given proportion of rules), we may also consider a partition that will result in a functional decomposition. There are three possibilities here.

As mentioned in the previous section, if there are disconnected components, the partition representing the components indicates a natural functional decomposition.

However, if the graph is connected, for obtaining a functional decomposition, we need to group rules freely based on the coupling among rules on the edges by relaxing the constraint on the number of rules per subset. Some times, this may result in duplication of few rules in the subsets. An accurate representation of the data and the hierarchy among the objects involved, if considered in the steps of the algorithms already discussed, gives us a partitioning close to a functional decomposition without duplication of rules. For this, we also take into account the *coupling* among the rules incident on edges. The *coupling* can be clearly seen if we observe the rule labels on the edges. The data connected by an edge, and edges emanating from and leaving some data node are closely connected and have some *dependency*. A graph as the representation scheme for the knowledge base depicts the dependencies clearly and

enables our partitioning algorithm exploit the *adjacency* and *dependencies* inherent in the structure. All rules corresponding to an edge should be preferably assigned to the same subset, and if two or more edges have some rules common, they indicate some amount of coupling, i.e., communication between the rules, the precise quantity being proportional to the number of common rules.

Alternatively, functional decomposition can be obtained by considering a final result (attribute with zero outdegree), including all the rules incident, on its incoming edges, and proceeding backwards in the same way until all external input attributes (with zero indegree) are considered. This gives all the rules and data corresponding to a subsystem concluding with the final attribute considered above. There may or may not be some overlap among the rules in different subsets indicating the interaction required among subsets, the latter being the case of disconnected components.

3.7 Knowledge Subset Allocation

Once partitioning is completed, the next step is to assign the subsets to individual agents. We discuss the allocation of subsets obtained in a given ratio representing capacities of agents using the heuristics discussed in sections 3.3.6 and 3.5. We do not discuss the allocation of subsets in a functional decomposition as this may not require load balancing. Even if the subsets have to be allocated in that way, load balancing may be given secondary importance and techniques similar to assignment of components to subsets may be used.

The subsets obtained using our heuristic can be easily assigned if all the agents are situated at equal distances from one another. The allocation problem becomes trivial to that of simply assigning the largest subset to the agent with maximum capacity, the second largest to agent with next highest capacity and so on.

procedure *equal_distance_allocation()*;

*(*Pi* is the list of subsets sorted in the decreasing order of size; *)
(* A_i is the list of agents sorted in decreasing order of capacity; *)

begin
for i := 1 to k do
    assign the subset P_i to agent A_i;
endfor;
end;

In this case, the allocation becomes optimal also. However, the distance between agents need not be equal in real life problems. Hence, the allocation strategy must consider both capacities of agents and distance between them.

3.7.1 Allocation Algorithm

Since our partitioning heuristic partitions the rulebase according to the capacities of agents, the subsets obtained and the agents have a one to one mapping where sizes of subsets match with capacities of agents.

Therefore, the allocation problem can be stated as a mapping problem where there is exact correspondence between the size of a partition and capacity of an agent, and the communication overhead is to be minimized based on the coupling (data transfer required) between subsets and the distance between agents.

Given a partition in the ratio $p_1 : p_2 : .. : p_k$, the coupling between subsets as the amount of data transfer, and the network of agents with capacities $s_1, s_2, .., s_k$, our objective is to map same size subsets onto agents of correspondingly same capacity such that communication overhead is minimized.

We propose a heuristic solution as follows:

procedure *allocate()*;

```
begin
sort the agents on the decreasing order of capacity;
sort the subsets on the decreasing order of size;
sort subset pairs in the decreasing order of coupling between the pairs;
(* group them together based on the amount of coupling *)
sort agent pairs in the increasing order of distance between the agents;
(* group them together based on the distance *)

select the first pair of subsets with the maximum coupling;
select the first pair of agents separated by the first minimum distance;
repeat (* with each subset pair *)
        repeat (* with agent pair *)
                if subset sizes match with agent capacities then
                        assign the subsets to agents of corresponding capacity;
                        mark the assignment okay;
                endif;
                consider the next agent pair
        until the assignment is okay;
        (* both subsets of the subset pair are assigned *)
        consider the next subset pair preferably having
                one subset from (previous pair or) assigned subset list
until (k-1) subsets are assigned;
end;
```

3.7.2 **Example 2**

To explain the allocation algorithm, we shall consider an enlarged rulebase.

The rulebase after adding six more rules to the example rulebase 1 is shown below.

R1.	AB → C	R2.	BDE→ F
R3.	BG → H	R4.	I → J
R5.	FG → K	R6.	AG → I
R7.	KM → L	R8.	M → N
R9.	O → N	R10.	O → P
R11.	Q → P	R12.	KQ → R

The knowledge graph for the example rulebase 2 is shown in figure 3.4.

A spanning tree for the Example rulebase 2 is shown in figure 3.5.

The final partition obtained after the initial partitioning and boundary refinement of the *connected_static_partitioning* heuristic is shown in figure 3.6 and in table 3.4.

Now this partition can be considered as input along with the network topology for illustrating the allocation algorithm.

Given:
the above partition P_1, P_2, P_3, P_4 with sizes of the parts in the proportion 2:1:2:1,
communication coupling among the subsets $P_1P_2 = 2$, $P_2P_3=1$, $P_2P_4 = 1$, and $P_4P_3 = 1$
and a network of agents A_1, A_2, A_3, A_4 with capacities in the proportion 2:2:1:1
and distance between the nodes as $A_1A_2 = 2$, $A_1A_3 = 2$, $A_1A_4 = 1$, $A_2A_3 = 2$, $A_2A_4 = 1$, and $A_3A_4 = 1$,
we shall use our algorithm to find an allocation with less communication overhead, and load balancing.

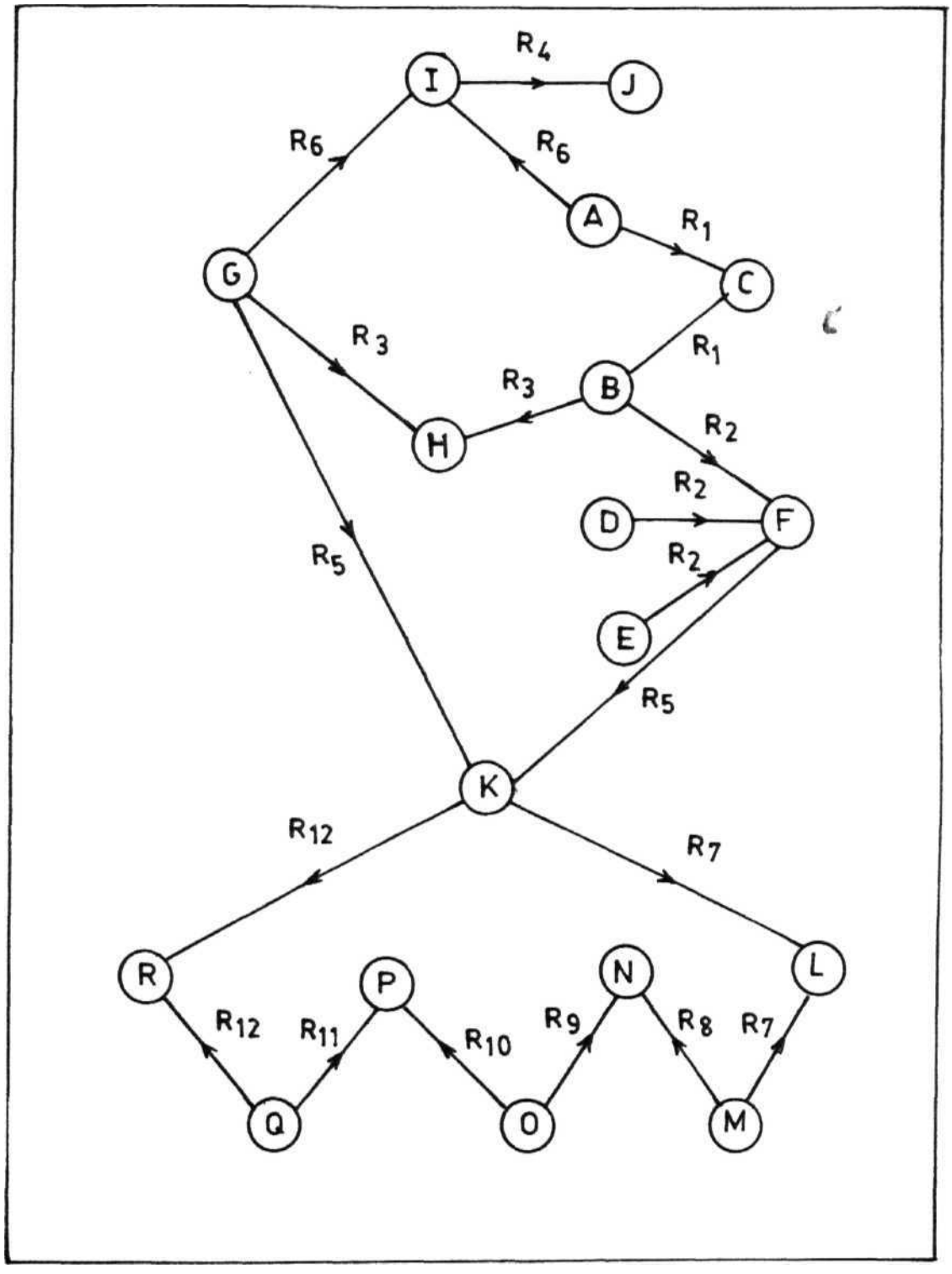

Figure 3.4: Knowledge Graph for the Example Rulebase 2

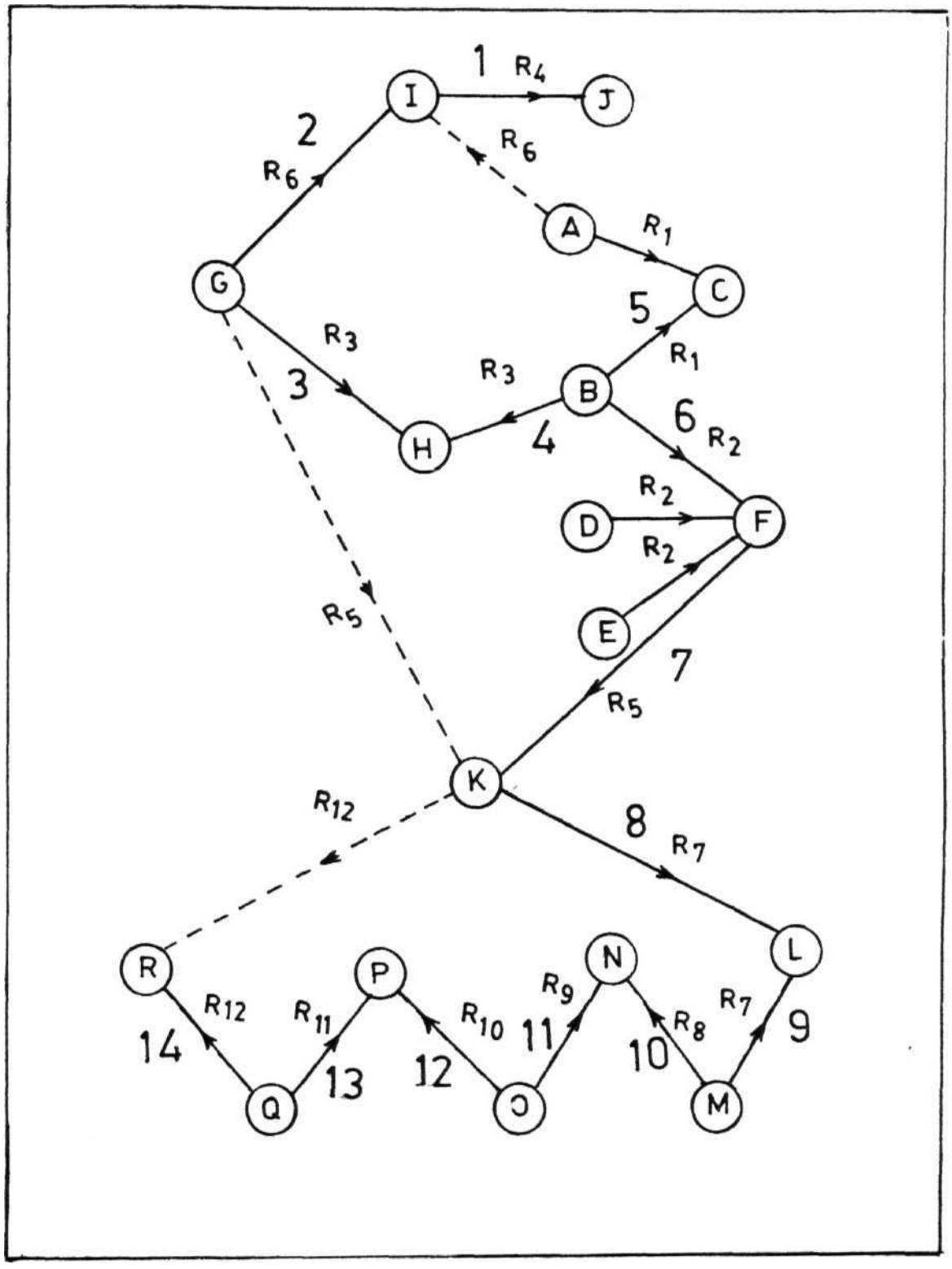

Figure 3.5: A Spanning Tree for the Knowledge Graph of Example Rulebase 2

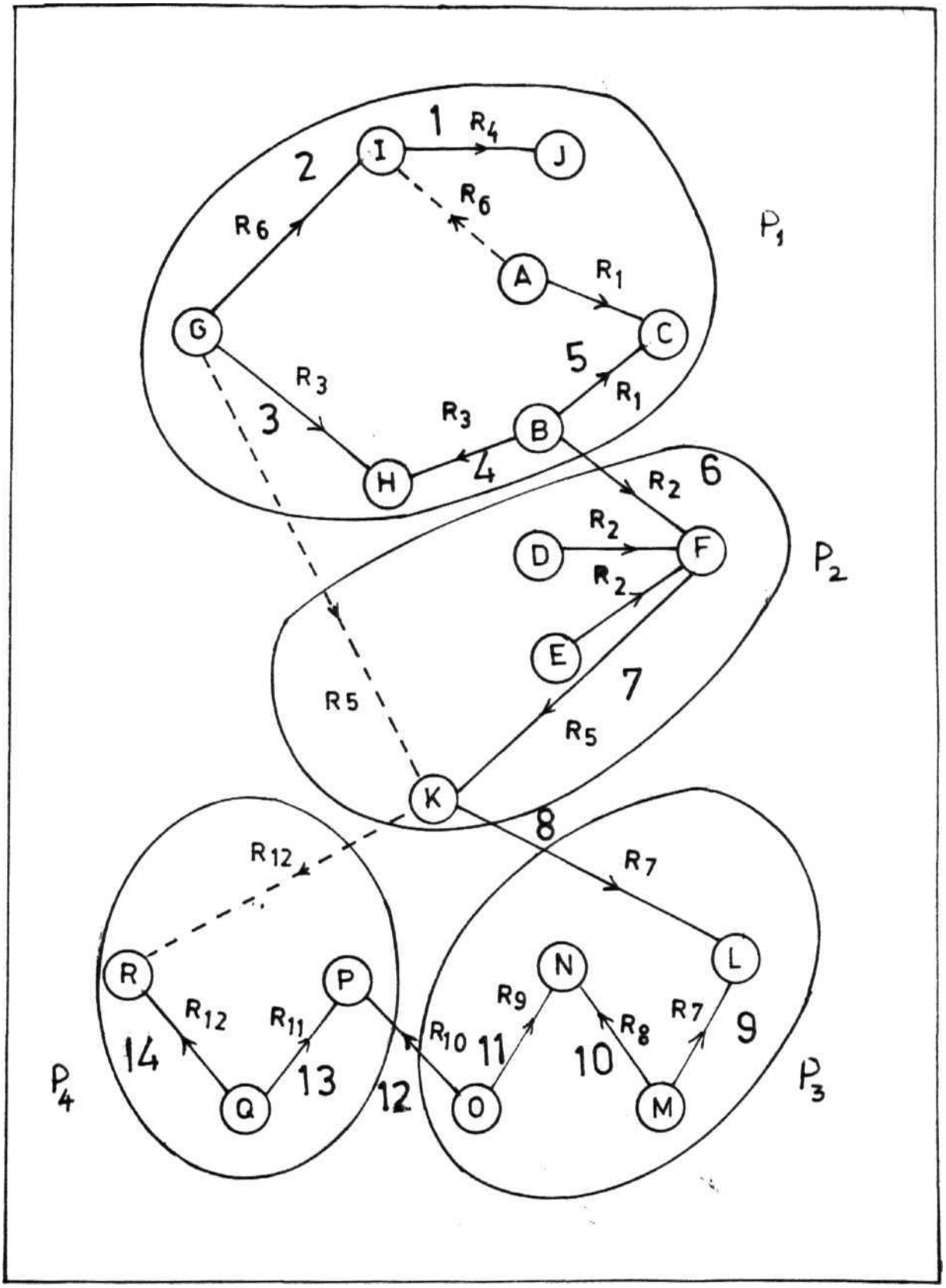

Figure 3.6: A 2:1:2:1 Partition of the Example Rulebase 2

P_1	P_2
RS_1 ={R1,R3,R4,R6};	RS_2 ={R2, R5};
VS_1 ={A,B,C,G,H,I,J};	VS_2 ={D,E,F,K};
EVS_1 ={A,C,H,I,J};	EVS_2 ={D,E,F};
DVS_1 ={A,B,C,G,H,I,J};	DVS_2 ={B,D,E,F,G,K};
NRF_1 ={};	NRF_2 ={B(1), G(1)};
MRB_1 ={B(2),G(2))};	MRB_2 ={K(3),K(4)};
P_3	P_4
RS_3 ={R7,R8,R9,R10};	RS_4 ={R11, R12}
VS_3 ={L,M,N,O};	VS_4 ={P,Q,R};
EVS_3 ={L,M,N,O};	EVS_4 ={Q,R};
DVS_3 ={K,L,M,N,O,P};	DVS_4 ={P,Q,R,K};
NRF_3 ={K(2),P(4)};	NRF_4 ={K(2)};
MRB_3 ={};	MRB_4 ={P(3)};

Table 3.4: A 2:1:2:1 Partition of Example Rulebase 2

We can see that the subsets P\ and P_3 are of size 2 units, and these have to be allotted only to agents of the corresponding capacity, viz., A\ and A_2. Interchanges in allocation are possible only between them. The remaining subsets can be assigned among themselves to any of the remaining agents.

The sorted lists of subset pairs and agent pairs are shown below.

Coupling between subset pairs		Distance between agents	
i)	$P_1P_2 = 2$	i)	$A_1A_4 = 1$
ii)	$P_2P_3 = 1$	ii)	$A_2A_4 = 1$
iii)	$P_2P_4 = 1$	iii)	$A_3A_4 = 1$
iv)	$P_4P_3 = 1$	iv)	$A_1A_2 = 2$
		v)	$A_1A_3 = 2$
		vi)	$A_2A_3 = 2$

We shall illustrate three different cases of allocation.

Considering the first pair of subsets P_1 and P_2, and the first pair of agents A_1 **and** A_4, P_1 can be assigned to A_1 and P_2 can be assigned to A_4. We show this by

$P_1 \rightarrow A_1$, and $P_2 \rightarrow A_4$. There are two more subsets to be assigned to two more agents. Leaving this assignment undisturbed, we proceed with the next subset pair in the list.

If we select P_2P_3 as the next pair of subsets for consideration, it will result in $P_2 \rightarrow A_4$, and $P_3 \rightarrow A_2$. Now that three of the subsets are assigned to three agents, and that the unassigned subset has an assignment compatible with the remaining agent, the final assignment is as shown below.

$$P_1 \rightarrow A_1$$
$$P_2 \rightarrow A_4$$
$$P_3 \rightarrow A_2$$
$$P_4 \rightarrow A_3$$

Had we selected P_2P_4 instead of P_2P_3 we could have got the same result though with an extra step.

Considering A_2A_4, P_2 is already assigned to A_4; P_4 and A_2 are incompatible as they belong to groups of different size and capacity respectively. P_4 belongs to the group of subsets of size 1, and A_2 belongs to the group of agents of capacity 2. Therefore, this assignment is not compatible.

Considering the next agent pair A_3A_4 for mapping the subset pair P_2P_4, the assignment $P_2 \rightarrow A_4$ still holds; P_4 can be safely assigned to A_3 as both of them belong to compatible groups, i.e., of size 1. Now, $P_4 \rightarrow A_3$.

As already three out of the four subsets have been assigned to three of the agents, and as P_3 and A_2 are compatible, we can make the assignment $P_3 \rightarrow A_2$.

The final assignment is shown below:

$$P_1 \rightarrow A_1$$
$$P_2 \rightarrow A_4$$
$$P_4 \rightarrow A_3$$
$$P_3 \rightarrow A_2$$

It can be verified that the this assignment holds true with the other subset and agent pairs in the list.

Similarly, another mapping

$$P_1 \rightarrow A_2$$
$$P_2 \rightarrow A_4$$
$$P_4 \rightarrow A_3$$
$$P_3 \rightarrow A_1$$

also is a minimal communication overhead assignment with the load balanced evenly among the agents.

3.8 Conclusions

We have proposed heuristics for statically partitioning knowledge bases by representing them as knowledge graphs. The partitioning algorithm presented handles both homogeneous and heterogenous partitioning cases and is independent of the graph structure and application domain. A partition in the required rule ratio can be obtained quickly and may be used as it is. Other wise, it may be used as a good starting partition for optimizing performance of algorithms KL and SA. It reduces data duplication and the related inconsistency and communication problems by taking care of the data dependencies. Also, the partition obtained helps in deciding and organizing the working memory contents. When an agent cannot proceed with local problem solving due to insufficient or incomplete data which may be available with other agents, directory information in the form of NRFs and MRBs help to reason, and send requests in a directed fashion. The heuristic for partitioning connected graphs is extended to deal with graphs having components.

The allocation heuristic assigns knowledge subsets with maximum coupling to a closest pair of agents of suitable capacity and minimizes communication while achieving load balancing and less communication.

Task decomposition can be done based on the partitioning of knowledge and data, and subtasks can be assigned to agents accordingly.

The static partitioning algorithm described assumes that all rules have equal probability of being fired. This is because determination of rule firing frequencies at compile time is difficult. However, the varying rule firing frequencies may cause load imbalance during run time and hence dynamic load balancing is necessary. Knowledge distribution for dynamic load balacing is discussed in chapter 5.

The next chapter presents two case studies highlighting the various aspects of the static partitioning algorithm developed in this chapter.

Chapter 4

Static Knowledge Base Partitioning and Allocation: Case Studies

This chapter presents two case studies, viz., Aerospace Vehicle Checkout Application in section 4.1 and Medical Diagnosis of Abdominal Pains in section 4.2. Results of using our partitioning heuristics on the rulebases for the two applications are discussed. Finally the last section gives the conclusions.

4.1 Case Study 1 : An Aerospace Vehicle Checkout Application

A rulebase for an aerospace vehicle checkout application is chosen to test our approach. Thirty seven rules in an encoded form (given in Appendix A) are considered for illustration. Since this is a monitoring application, the system has to continuously keep on firing rules during *countdown* till the launch takes place. The launch cannot take place if there is any malfunction of some part in the physical system. A fault in a system component is indicated by the expert system REX [104] using a *Hold* operation, which is very much like action part of a rule. *Hold* essentially displays a message indicating a fault in a subsystem and suggests corrective action. Performing of this action may be automated or done manually. After waiting for a specified a period of time, the expert system REX resumes the inference process. As a result of taking the corrective action, the new data will be sent to the expert system's working memory as external input.

4.1.1 Partitioning in the given ratio for Load Balancing

Since this is a monitoring application, many rules corresponding to fault diagnosis are for controlling the system, and use the *Hold* operation. Therefore, we treat this as a control variable and use it as a don't care attribute during the actual assignment

of rules. The rulebase follows OPS5 syntax in other aspects. It is observed that about 40% of the rules are complements of another set of rules. Treating all the *Hold* parts of rules as a single variable, the knowledge graph constructed for these 37 rules is as shown in figure 4.7. It may be noted that attributes are shown as big dots in the knowledge graph.

While attempting to generate a spanning tree for the knowledge graph, it is found that there are three components in the knowledge graph. Each of them has 33, 2 and 2 rules respectively. After generating a spanning forest, finding chains for the components, and continuing marking of edges of the components with the same sequence of integers, we have the resulting situation as shown in figure 4.8. The number of rules present on the chain edges is small compared to the total number of rules in the rulebase; branches have more rules. Hence, in the examples discussed below, if some edge to be cut happens to be a branch, we shall not skip it, but take as it is.

Case **1**

Partitioning in the ratio 1:1

Desired sizes of rulebase subsets to be obtained are $z_1 = 19$ and $z_2 = 18$.

On applying the heuristic *balanced_subset_components* discussed in section 3.5 of chapter 3, it is found that the components do not form subsets with rules in the given ratio.

Since the largest component has more rules than required for the first subset, it has to be cut. The edges of the spanning trees for components are labelled as though there is a single spanning tree for all the components.

The total number of rules = 37, and total number of labelled edges = 17.

Edge to be cut $e_1 = 9$.

Cutting at 9th edge gives the proposed rule sets, nonconflicting rule sets and the

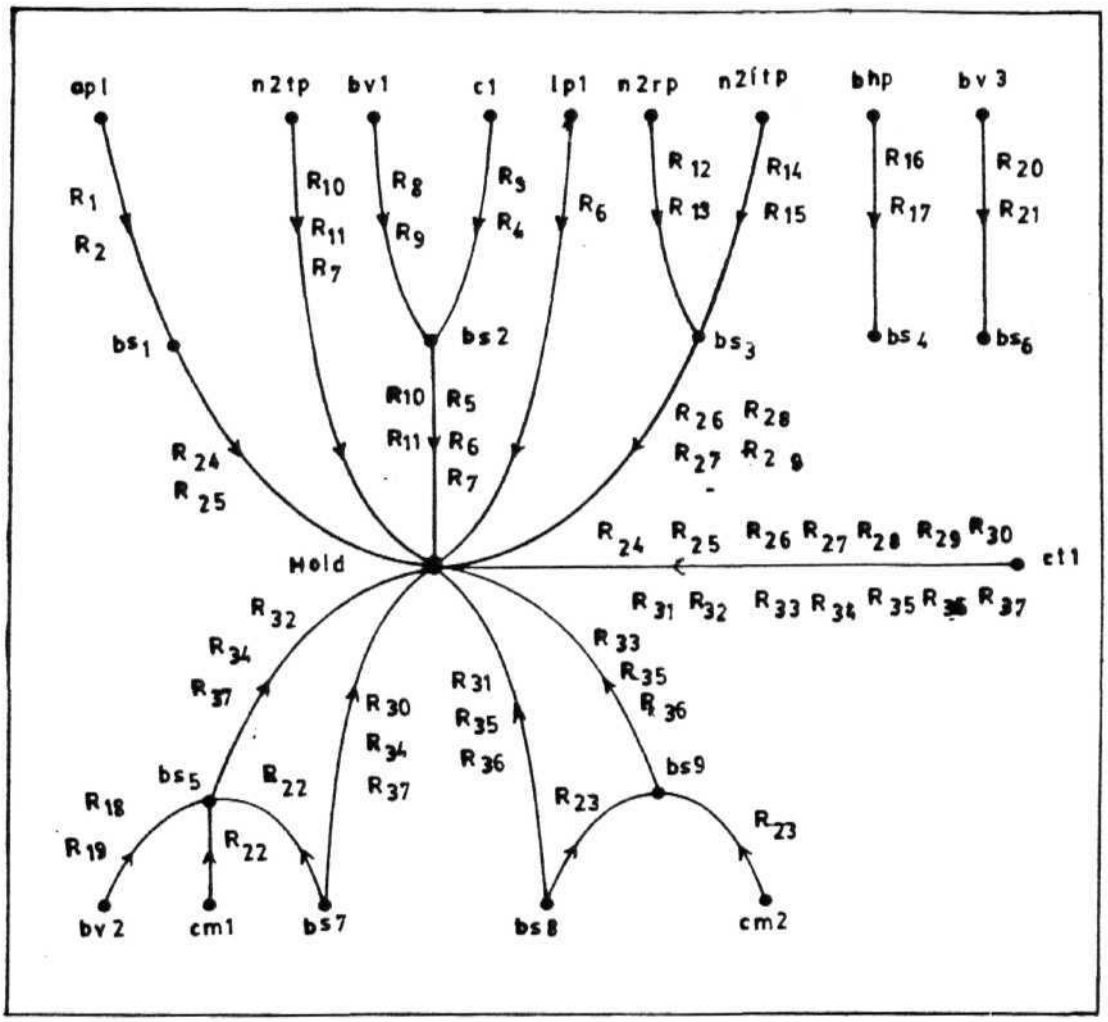

Figure 4.7: Knowledge Graph for a portion of Aerospace Rulebase

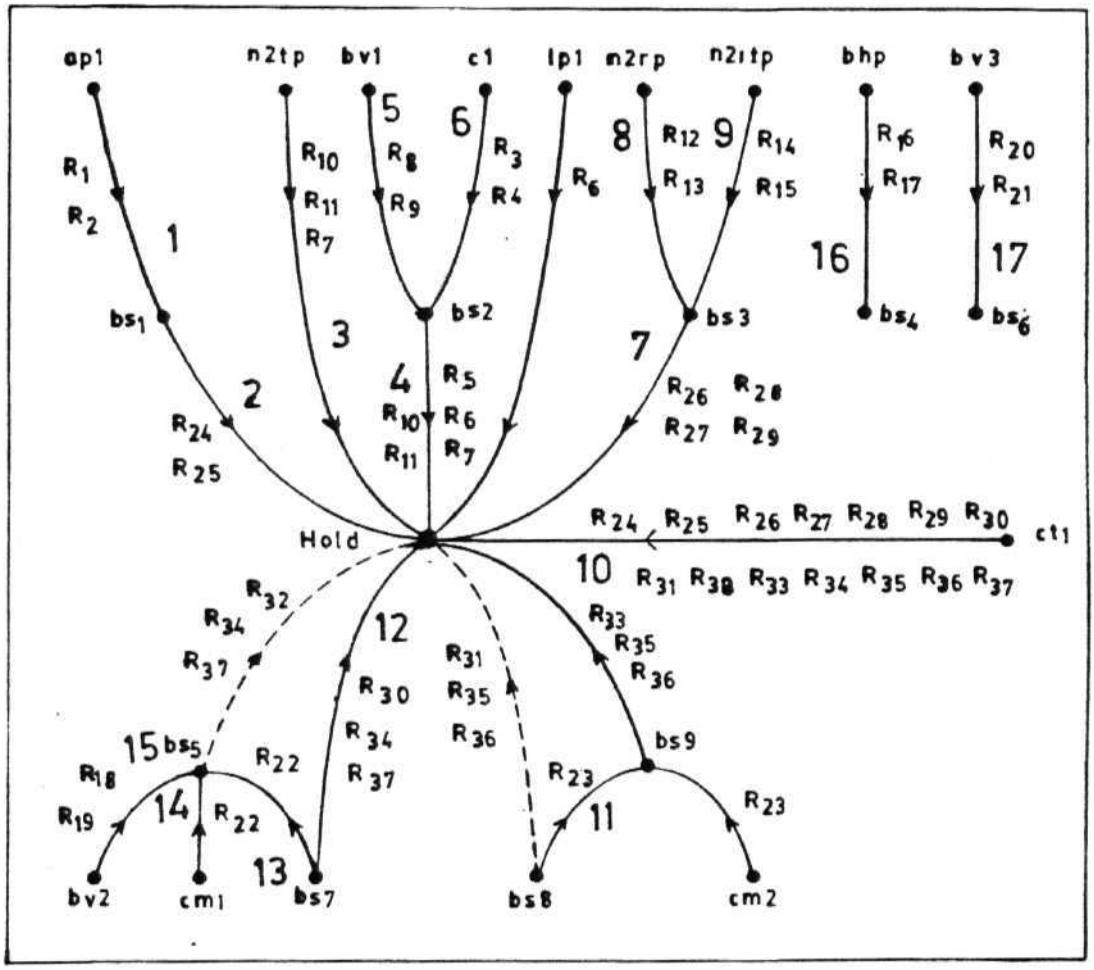

Figure 4.8: A Spanning Tree for the Knowledge Graph of Aerospace Rulebase

conflict rule sets as

$PRS_1 = \{R_1, .., R_{13}, R_{24}, .., R_{29}\}$
$RS_1 = \{R_1, .., R_{13}\}$
$|RS_1| = 13$

$PRS_2 = \{R_{16}, .., R_{37}\}$
$RS_2 = \{R_{16}, .., R_{23}, R_{30}, .., R_{37}\}$
$|RS_2| = 16$

CRS — $\{R_{14}, R_{15}, R_{24}, R_{25}, R_{26}, R_{27}, R_{28}, R_{29}\}$

All the rules in CRS except for R_{14} and R_{15} are of the same size (number of attributes in a rule), i.e., 3 attributes (including the holds i.e., $h6$, *h7, h8, h9,* $h10$ and $h11$. R_{14} and R_{15} have 2 attributes each.

Both the rules R_{24} and R_{25}, have one attribute each, i.e., bs_1 and ct_1 (excluding the control variable *Hold),* in *P* and P_2 respectively. Therefore, these rules can be allotted to any of the subsets P_1 and P_2. Since P_1 requires some more rules for its desired size, these are allotted to P_1. Now, RS_1 = $\{R_1, .., R_{13}, R_{24}$, *R25}* with its size $|RS_1| = 15$.

Similarly, the rules R_{26}, R_{27}, R_{28} and R_{29} should be assigned to P_1. This gives *RS* — $\{R_1, .., R_{13}, R_{24}, .., R_{29}\}$ making the size of RS_1 equal to z_1 i.e., *\RS* — 19. P_1 is marked *okay.*

The remaining rules for which the conflict is to be resolved are R_{14}, R_{15}. Both P_1 and P_2 have an attribute count of 1 for these rules. Since P_1 is in the *okay* list, and P_2 still needs some more rules for its desired size, these may be assigned to P_2. This means, we should duplicate bs_3 in P_2, and as soon any of these rules is fired, the result may have to be sent to P_1. If P_1 requires, it will request it from P_2.

$RS_2 = \{R_{14}, .., R_{23}, R_{30}, .., R_{37}\}$; $|RS_2| = 18$.
Hence P_2 is marked *okay.*

Susbset P_1
$VS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp, bs_3\}$
$RS_1 = \{R_1, .., R_{13}, R_{24}, .., R_{29}\}$
$NRF_1 = \{(ct_1, 2)\}$
$MRB_1 = \{(bs_3, 2)\}$
$DVS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp, bs_3, ct_1\}$
$EVS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp\}$
$\|RS_1\| = 19$
Susbset P_2
$VS_2 = \{n2itp, ct1, bs_7, cm_1, bv_2, bv_3, bs_4, bs_5, bs_6, bs_8, cm_2, bs_9, bhp\}$
$RS_2 = \{R_{14}, .., R_{23}, R_{30}, .., R_{37}\}$
$NRF_2 = \{(bs_3, 1)\}$
$MRB_2 = \{(ct_1, 1)\}$
$DVS_2 = \{n2itp, ct1, bs_7, cm_1, bv_2, bv_3, bs_3, bs_4, bs_5, bs_6, bs_8, bs_9, cm_2, bhp\}$
$EVS_2 = \{n2itp, bs_7, cm_1, bv_2, bv_3, bs_4, bs_5, bs_6, bs_8, cm_2, bs_9, bhp\}$
$\|RS_2\| = 18$

Table 4.5: A 1:1 Partition of the knowledge graph of Aerospace Rulebase

Final Partition:

The final partition is as given in table 4.5. Communication between the agents is necessary for 8 rules with the number of attributes to be duplicated as 2, i.e., for ct\ and bs_3.

Case **2**

Partitioning in the ratio 2:1

As seen in the previous case, there are 3 components in the knowledge graph and the components have 33, 2 and 2 rules respectively.

Desired sizes of rulebase subsets to be obtained are z_1 = 25 and z_2 = 12.

Component sizes and the subset sizes do not match in this case also. Hence, we give integer markings to the edges and cut the graph as though there is only one spanning tree.

The total number of rules is 37, and total number of marked edges /is 17.

Edge to be cut e_1 = 12.

Cutting at 12th edge gives the following proposed rule sets, exclusive rule sets and the conflict rule sets.

$PRS_1 = \{R_1, .., R_{15}, R_{23}, .., R_{37}\}$
$RS_1 = \{R_1, .., R_{15}, R_{23}, .., R_{29}, R_{31}, R_{33}, R_{35}, R_{36}\}$
$|RS_1| = 26$

$PRS_2 = \{R_{16}, .., R_{22}, R_{32}, R_{34}, R_{37}\}$
$RS_2 = \{R_{16}, .., R_{22}\}$
$|RS_2| = 7$

$CRS = \{R_{30}, R_{32}, R_{34}, R_{37}\}$

Since subset $P\backslash$ is already possessing enough number of rules for its required size, all the rules in the CRS will be assigned to P_2. This also has an advantage that strongly coupled rules are in the same subset. Since balanced partitioning is not, possible, the extra rule with P_1 is kept with P_1 itself and the algorithm exits.

Final Partition:

The resulting partition is shown in table 4.6.

Communication between the agents is necessary for 4 rules, i.e., R_{30}, R_{32}, R_{34}, and R_{37}. Number of attributes to be duplicated is 1, i.e., for ct_1.

Case 3

Partitioning in the ratio 1:2:1

We can see that the total number of rules = 37, and total number of labelled edges

Susbset P_1
$VS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp, bs_3, n2itp, ct_1, bs_8, cm_2, bs_9\}$
$RS_1 = \{R_1, .., R_{15}, R_{23}, .., R_{29}, R_{31}, R_{33}, R_{35}, R_{36}\}$
$NRF_1 = \{\}$
$MRB_1 = \{(ct_1, 2)\}$
$\|RS_1\| = 26$
Susbset P_2
$VS_2 = \{bs_7, cm_1, bhp, bv_2, bv_3, bs_4, bs_5, bs_6\}$
$RS_2 = \{R_{16}, .., R_{22}, R_{30}, R_{32}, R_{34}, R_{37}\}$
$NRF_2 = \{(ct_1, 1)\}$
$MRB_2 = \{\}$
$\|RS_2\| = 11$

Table 4.6: **A** 2:1 Partition of the Knowledge Graph for Aerospace Rulebase

= 17.

Desired sizes of rulebase subsets to be obtained are z_1 = 10, z_2 — 19, and z_3 = 8.

Component sizes do not match subset sizes. Hence we proceed in the same way as explained in the previous cases.

Edges to be cut e_1 — 5 and e_2 = 13.

Cutting at 5th and 13th edges gives the proposed rule sets, exclusive rule sets and the **conflict rule** sets as

$VS_1 = \{ap_1, bs_1, n2tp, bs_2\}$
$PRS_1 = \{R_1, R_2, R_5, R_6, R_7, R_{10}, R_{11}, R_{24}, R_{25}\}$
$RS_1 = \{R_1, R_2, R_5, R_7, R_{10}, R_{11}\}$
$|RS_1| = 6$

$VS_2 = \{bv_1, c_1, lp_1, n2rp, bs_3, n2itp, ct_1, bs_8, cm_2, bs_7\}$
$PRS_2 = \{R_6, R_8, R_{12}, .., R_{15}, R_{22}, .., R_{37}\}$
$RS_2 = \{R_{12}, .., R_{15}, R_{23}, R_{26}, .., R_{31}, R_{33}, R_{35}, R_{36}\}$

$|RS_2| = 14$

$VS_3 = \{cm_1, bv_2, bs_5, bhp, bs_4, bv_3, bs_6\}$
$PRS_3 = \{R_{16}, .., R_{22}, R_{30}, R_{32}, R_{34}, R_{37}\}$
$RS_3 = \{R_{16}, .., R_{21}, R_{32}\}$
$|RS_3| = 7$

$CRS = \{R_3, R_4, R_6, R_8, R_9, R_{22}, R_{24}, R_{25}, R_{30}, R_{34}, R_{37}\}$

The rules R_3, R_4, R_8 and R_9 are of size 2; R_6, R_{22}, *R24* and R_{25} are of size 3; R_{34} and *R37* are of size 4.

The rules R_{34} and R_{37} have P_2 and P3 as candidate subsets. P_2 has a higher attribute count, and still requires 5 rules for its desired size. Hence these two rules are assigned to P_2. $RS_2 = \{R_{12}.., R_{15}, R_{23}, R_{26}, .., R_{31}, R_{33}, .., R_{37}\}$; $|RS_2| = 16$.

The next set of rules with size 3 are $R_6, R_{22}, R_{24}, R_{25}$.

Rule R_6 can go to either P_1 or P_2. Hence it is assigned to P_1. $RS_1 = \{R_1, R_2, R_5, R_6, R$
Now $|RS_1| = 7$.

For the rule R_{22}, P3 has the highest attribute count. Hence, the rule is assigned to P_3. $RS_3 = \{R_{16}, .., R_{22}, R_{32}\}$; $|RS_3| = 8$. P_3 is marked okay.

The rules R_{24}, R_{25} should go to P_1 as it has got 2 out of their 3 attributes with it and still requires 4 rules for its share. $RS_1 = \{R_1, R_2 R_5, R_6, R_7, R_{10}, R_{11}, R_{24}, R_{25}\}$
Hence, $|RS_1| = 9$.

The new CRS has only rules of size 2 attributes, i.e., R_3, R_4, R_8 and R_9.

Rules R_3 and R_4 can go to either P_1 or P_2. Assigning them to P_2 makes $RS_2 = \{R_3, R_4, R_{12}, .., R_{15}, R_{23}, R_{26}, .., R_{31}, R_{33}, .., R_{37}\}$ and $|RS_2| = 18$.

Remaining rules R_8 and R_9 can go to either P_1 or P_2. However, since both of them require one rule each, P_1 is given R_8 and P_2 is given *RQ* making them okay.

Susbset P_1
$VS_1 = \{ap_1, bs_1, n2ltp, bs_2\}$
$RS_1 = \{R_1, R_2, R_5, R_6, R_7, R_8, R_{10}, R_{11}, R_{24}, R_{25}\}$
$NRF_1 = \{(ct_1, 2), (bv_1, 2), (c_1, 2)\}$
$MRB_1 = \{(bs_2, 2)\}$
$\|RS_1\| = 10$
Susbset P_2
$VS_2 = \{bv_1, c_1, lp_1, n2rp, bs_3, n2itp, ct_1, bs_8, cm_2, bs_7, bs_9\}$
$RS_2 = \{R_3, R_4, R_9, R_{12}, .., R_{15}, R_{23}, R_{26}, .., R_{31}, R_{33}, .., R_{37}\}$
$NRF_2 = \{(bs_2, 1), (bs_7, 3)\}$
$MRB_2 = \{(ct_1, 1), (ct_1, 3), (bv_1, 1), (c_1, 1)\}$
$\|RS_2\| = 19$
Susbset P_3
$VS_3 = \{bhp, bs_4, bs_5, bs_6, bv_2, bv_3, cm_1\}$
$RS_3 = \{R_{16}, .., R_{22}, R_{32}\}$
$NRF_3 = \{(ct_1, 2)\}$
$MRB_3 = \{(bs_7, 2)\}$
$\|RS_3\| = 8$

Table 4.7: A 1:2:1 Partition of the Knowledge Graph for Aerospace Rulebase

$|RS_1| = 10$; P_1 is marked okay.
$|RS_2| = 19$; P_2 is marked okay.

Final Partition:

The final partition is given in table 4.7.

Communication between the agents is necessary for 7 rules with the number of attributes to be duplicated as 5. Actually, the number of duplications required for ct_1 is 2, and one each for c_1, bv_1, bs_2 and bs_7.

4.1.2 Functional Decomposition

A free grouping of rules connected with *and* nodes forming *clusters of branches* gives us the decomposition shown in table 4.8. The rule set pertaining to a subset P_i is

Subset	Rules
RS_1	$\{R_1, R_2, R_{24}, R_{25}\}$
RS_2	$\{R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}, R_{11}\}$
RS_3	$\{R_{12}, R_{13}, R_{14}, R_{15}, R_{26}, R_{27}, R_{28}, R_{29}\}$
RS_4	$\{R_{18}, R_{19}, R_{22}, R_{30}, R_{32}, R_{34}, R_{37}\}$
RS_5	$\{R_{23}, R_{31}, R_{33}, R_{35}, R_{36}\}$
RS_6	$\{R_{16}, R_{17}\}$
RS_7	$\{R_{20}, R_{21}\}$

Table 4.8: A Functional Decomposition for the Aerospace Rulebase

denoted as $_i$

When we considered the objects involved in the physical system, and their hierarchy (please see Appendix A), it is observed that RS_1 corresponds to those connected with *bs.ftc.hydraulic.system*, RS_2 to *External Supply* object of the *sc_current* subsystem, and RS_3 to *bs.sitvc* subsystem. However, RS_1 and RS_2 are interacting subsystems as the rule numbered R_6 uses data pertaining to both. Similarly, *RS3* needs the rules R_{16} and R_{17} which belong to RS_6 to completely represent its subsystem . Then, RS_4 corresponds to *sc_battery* and *cpif_battery* objects of the subsystem *bs.sc* and RS_5 corresponds to *bs.cpif_bat* object. It is also observed that a branch within a cluster of branches (the latter pertaining to one big object) represents the rules pertaining to a subsystem within the bigger object.

Finally, the sets RS_6 and RS_7 corresponding to the disconnected components in the graph actually belong to the *bs.sitvc* and *bs.sc* objects. These would have been assigned to the appropriate subsets if it is known that the data used by these rules actually belong to the *bs.sitvc* and *bs.sc* objects. Therefore, when knowledge of relationships between data and the physical objects (subsystems) is considered while grouping the rules, the result is a functional decomposition pertaining to the objects of the physical system.

However, it is to be noted that, inspite of the absence of knowledge about object structure and the incompleteness of the rulebase, we are able to get a decomposition that is very close to functional decomposition. Considering the knowledge about objects associated with the data yields better results. In this case study, it is actually the functional decomposition.

Also the partitions obtained using the heuristic with load balancing as the objective are close to functional decomposition.

4.1.3 Discussion

This application presents two situations which were not discussed in the examples of the previous chapter. Edges have multiple rules as labels, and there are three disconnected components in the knowledge graph corresponding to this rule set. The partitions obtained using our algorithm achieve both load balancing and reduced communication. Some of these partitions obtained are very close to functional decomposition also.

4.2 Case Study 2: Medical Diagnosis of Acute Abdominal Pains

Abdominal problems are one major wing of medical therapeutics and surgery. These can be classified as *chronic* and *acute* abdominal problems. Acute abdominal problems are the most common and the most challenging, and many times life threatening to the patient who comes to the emergency department. The physician on duty has to be very diligent and distinguish between medical and surgical abdominal emergencies. In the latter group, crucial decisions have to be taken in favour of surgery to save patient's life. For example, cases like ruptured spleen and ruptured appendix need immediate surgical intervention. To achieve this goal, the physician has to perform various clinical tests. With the help of lab investigations and advanced techniques like X-ray and Ultrasonography, physician can reach at the accurate diagnosis, and patient can be treated in an appropriate way.

Many abdominal problems present themselves with signs and symptoms common to most of them, with some clinching diagnostic features unique to each disease. The physician has to distinguish between those signs and symptoms to arrive at the correct diagnosis [21, 26, 89, 92]. In this context, Expert systems can help to avoid mistakes and improve the efficiency in diagnosis. Partitioning the knowledge base and parallel exploration of multiple diseases can further speed up the process of diagnosis.

4.2.1 Partitioning in the given ratio for Load Balancing

This knowledge base has 400 rules pertaining to 25 diseases. Rules for three diseases viz., Acute Cholecystitis, Perforated Peptic Ulcer and Pancreatitis are given in Appendix B. These are again given in an encoded form for accommodating and visualizing in the form of knowledge graph on the paper. We consider a subset of the rules, i.e., from R_1 to R_{31} corresponding to Acute Cholecystitis for discussion here. The knowledge graph for these rules is given in figure 4.9. It is important to note that unlike in aerospace application, the percentage of data to be shared by the knowledge subsets corresponding to different diseases is considerably more. Even in this case study, as there are a large number of rules present on branch edges, if an edge determined to be cut, e,, happens to be a branch edge, it is left as it is without skipping the branch portion.

Partitioning in the ratio 2 : 1

There is a single component in the knowledge graph.
Number of integer marked edges in the spanning tree / = 31.
Size of the rulebase subsets to be obtained are z_1 — 21, and z_2 = 10.

Edge to be cut e_1 = 20.

Cutting at edge labelled 20 gives the following vertex sets and rule sets.
VS_1 = { vp, vf, li, lif, liq, hpc, po, pot, pi, pl, pr, di, abt, abrt, cvt, cvts, ps, pp, xil, xcal, j, prp, sp, AC1,.., AC9, AC11, AC12, AC13, AC16, AC18, AC19,AC}
PRS_1= $\{R_1, .., R_9, R_{11}, R_{12}, R_{13}, R_{16}, R_{18}$, #19? $R_{25}, R_{26}, R_{28}, R_{31}\}$
RS_1 = {#1, ..,#9, #n, #12? #13? #16? #18? #25? #26? $R_{28}\}$
$|RS_1|$ = 17

VS_2 = {afp, fc, fr, fs, prt, sgb, esr, lc, sal, ecg, sgotl, sd, abwr, abwg, ms, AC10, AC14, AC15, AC17, AC20, AC21, PPU, ACPAN, MI}
PRS_2 = $\{R_{10}, R_{14}, R_{15}, R_{17}, R_{21}, R_{22}, R_{23}, R_{24}\}$
$RS_2 = \{R_{10}, R_{14}, R_{15}, R_{17}, R_{21}, R_{22}, R_{23}$, #24}
$|RS_2| = 8$

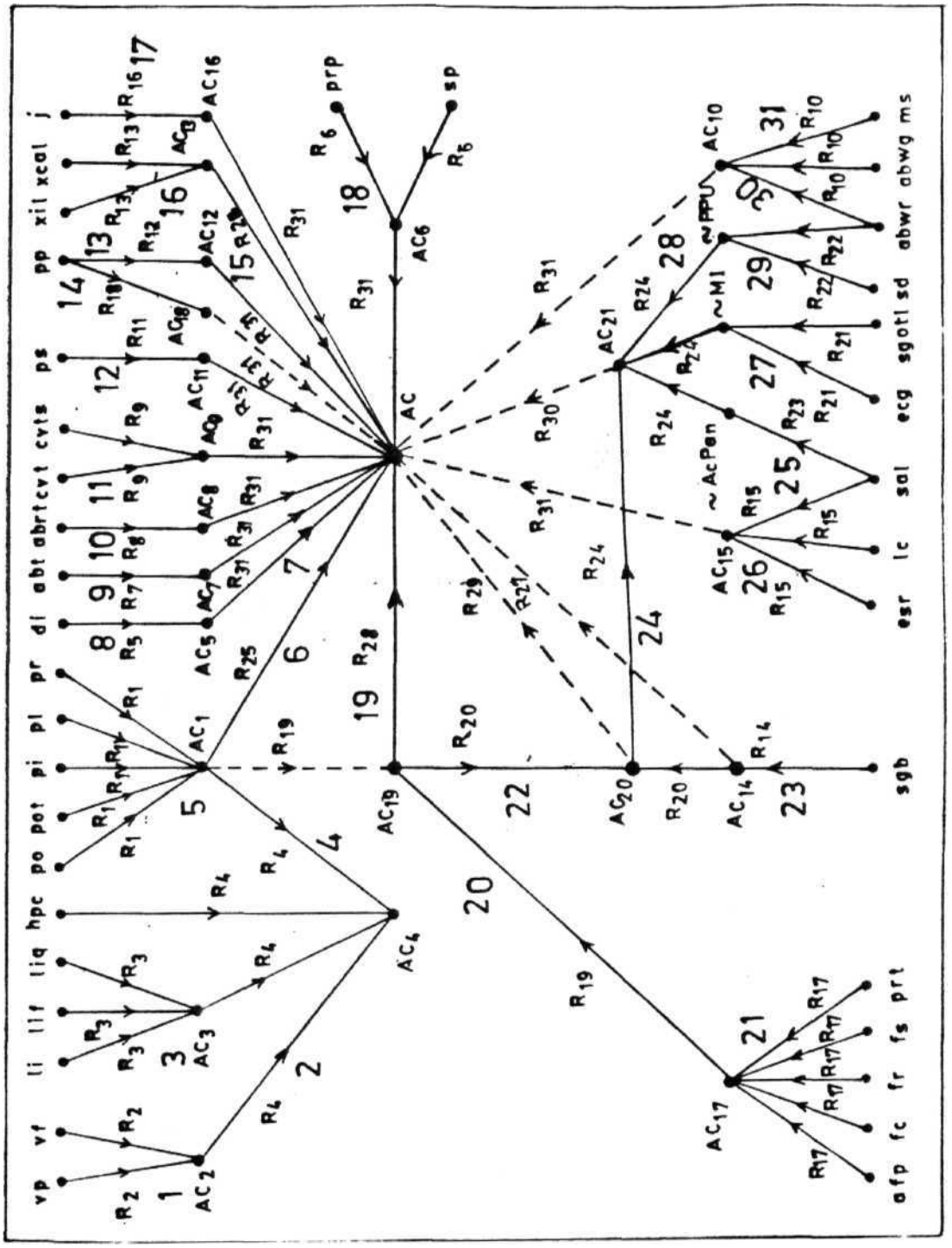

Figure 4.9: Knowledge Graph for a portion of Medical KB

CRS = {#19, #20, #27, #29, R_{30}, #31 }

Among the rules in CRS, R_{31} has the highest number of attributes. Since 9 of the attributes (out of 11) are present with P_1, it should be assigned to P_1. Now, $|RS_1| = 18$.

For R_{19}, P_1 has 2 out of 3 attributes pertaining to R_{19} with it. The rule is assigned to P_1. $|RS_1| = 19$.

Similarly, R_{20} should go to P_2 based on attribute count. Assuming R_{20} to P_2 makes $|RS_2| = 9$.

R_{27} can go to either P_1 or P_2. Therefore, it is assigned to P_1 making $|RS_1| = 20$.

Similarly, R_{29} is assigned to P_1; $|RS_1| = 21$. P_1 is marked okay.

R_{30} can go to either P_1 or P_2. It is assigned to P_2 making $|RS_2| = 10$. P_2 is marked okay.

RS_1 — {#1, ..,#9, #11, #12, #13, #16, #18, #19) R_{25}, ., R_{29}, #31/
$|RS_1| = 21$

Both P_1 are P_2 marked okay; since the required partitioning is obtained, the algorithm exits.

Final Partition:

The final partition is given in table 4.9.

4.2.2 Functional Decomposition

A funcional decomposition may be obtained easily by considering the degree of the attributes. For example, when we encounter an attribute that has zero outdegree,

	Susbset P_1
VS_1	{ vp, vf, li, lif, liq, hpc, po, pot, pi, pl, pr, di, abt, abrt, cvt, cvts, ps, pp, xil, xcal, j, prp, sp, AC1,.., AC9, AC11, AC12, AC13, AC16, AC18, AC19,AC}
RS_1	$\{R_1, .., R_9, R_{11}, .., R_{12}, R_{13}, R_{16}, R_{18}, R_{19}, R_{25}, .., R_{29}, R_{31}\}$
NRF_1	{(AC10,2), (AC14,2), (AC15,2), (AC17,2),(AC20,2)}
MRB_1	{(AC,2), (AC19,2)}
$\|RS_1\|$	21
	Susbset P_2
VS_2	{afp, fc, fr, fs, prt, sgb, esr, lc, sal, ecg, sgotl, sd, abwr, abwg, ms, AC10, AC14, AC15, AC17, AC20, AC21, P̃PU, ÃCPAN, M̃I}
RS_2	$\{R_{10}, R_{14}, R_{15}, R_{17}, R_{20}, .., R_{24}, R_{30}\}$
$\|RS_2\|$	10
NRF_2	{(AC,1), (AC19,1)}
MRB_2	{(AC10,1), (AC14,1), (AC15,1),(AC17,1),(AC20,1)}

Table 4.9: A 2:1 Partition of the Medical Diagnosis Application Rulebase

considering all rules incident on the incoming edges while traversing back to data with zero indegree in the branches (connected to it) gives a functional decomposition.

If we consider the rules pertaining to the three diseases given in Appendix B, the above strategy separates rules for each disease. Some rules using data associated with more than one disease were duplicated in all the concerned subsets.

4.2.3 Discussion

Our partitioning algorithm discussed in section 3.3.6 always guarantees static load balancing. It gives good results with respect to both load balancing and communication for information exachange if an appropriate proportion of the sizes of rulebase subsets is given as input.

However, in this application, a large number of attributes (data) are shared by closely related diseases. Therefore, communication will be more for some subset sizes because data is distributed without any replication in our model. This is true in situations where rules pertaining to one disease are spread over multiple agents. An

example is a partition in which the rules corresponding to the diseases Cholecystitis, Pancreatitis and Perforated Peptic Ulcer are partitioned into three subsets using the static load balancing partitioning algorithm, and the subsets are allocated to three agents. This problem may be remedied by allowing duplication of necessary data at the required places. The data updation overheads will also be less as most of the values remain same for a large percent of data. Otherwise, checking once for the already obtained data eliminates unnecessary communication.

To summarize, functional decomposition is more suitable to this application with respect to the information exchange required. Pure load balancing yields good results with respect to information exchange also when the necessary data is duplicated at all places. Instead, partitioning for load balancing within a partition corresponding to a functional decomposition is a better choice. Futher, in an application like medical diagnosis, the entire knowledge base need not be active at the same time. Therefore, a control component may first be initiated and then the process corresponding to the related diseases be invoked dynamically.

Hence, dynamic knowledge distribution becomes appropriate not only for dynamic load balancing but for dynamic invocation of knowledge subsets depending on the actual problem being solved and the solution progress. Dynamic distribution of knowledge is the subject of the next chapter.

Chapter 5

Dynamic Knowledge Distribution

In Chapter 3, we have seen how knowledge can be partitioned and allocated statically. However, dynamic knowledge distribution becomes important for dynamic load balancing, problem based knowledge allocation and for adapting to changes to the knowledge base from time to time. As with static partitioning, many of the techniques aim at homogeneous partitioning and do not take data distribution into account. Besides heterogeneous partitioning and data distribution, the complex interdependencies among the knowledge subsets require special attention. Considering these three aspects, we develop new techniques for dynamic knowledge distribution in this chapter. The chapter is organized as follows. Section 5.1 gives an introduction to the problem; it explains the importance of dynamic knowledge distribution and the need for new techniques. Section 5.2 discusses the various heuristics to partition and allocate knowledge dynamically for dynamic load balancing. This covers making local exchange of rules between (neighbouring) knowledge subsets, repartitioning and reallocating the entire knowledge base, and adapting to changes to the knowledge base from time to time. Section 5.3 discusses problem based dynamic knowledge distribution and the last section gives the conclusions.

5.1 Introduction

Dynamic knowledge distribution becomes important for the following reasons:

- Some AI programs exhibit a rapidly changing computational requirement and have unpredictable run time behaviour [128] because they must respond to changes in external environment. Due to this characteristic, scalability is hard to **achieve** unless both static and dynamic load balancing are done. Even if the **run time** situation does not change so quickly and unpredictably, in initial stages of problem solving, knowledge and data may have to be distributed

arbitrarily as it may be difficult to have a compile time estimate of the run-time behaviour. In such cases, redistribution (reorganization or remapping) of knowledge is needed to balance the load, minimize communication and improve performance. Organizational self design [71] is an example for adaptive reorganization of the agents and their knowledge for better performance in Distributed AI systems.

- When new knowledge is being aquired either by local learning or from the external world, knowledge base in its entirety undergoes changes. Assigning new knowledge to appropriate agents, and checking for load balance become crucial in such situations.

- When agents are created dynamically as problems arrive, as in Contract Net Protocol [114, 115], an agent may have to be provided with appropriate knowledge for solving a task assigned to it. Knowledge exchange in such situations may be either directly done between two contractor agents themselves or through a manager. This knowledge transfer may also be considered as a separate task by itself.

- When the main memory is insufficient to accommodate the entire knowledge base, dynamic relocation of the modules or subsets of knowledge may be necessary.

- Knowledge distributed to agents for solving a particular task may not be suitable for solving another task. Therefore, knowledge of an agent may undergo revisions in the form of exchanging portions of knowledge subsets already available with other agents. Knowledge may need to be duplicated (where often required) or deleted (where not required). The goal is to make the partition self-sufficient for solving most of the tasks, thus minimizing information exchange. The new distribution is expected to be better suited for a set of constraints and load at a given time. This is similar to the dynamic allocation of documents based on their use at a node and other constraints [65].

Keeping in view the goals of *load balancing* and minimal communication, dynamic knowledge distribution can be done in the following ways.

1. Minor redistribution in the form of local exchanges between neighbouring agents (with or without duplication).

2. Repartitioning and reallocating the entire knowledge base periodically over fixed or varying length time intervals based on the run-time statistics. If run-time behaviour becomes stable after certain time, static distribution itself may include an estimate of this.

However, if pure functional decomposition is the objective, there is no need for reorganization. The knowledge remains where it belongs.

In other cases, knowledge distribution involves the following.

1. Distributing the required chunk of knowledge after the problem arrival (if the required knowledge is not already available with the agent).

2. Limiting the changes due to additions (or deletions) as a result of new knowledge acquired, to the knowledge subset that is closely associated with the change and its neighbouring subsets. If this leads to severe imbalance over a period of time, complete redistribution may be necessary.

Dynamic load balancing has been addressed in areas like parallel and distributed computing [3, 17, 20, 25, 42, 66, 105, 110, 128]. However, these are suitable basically for data parallel programs and specific types of task interaction patterns.

Dynamic distribution of knowledge in production systems has been dealt with by Ishida et al. [71] in the context of *organizational self design.* Knowledge representing agents is *composed* or *decomposed* adaptively depending on the varying performance changes in the system. This has already been discussed in chapter 2. It was also noted that though two agents are composed to form one agent, and one agent is decomposed into two by arbitrarily clubbing and halfing rules in the agents, need for better methods has been emphasized. This leads to unnecessary communication overheads but the problem is less severe when agents are not separated by physical distance and a shared memory is used. For effective reorganization and communication, the interdependencies need to be taken into account.

Dynamic scheduling of production rules (distribution at run-time) is discussed by Tout and Evans [119]. If the queue of tasks to be investigated is not empty, each

idle processor requests and gets a predefined number of rules, and investigates to find the applicable rules. Their performance analysis indicates that the design of parallel expert systems with local working memories improves speedup as well as efficiency.

However, both organizational self design [71] and dynamic scheduling of forward chaining systems [119] assume a homogeneous partitioning. Further, interprocessor distance is not considered and any agent can process any rule. While the first does not consider data distribution, the second allows full duplication or no duplication of the database. Shared memory systems also have synchronization and concurrency problems [57].

However, in distributed AI systems, agents could be separated by physical distance and communicate may be by message passing. Further, dependencies among rules must be taken into account.

Keeping local copies of the entire database involves updation of all copies in local memories after every cycle. Instead, an incremental update followed by periodic reorganization with necessary data stored only in the local copies can reduce the update overhead and inconsistency problems. In this case at the end of every cycle, there might be only a few changes from each node and the total number of changes will have an upper bound equal to the number of data elements in the working memory. This is beneficial especially when the amount of data shared is not large. Shared memory systems have problems like synchronization and concurrency control.

Further, the work by the above two groups assumes dynamic creation of processes (agents) to select and assign a set of rules. When agents are fixed (with a given capacity), the changing knowledge and data make different instances of the agents. This situation, however, requires new techniques for dynamic distribution to be developed.

Unlike in data parallel programs, where the interaction is only between adjacent modules (when the task graph is a chain of modules) or there is no interaction at all [15, 20, 75], knowledge subsets in Distributed AI problems exhibit complex interdependencies. Therefore, mechanisms are needed for other domains to take care of more complex relationships among modules.

First, we attempt to see how our static partitioning heuristics can be modified to suit dynamic load balancing with reduced communication. Then, we discuss about the dynamic reorganization which becomes necessary with changes to the knowledge base and the problem based distribution of knowledge [97].

5.2 Dynamic Partitioning and Allocation with Load Balancing

A dynamic load balancing scheme consists of four policies: *control policy* (who makes the load transferring decision), *information policy* (method of exchanging load status), *location policy* (determining the possible candidates to whom the task has to be transferred) and *transfer policy* (when exactly the dynamic load balancing procedure has to be invoked) [128]. The terms *agent, node*, and *process* are used synonymously.

There are various overlapping approaches to dynamic load balancing. In a *centralized* approach, control is authorized to a single controller whereas in a *decentralized* approach, control is distributed to each node or multiple nodes [128]. In a *deterministic* (or state dependent) approach, balancing is done based on current system state, utilization of CPU and memory, average response time, etc., whereas in a *probabilistic* (or nondeterministic) approach, a job is despatched according to a set of branching probabilities [42, 128]. Similarly, a distribution scheme could be either *adaptive* or *nonadaptive.* In the former, load balancing policies are modified as the system state changes while in the latter, the balancing policy remains unchanged. Finally, in a *cooperative approach*, nodes coordinate with each other in making a decision, while in a *noncooperative approach (isolated),* this is done without considering states of other nodes in the system [128].

In all the heuristics to be discussed, we make a few assumptions about the physical network of agents. The network protocol is assumed to be taking care of the data exchange format and other communication aspects. It is also assumed that the distance metric used incorporates factors like communication via other nodes and that infinite buffer space is available at the sender, receiver and intermediate nodes thus implying no queueing delays.

Further, we differentiate between a *physical network* and a *logical network* in

the system. The *physical network* is the actual collection of agents with given interconnections. A mapping of the possibly interdependent knowledge subsets onto physical network gives us the *logical network.* Thus the logical network may be a subgraph of the physical network. It represents the physical communication links which become active as a result of the communication required among knowledge subsets assigned to agents in a particular allocation. When there is a change in the knowledge possessed by an agent, either due to redistribution (repartitioning and allocation) or because of transfer of small portions, the logical network also may change. The new distribution may have new coupling between subsets and hence a new logical network may become active.

We shall now explain the load balancing policies used in our heuristics.

Control, Information and Transfer Policies

The control policy is centralized in all the heuristics to be presented.

The general strategy used in information and transfer policies is to measure the load on each agent during a certain time interval and use this information to predict and balance the load in the next interval. An update policy is used to exchange load information among the agents and compute the desired load for each agent using the available (actual) load information. Agents are categorized as heavily loaded or lightly loaded depending on whether the difference between the actual load and the desired load is positive or not. This information is used to bring loads on agents close to their desired loads in the next interval.

Depending on the variance in the load imbalances in the previous intervals, length of the time interval is changed to achieve an adaptive load distribution. Adaptive load balancing can be achieved by invoking the particular balancing heuristic whenever the load crosses a predefined threshold also.

Location policy

In a CPS system using distributed local memories and message passing, load transfer should be done between neighbouring agents to the maximum possible extent.

After identifying the heavily loaded and lightly loaded nodes, if the execution of tasks is independent of the node as well as other tasks, it is simple to transfer load from a heavily loaded node to a lightly loaded node. However, when there is interdependence among the tasks, transferring excessive load from one node to another is not straightforward. The dependencies impose some constraints.

If the load is transferred arbitrarily between a heavily loaded and a lightly loaded node, particularly when there is no connection between them in the logical network, it leads to unnecessary communication and excessive delays. Therefore, while transferring rules, it is necessary to follow the logical network considering the interdependencies among the subsets in the knowledge graph. This implies that for rules to be transferred between agents, the corresponding subsets should have a dependency arc between them. This is also because firing frequencies of rules with input or output dependency are interdependent and an arbitrary transfer will increase the communication. Secondly, this implies that when a heavily loaded node is not adjacent to a lightly loaded node in the logical network, the transfer cannot take place directly between them, but should *ripple* through the nodes in between.

Imbalance vs. Communication Delays

Achieving perfect load balance is not possible because a rule cannot be subdivided to make its firing frequency exactly match the desired load. Therefore, the goal is to have a fairly optimal load distribution with less communication. Either of these requirements may be considered as primary and the other secondary in a given situation. Transferring of knowledge or data for load balancing should not result in more delay due to extra communication than that caused by executing the excess load locally. Therefore, information policy should consider and compare the delays involved in both imbalance and communication as above.

Every node can be allowed to take an extra load (defined as a threshold) r that can be tolerated by the system. Only when the threshold is exceeded, a load transfer decision may be taken. Similarly, data updation overheads may also be taken into account. The threshold value is taken as some constant in the examples discussed below.

Starting point

Our dynamic load balancing and distribution start at a node with the heaviest imbalance. The option of starting the balancing process at a heavily loaded node is chosen because heavily loaded nodes cause more delays in problem solving.

Stop criterion

Since perfect load balancing is not possible, as discussed earlier, we must be able to determine when to stop. One method is to specify a threshold τ as discussed earlier, another is to limit the number of iterations. We shall use the first method in our heuristics.

5.2.1 Local transfer of Knowledge for Load Balancing

The heuristic presented below first computes the load on each agent during an interval, determines the load imbalance, and adjusts the load imbalances starting at a heavily loaded node.

Load on an agent A_i in an interval A is calculated as the product of the average rule processing time (average rule matching time t_m + average rule firing time t_f) and sum of the firing frequencies of all rules assigned to A_i.

Sum of the loads and agent capacities are used to calculate the ideal loads for agents. Imbalance with an agent, /,, is determined as the difference between *ideal (or desired) load* D_i and *actual load* L_i corresponding to the previous interval, i.e., $I_i = L_i - D_i$. If I_i is positive, the node is considered to be *heavily loaded,* and if I_i is negative the load is considered to be *lightly loaded.* Obviously, if there is an imbalance, there must be atleast one node which is heavily loaded and another which is lightly loaded.

Starting at a subset (and hence the concerned agent) with the heaviest imbalance, loads are adjusted by transferring some excessive load from the heavily loaded node to a lightly loaded node such that the resulting loads on the agents are close to their ideal loads in the previous interval.

Load transfer has to be made by shifting a few rules from the heavily loaded node to a lightly loaded node. A rule should be chosen for transfer such that it is close to the rules in the lightly loaded node, i.e., the semipath between this rule and another in the lightly loaded node is as short as possible (=1). After selecting such a rule, it is transferred only if any of the following is true:

- the imbalance vanishes, i.e., when the product of its rule firing frequency (sum of frequencies) and the average processing time is equal to the excessive load
- the (more) heavily loaded node becomes less heavily loaded, i.e., when the load corresponding to the rules as described in the above case is less than excessive load
- the resulting magnitude of imbalance is less than the magnitude of the present imbalance at least at the heavily loaded node, i.e., when the load corresponding to the rule is only slightly greater than the excessive load.

However, since the knowledge subsets will have interdependencies, it is necessary to follow the logical network and transfer the load only to its closest neighbours. Supposing more than one neighbour is a candidate for transferring the load, the closest neighbour with closest imbalance in the opposite sense, i.e., lightly loaded node (for a heavily loaded one) is chosen for transferring the excessive load.

If the heavily loaded node doesn't have a lightly loaded node as its neighbour in the *logical network*, the load changes *ripple* through the intermediate neighbours. It is preferable to keep the number of such intermediate nodes small. Also, balanced nodes are kept undisturbed as far as possible.

After the transfer is made, the directories of agents including the rule sets, vertex sets, NRFs and MRBs are updated to reflect the change.

The process is repeated with the new heaviest node until the load imbalance is below the threshold r (some constant in our discussion). If the variance in the load imbalances in successive time intervals of observation increases, the length of the interval is shortened, otherwise it is increased by an appropriate amount.

Algorithm *dyndistl*

(* dynamic distribution of knowledge using local changes to knowledge subsets *)

1. perform initial partitioning and allocation;

2. after a time interval A

a. for each agent (part) A_i

let f_j be the number of times $R_j \in A_i$ is fired;

actual load $L_i = ((t_m + tt_f) * \sum_{j \in A_i} f_j$;

endfor;

b. for each agent A_i

let the desired load $D_i = \sum_{j=1}^{n} L_j * \frac{P_i}{\sum_{j=1}^{n} P_j}$;

where P_i corresponds to the capacity of agent A_i;

imbalance $I_i = L_i - D_i$;

if $I_i > 0$ then include A_i in *heavily loaded node list HL*

else if $I_i < 0$ include A_i in *lightly loaded node list LL*;

endif;

endif;

endfor;

c. let the variance of imbalances be $VI_\lambda = \sum_{i=1}^{n} |I_i|^2/n$;

d. let A_h be an agent with highest imbalance in HL

repeat

let RS_h and VS_h be the rule set and vertex set respectively of A_h;

if $\exists$ a neighbour in LL then

select a node A_a such that $|I_h| - |I_a|$ is the smallest

else select the closest neighbour A_a in the logical network;

(* the changes in the load ripple through the agents between *)

(* A_h and A_l where A_l is in LL *)

endif;

let RS_c be the rules on the edges E_c between A_h and A_a;

if $RS_c \cap RS_h$ is a null list then

pull out a few edges from A_h into A_a such that

the new $RS_c \cap RS_h$ is not null;

update VS_h, VS_a;

update RS_h, RS_a;

update MRBs and NRFs;

endif;

let R_c be a rule in RS_c;

let f_c be the firing frequency of R_c;

let f_l be the corresponding load;

$f_l = f_c * (t_m + t_f)$;

let v_i and v_j be the end vertices of e_c

(*associated with R_c*) in A_h and A_a respectively *)

while $(|I_h - f_l| > \tau)$ do

$RS_a = RS_a \cup R_c$;

$RS_h = RS_h - R_c$;

$VS_h = VS_h - v_i$;

$VS_a = VS_a \cup v_i$;

$L_h = L_h - f_l$;

$L_a = L_a + f_l$;

$I_h = I_h - f_l$;

$I_a = I_a + f_l$;

let v_k and v_i be the end vertices of the new

outcoming edge where $v_k \in A_h$ and $v_i \in A_a$;

(* v_i is the common vertex for the corresponding undirected edges

$v_i v_j$ and $v_k v_i$ where v_i now belongs to A_a *)

$NRF_h = NRF_h - (v_j, A_a) \cup (v_i, A_a)$;

$NRF_a = NRF_a - (v_i, A_h) \cup (v_k, A_h)$;

$MRB_h = MRB_h - (v_i, A_a) \cup (v_k, A_a)$;

$MRB_a = MRB_a - (v_j, A_h) \cup (v_i, A_h)$;

select the next suitable rule R_c for transferring, if any,

let f_c be the firing frequency of R_c;

let v_i and v_j be the end vertices of e_c on which R_c is incident;

endwhile;

mark A_h unsuitable for futher transfer of load;

choose the next node in HL;

until the maximum of all node imbalances is τ;

3. if the variance in the load imbalances over the past few intervals

is increasing, then set the next interval shorter;

else if it is decreasing, set the next interval longer;

4. Start the new interval, and go to step 2

Example

We shall consider the 2:1:2:1 partition of rulebase 2 discussed in section 3.7. However, interagent distances are considered to be 1, and we call the agents having subsets P_1, P_2, P_3 and P_4 as A_1, A_2, A_3 and A_4 respectively.

Now the problem can be formally stated as below.

Given

1. the network of agents A_1, A_2, A_3 and A_4 with capacities in the proportion 2:1:2:1,

2. the communication coupling $P_1P_2 = 2$, $P_2P_3=1$, $P_2P_4=1$, $P_4P_3 = 1$ and

3. the distance between agents as $A_1A_2 = 1$, $A_1A_3 = 1$, $A_1A_4 = 1$, $A_2A_3 = 1$, $A_2A_4 = 1$ and $A_3A_4 = 1$;

4. the average rule processing time $(t_m + t_f)$ as 1, and the firing frequency vector of rules $\{R_1,..R_{12}\}$ in a given time interval as

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}
1	2	1	2	2	2	2	2	1	3	3	3

5. the allocation of rule subsets $P_1 = \{R_1, R_3, R_4, R_6\}$, $P_2 = \{R_2, R_5\}$, $P_3 = \{R_7, R_8, R_9, R_{10}\}$ and $P_4 = \{R_{11}, R_{12}\}$ to A_1, A_2, A_3 and A_4 respectively

6. rulebase subset interaction graph obtained for the 2:1:2:1 partition of the example rulebase 2 as shown in section 3.7 of chapter 3,

let us find a snapshot of the dynamic allocation at the next time interval based on the previous interval statistics using our heuristic *dyndistl.*

Load situation on agents is given as

Agents	A_1	A_2	A_3	A_4
Desired Loads	8	4	8	4
Actual Loads	6	4	8	6
Imbalance	-2	0	0	+2

The node with the heaviest imbalance is P_4. P_1 is lightly loaded and can take another 2 units. However, these two are not neighbours in the logical network. P_2, and P_3 are perfectly balanced.

Starting the load balancing process at P_4, its neighbours are P_2 and P_3 in the logical network. Since both are balanced evenly, it is not possible to achieve load

balancing without disturbing at least one of them temporarily. We consider P_2 as an intermediate node for allowing load changes to ripple through it from P_4 to P_1 and balance the load.

Now, the edge KR between P_4 and P_2 has rule R_{12} on it. Therefore, $L_h = L_4 = 6$, and $R_c = R_{12}, f_l = f_c = f_{12} = 3$ units;

$v_i = R; v_j = K; v_k = Q; L_4 = 6, I_4 = +2, L_2 = 4$ and $I_2 = 0$

$VS_4 = \{P, Q, R\}$

$RS_4 = \{R_{11}, R_{12}\}$

$NRF_4 = \{K(2)\}$ $\qquad$ $NRF_2 = \{B(1), G(1)\}$

$MRB_4 = \{P(3)\}$ $\qquad$ $MRB_2 = \{K(3), K(4)\}$

Since the new imbalance value $|I_4 - f_{12}| <= \tau$, after shifting the edge KR corresponding to R_{12} to P_2, the new P_4 and P_2 will be as shown below.

$RS_4 = \{R_{11}\}$ $\qquad$ $RS_2 = \{R_2, R_5, R_{12}\}$

$VS_4 = \{P, Q\}$ $\qquad$ $VS_2 = \{D, E, F, K, R\}$

$NRF_4 = \{R(2)\}$ $\qquad$ $NRF_2 = \{B(1), G(1), Q(4)\}$

$MRB_4 = \{Q(2), P(3)\}$ $\qquad$ $MRB_2 = \{K(3), R(4)\}$

$L_4 = 3$ $\qquad$ $L_2 = 7$

$I_4 = +1$ $\qquad$ $I_2 = +3$

The new load status of the agents is as shown below.

Agents	A_1	A_2	A_3	A_4
Desired Loads	8	4	8	4
Actual Loads	6	7	8	3
Imbalance	-2	+3	0	-1

The node with the new heaviest imbalance 3, is P_2. P_4 and P_1 are lightly loaded with imbalances of -1 and -2 respectively. Since the lightly loaded neighbour with the closest imbalance is P_1, P_1 is considered for the load transfer.

Now, the edge BF (labelled 6) between P_1 and P_2 has rule R_2 on it.

Therefore, the heavily loaded agent is $A_h = P_2$; The accepting agent A_a is P_1. $R_c = R_2$, $fc = f_2 = 2$ units;

The set of vertices (each is a v_i) pertaining to all edges belonging to R_2, to be transferred from P_2 to $P_1 = \{D, E, F\}$;

After all these are transferred from P_2 to P_1, $v_j = B$ $v_k = K$, $v_i = F$; $L_1 = 6$, $I_1 = -2$, $L_2 = 7$ and $I_2 = +3$.

Since $|(I_2 — f_2)| <=$ T, we allow the transfer of rule and some vertices.
Before the transfer of the rule R_2, subsets P_1 and P_2 are as shown below:

$RS_1 = \{R_1, R_3 R_4, R_6\}$ $\quad$ $RS_2 = \{R_2, R_5, R_{12}\}$

$VS_1 = \{A, B, C, G, H, I, J\}$ $\quad$ $VS_2 = \{D, E, F, K, R\}$

$NRF_1 = \{\}$ $\quad$ $NRF_2 = \{B(1), G(1), Q(4)\}$

$MRB_1 = \{G(2), B(2)\}$ $\quad$ $MRB_2 = \{K(3), R(4)\}$

$L_1 = 6$ $\quad$ $L_2 = 7$

$I_1 = -2$ $\quad$ $I_2 = +3$

The new CRS edge is the edge FK labelled 6. All the vertices between edges 6 and 7 in the spanning tree (the unmarked ones here) are transferred from P_2 to P_1. However, D and E do not appear in the NRFs and MRBs as they are not present on the new edge FK.

$VS_1 = \{A, B, C, D, E, F, G, H, I, J\}$ $VS_2 = \{K, R\}$

$RS_1 = \{R_1, R_2, R_3, R_4, R_6\}$ $RS_2 = \{R_5, R_{12}\}$

$NRF_1 = \{\}$ $NRF_2 = \{F(1), G(1), Q(4)$

$MRB_1 = \{G(2), F(2)\}$ $MRB_2 = \{K(3), R(4)\}$

$L_1 = 8$ $L_2 = 5$

$I_1 = 0$ $I_2 = +1$

The new load status of the agents is as shown below:

Agents	A_1	A_2	A_3	A_4
Desired Loads	8	4	8	4
Actual Loads	8	5	8	3
Imbalance	0	+1	0	-1

We stop at this stage as the magnitude of imbalance with each agent is equal to 1 here.

The resulting partitioning of the knowledge graph is shown in figure 5.10

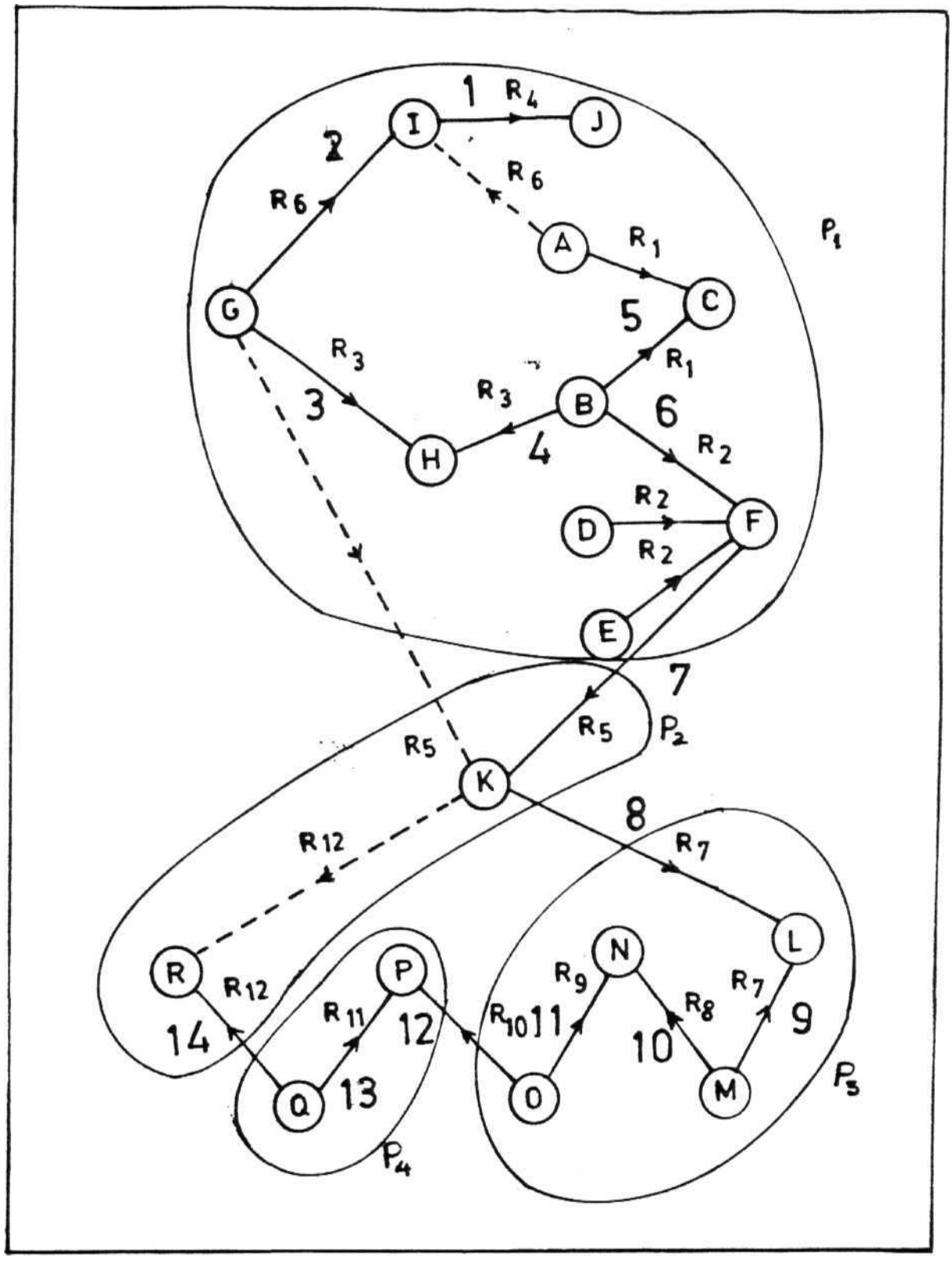

Figure 5.10: A 2:1:2:1 Dynamic Partitioning of Rulebase 2 using Local Changes

P_1	P_2
$VS_1 = \{A, B, C, D, E, F, G, H, I, J\}$ $RS_1 = \{R_1, R_2, R_3, R_4, R_6\}$ $NRF_1 = \{\}$ $MRB_1 = \{G(2), F(2)\}$ $L_1 = 8$ $I_1 = 0$	$VS_2 = \{K, R\}$ $RS_2 = \{R_5, R_{12}\}$ $NRF_2 = \{F(1), G(1), Q(4)\}$ $MRB_2 = \{K(3), R(4)\}$ $L_2 = 5$ $I_2 = +1$
P_3	P_4
$RS_3 = \{$ R7,R8,R9,R10$\}$ $VS_3 = \{$ L,M,N,O$\}$ $NRF_3 = \{$ K(2),P(4)$\}$ $MRB_3 = \{\}$ $L_3 = 8$ $I_4 = 0$	$VS_4 = \{P, Q\}$ $RS_4 = \{R_{11}\}$ $NRF_4 = \{R(2)\}$ $MRB_4 = \{Q(2), P(3)\}$ $L_4 = 3$ $I_4 = +1$

Table 5.10: A 2:1:2:1 Dynamic Partitioning of Rulebase 2 using Local Changes

Final Partition:

The resulting partition is shown in table 5.10.

5.2.3 Repartitioning and Reallocation of the Entire Knowledge Graph

The static partitioning heuristic discussed in chapter 3 can be modified to partition the rules based on the load by considering rule firing frequencies. Changes are required for some of the procedures as described below.

Instead of just counting on the number of rules, we should sum the firing frequencies (multiplied with average rule firing time) of rules. Mainly the procedures dealing with determination of the sizes of the subsets and checking whether balanced partitioning is obtained need some changes for dynamic repartitioning.

procedure *determine_size_of_subset()*;

begin

for i := 1 to k do

Desired load of ith rulebase subset

$z_i = \sum_{j=1}^{n} L_j * \frac{p_i}{\sum_{j=1}^{k} p_j}$;

endfor;

end;

procedure *check_if_balance_possible(bal_possible)*;

(* check if required partitioning is possible *)

begin

if $\sum_{j=1}^{n} L_j mod \sum_{j=1}^{k} p_j = 0$ then bal_possible = true

else bal_possible = false;

endif;

return(bal_possible);

end;

procedure check_if_balance_obtained(balance_obtained):

(* Check if required partitioning is obtained *)

begin

balance_obtained = false;

for i := 1 to k do (* for each subset *)

$L_i = (t_m + t_f) * \sum_{R_j \in RS_i} f_j$;

if ($|I_i| <= \tau$) then put the subset in the okay list;

```
        endfor;
        if (size of okay list < k) and (bal_possible = true) then
                balance_obtained = false;
        endif;
end;
```

The following changes are to be incorporated in step 4 of the static partitioning algorithm discussed in chapter 3 (sections 3.3.6 and 3.5).

```
4   (* partitioning *)
    (a) determine_edges_to_be_cut(); (* initial decomposition *)
    (b) balance_obtained = false;
        iter := 0;
    (c) repeat
        i   find_data_and_proposed_rule_sets(); (* initial decomposition *)
        ii  find_proposed_and_cutset_rules(); (* initial decomposition *)
        iii assign_rules(); (* boundary refinement *)
        iv  check_if_balance_obtained(balance_obtained);
            if (balance_obtained = false) and (iter < maxiter) and (bal_possible = true) then
            (* maxiter is a constant defined by the user *)
                call dyndist1;
                iter := iter + 1;
            endif;
        until (balance_obtained = true) or (bal_possible = false) or (iter > maxiter)
```

Example

Considering the same rulebase with 12 rules in the previous section, let us obtain a partition with the rulebase subset loads in the ratio 2:1:2:1.

Desired loads of the subsets, $D_1 = 8$, $D_2 = 4$, $D_3 = 8$, and $D_4 = 4$.

Edges to be cut are $e_1 = 6$, $e_2 = 8$, and $e_3 = 12$.

Cutting the graph at the edges labelled 6, 8, and 12, we get the conflict rule set, vertex sets, proposed rule sets and nonconflict rule sets as below.

The Conflict Rule Set $CRS = \{R_2, R_5, R_7, R_{10}, R_{12}\}$.

$VS_1 = \{A, B, C, G, H, I, J\}$

$PRS_1 = \{R_1, R_3, R_4, R_6\}$

$RS_1 = \{R_1, R_3, R_4, R_6\}$

$L_1 = 6$

$VS_2 = \{F, K\}$

$PRS_2 = \{R_2, R_5\}$

$RS_2 = \{\}$

$L_2 = 0$

$VS_3 = \{L, M, N, O\}$

$PRS_3 = \{R_7, R_8, R_9, R_{10}\}$

$RS_3 = \{R_8, R_9\}$

$L_3 = 3$

$VS_4 = \{P, Q, R\}$

$PRS_4 = \{R_{11}, R_{12}\}$

$RS_4 = \{R_{11}\}$

$L_4 = 3$

The rule R_2 on the edge between P_1 and P_2 has a firing frequency $f_2 = 2$. Since P_1 needs some rules with a load of 2 units for its desired load $D_1 = 8$, R_2 may be assigned to P_1. Shifting the edges corresponding to R_2, $e_1 = 7$.

The new $VS_1 = \{A, B, C, D, E, F, G, H, I, J\}$ and $VS_2 = \{K\}$. Since $L_1 = 8$ now, P_1 is kept in the okay list.

Since the number of edges in between e_1 and e_2 (excluding both) is zero, e_2 is made 9. This makes $VS_2 = \{K, L\}$, $VS_3 = \{M, N, O\}$ and $CRS = \{R_5, R_7, R_{10}, R_{12}\}$.

Considering R_5 first, it should go to P_1 according to the attribute count corresponding to that rule. However, since P_1 is marked okay, and since P_2 still requires rules whose total load will be less than or equal to 4 units, R_5 (with $f_5 = 2$) units can be assigned to P_2.

R_7 should go to P_2 based on the attribute count, and it can be safely assigned to P_2 as after adding this rule, its load becomes equal to its desired load, i.e., $L_2 = D_2 = 4$. P_2 is marked okay and is kept in the *okay* list.

The new $CRS = \{R_{10}, R_{12}\}$.

Considering rule R_{10}, it should go to any of P_3 and P_4. Since P_3 requires some more rules for its share, assigning R_{10} to P_3 will bring its load L_3 closer to D_3. Therefore, assigning the rule to P_3, $RS_3 = \{R_8, R_9, R_{10}\}$; $L_3 = 6$. However, it still needs 2 units for its desired load.

With the CRS — $\{R_{12}\}$, $R\backslash 7$ may be assigned to any of P_4 and P_2 based on the attribute count. However, since P_2 is marked okay, it should be assigned only to P_4. Now, $RS_4 = \{R_{11}, R_{12}\}$, and $L_4 = 6$ with an excessive load of 2 units making it a heavily loaded node.

Using step 3(d) of the heuristic *dyndist1*, the load imbalance is adjusted such that R_{11} goes to P_3 making $RS_3 = \{R_8, R_9, R_{10}, R_{11}\} L_3 = 9, RS_4 = \{R_{12}\}, and L_4$ — 3. Since the individual imbalances are below 1, with a total imbalance of 2 units, the algorithm stops.

Final Partition

The resulting partition of the knowledge graph is shown in figure 5.11 and table **5.11.**

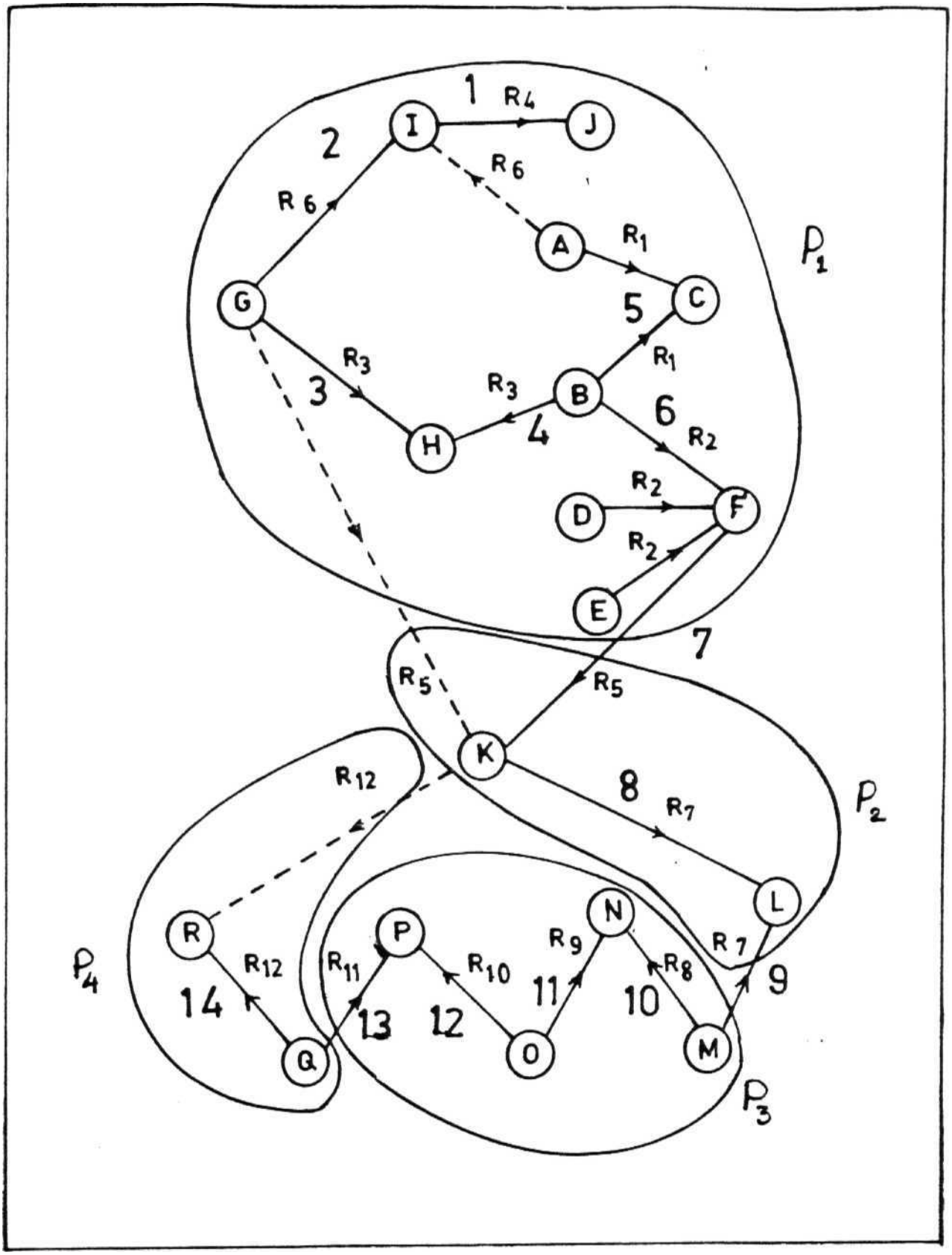

Figure 5.11: A 2:1:2:1 Dynamic Repartitioning of Rulebase 2

P_1	P_2
$VS_1 = \{A, B, C, D, E, F, G, H, I, J\}$ $RS_1 = \{R_1, R_2, R_3, R_4, R_6\}$ $NRF_1 = \{\}$ $MRB_1 = \{(F,2), (G,2)\}$ $L_1 = 8$	$VS_2 = \{K, L\}$ $RS_2 = \{R_5, R_7\}$ $NRF_2 = \{(F,1), (G,1), (M,3)\}$ $MRB_2 = \{(K,3), (K,4)\}$ $L_2 = 4$
P_3	P_4
$VS_3 = \{M, N, O, P\}$ $RS_3 = \{R_8, R_9, R_{10}, R_{11}\}$ $NRF_3 = \{(K,2), (Q,4)\}$ $MRB_3 = \{(M,2)\}$ $L_3 = 9$	$VS_4 = \{Q, R\}$ $RS_4 = \{R_{12}\}$ $NRF_4 = \{(K,2)\}$ $MRB_4 = \{(Q,3)\}$ $L_4 = 3$

Table 5.11: A 2:1:2:1 Dynamic Repartitioning of Rulebase 2

5.2.4 Adaptive Reorganization for accommodating Changes to Knowledge Base

As mentioned in the first section of this chapter, when the knowledge base undergoes changes in the form of additions and deletions of rules due to self learning capability or acquisition of new knowledge from the external world, say from knowledge engineer, adaptive reallocation (reorganization) becomes necessary. This can be done by adding a new rule to a subset which has the maximum number of attributes corresponding to its premise and action parts. If the addition results in an imbalance, and if the imbalance exceeds a threshold, either of local reorganization or repartitioning heuristics can be used to balance the load.

Similarly, if some rule is deleted from a part, reorganization or redistribution may be done afresh. If necessary, a copy of the rule may be stored in the back up for historical or statistical purposes.

The metaknowledge directories of the corresponding agents have to be updated for reasoning and further dynamic distribution.

The steps are given in the following algorithm.

Algorithm *dynreorg*

1. Define a time window and a global threshold
2. Within each window,
   a. if addition of a rule then
      choose a subset with maximum attribute count corresponding to that rule;
      assign the rule to that subset if threshold is not exceeded;
   b. if deletion of a rule,
      delete it from the corresponding part;
      store it in the backup copy for future use;
      if the size falls below a certain minimum
      and if there is a rule which belongs more closely to this subset
      bring and add it to this part;
   c. update the directories of corresponding agents;
      propagate the changes to the user agents;
3. At the end of window,
   perform repartitioning and reallocation;
   using algorithms described in section 5.2.3;
4. Check the number of changes in the previous intervals;
      if the number of updates are increasing over a few intervals then
         set the window size shorter
      else
         if the variance is decreasing and changes are few in number then
            set the window longer;
         endif;
      endif;
5. Start the new window and go to step 2.

5.2.5 Local Reorganization using Active and passive sets

Another way of doing the allocation is by defining active and passive sets of the rulebase. An agent may or **maynot** have the entire knowledge base with it. However, to reduce the pattern matching time and to achieve parallelism, a partition may be made. Each agent can then have its active set as the subset assigned to it and the remaining portion of the rulebase as its passive set. However, as the probelm solving proceeds, if the partitioning proves to be either imbalanced, or inefficient with respect to communication we may make a few modifications to the active and passive sets. Depending on the number of requests for data items from other agents, if the other agent does not seem to be using the data generated by the corresponding rule, the rule can be shifted to a requesting agent. The steps at an abstract level are listed below.

Algorithm *Active_and_Passive_Sets*

1. make an initial partition and allocation;
   let the subsets assigned to the agents be $P_1, P_2, .., P_k$;
   for i := 1 to k do
      Active set $AS_i = P_i$;
      Passive set $PS_i = P - P_i$;
   endfor;
2. within a time window,
      keep a count of communication requests for a partial results (or data item)
      from an agent A_r to all sending agents A_s;
3. if the agent A_s generating the data item has a smaller utilization count then
      assign the rule generating the data item
      to a user agent with highest utilization count provided
      the communication due to this does not increase
      get the rule from the passive set and include it in the active set;
   endif;

The same policy may be used for dynamic relocation of rules.

5.3 Problem Based Knowledge Distribution

When the knowledge base partitioning is fixed as we did for monitoring applications, the task partitioning involves just matching and firing the rules enabled; task partitioning is implicit. However, in domains like medical diagnosis, a problem based distribution of knowledge and hence dynamic creation of agents is essential. In such applications, depending on the symptoms, we will have to explore the possibility of several diseases. Knowledge can be partitioned statically based on the disease or physical organs. When more than one disease is likely and the answer must be obtained quickly (using forward chaining), simultaneously multiple paths may have to be explored. Based on the confidence factors, if any, final decision may be arrived at. Backward chaining may be used for differential diagnosis.

For example, in acute abdominal pains, if the pain is in the *epigastrium*, RU or LU, the related diseases could be *Acute Cholecystitis, Acute Pancreatitis, Perforated Peptic (Duodenal) Ulcer, Acute Hepatitis, Acute Pyelonephritis* and the like. Considering only the first three for illustration, Acute Pancreatitis (AcPan) and Perforated Peptic Ulcer (PPU) could cause pain in any of the right or left upper quadrants or epigastrium. This may require the doctor to keep all three in mind and simultaneouly explore the possibilities and decide on which should be the actual problem. However, as Acute Cholecystitis (AcCh) has its pain mainly in the right upper quadrant, if the pain is experienced in any other region, Acute Cholecystitis may be given secondary importance. With pain in the right upper quadrant area, all the three have to be simultaneously explored.

In all the three, i.e., PPU, AcCh, AcPan, the symptom *vomiting* may be present in the order of increasing frequency. In the first the frequency may be *absent to few times,* in the second it may be *few to many times* and in the last it may be *multiple and persistent.* Therefore, unless vomiting is absent, all the three have to examined, and only if vomiting is persistent implying AcPan, all the modules corresponding to the three diseases should be kept active. There are many other symptoms which are commonly present in almost all these diseases, but the degree (or level) of frequency may be different. Since patients may not give correct details,

it is difficult to determine the level unless the doctor is experienced and patient is under direct examination.

The features specific to a particular disease help in the exclusion of some possibilities. But, sometimes multiple problems may be simultaneously present. Therefore, simultaneous activation of the related modules is essential. Apart from this, confirmation with data obtained from laboratory tests, X-ray and Sonogram is needed for a correct diagnosis. In case of a wrong diagnosis, recomputation is necessary.

For example, a rule like

if

(location_of_pain = *epigastruim*)

(patient _position = *flat)*

(abdominal_wall_rigidity = *boardlike)*

(previous_history = *ulcer)*

(vomiting_frequency = *nil)*

(peristaltic_ sounds = *absent)*

(dietary. intolerance_to = *cabbage)*

then

(disease *PPU)*

(confidence_factor = 95%)

may indicate Perforated Peptic Ulcer to a large extent. Confirmation with X-ray may indicate 75% *free air* in the *ileus.*

The invocation of modules dynamically depending on the current status of problem solving requires estimation of the probabilities of a module being (the most) relevant. Determination of these probabilities and correct invocation of suitable modules is the task in hand. Confidence factors, and probability and uncertainty theories may be considered for the calculation of probabilites.

For example, the following rules may be used to activate the modules corresponding to the diseases appearing in the conclusion part of the rule.

if

(location_of_pain = *epigastruim/RUQ(0.9))*

(radiation_of_pain = *around to back, angle of scapula, right shoulder)*

then

(disease may **be *AC)***

if

(location-oLpain = *epigastrium/RUQ/LUQ)*

(radiation_of_pain = *diffused)*

then

(disease may be *PPU)*

if

(location_of_pain = *epigastruim)*

(radiation_of_pain = *slow, spreading through back)*

then

(disease may be *AcPan)*

if

(location_of_pain = *chest)*

(sensation_of_pain = *band around chest)*

(radiation_of_pain = *arms, left epigastrium, neck, head)*

then

(disease may be *MI)*

if

(location_of_pain = *RUQ))*

(size_of Jiver = *enlarged*)

(liver_palpability = *yes*)

(liver-tenderness = *yes)*

then

(disease may be *AH)*

Only the portions of the corresponding diseases may be activated by considering the disease name as the final result attribute and including all rules incident on the edges leading to external input attributes (ignoring the edge direction) or using a static functional decomposition obtained in the same way.

However, as the data is processed, rules excluding the possibility of some diseases may get fired. This can inactivate the modules corresponding to the disease whose name has been removed from the working memory.

For example the following rules exclude the possibility of the diseases Myocardial Infarction, Perforated Peptic Ulcer, and Acute Pancreatitis respectively.

if

(ECG = *normal)*

(SGOT_Level = *insignificant)*

then

MI

if

(Degree_of_Shock <> *not profound)*

(abdominal. wall_rigidity = *boardlike*)

then

PPU

if

(Serum_Amylase_Level < *1000 units/litre*)

then

AcPan

After this has been done, checking for all the symptoms, and seeking data if necessary from the user and other modules, correct diagnosis can be arrived at.

5.4 Conclusions

We have discussed various methods for distributing knowledge dynamically far load balancing, problem based allocation, and for adaptive reorganization to accommodate changes to the knowledge base. For load balancing, we have developed several methods like making minimal changes to the knowledge subsets and repartitioning/reallocation of entire knowledge base, adaptive reorganization to accommodate changes to the knowledge base, and the use of active and passive sets of rules. Dynamic invocation of appropriate knowledge subsets depending on the problem being solved is discussed for medical diagnosis domain.

Apart from partitioning programs, data and knowledge, system partitioning, i.e., clustering the nodes, can also improve performance. This simplifies the management of resources and diminishes the overhead in dynamic load balancing as communication between processors belonging to the same cluster is less expensive than those belonging to different clusters [13]. Assuming newly created tasks are initially assigned randomly to clusters and processors, with periodic exchange of load information, tasks can be eventually redirected to another processor of the same cluster. This is a compromise between load balancing and minimizing communication.

One of the major goals of knowledge distribution is minimizing information exchange. While a good distribution of knowledge and data facilitates task decomposition, allocation and problem solving, actual problem solving in CPS systems involves reasoning by multiple agents. Since task sharing is made implicit by the distribution of knowledge, agents need to cooperatively exchange results of local problem solving. Reasoning for seeking information from other agents is discussed in the next chapter.

Chapter 6

Distributed Reasoning with Incomplete Information

In the previous chapters we have seen how knowledge can be partitioned and allocated to different agents. This reduces the communication necessary for information exchange between agents. However, if they cannot proceed with local problem solving during the actual problem solving, it is important for the agents to reason about when they should seek nonlocal information and what exactly to request and from whom. In this chapter, we present a distributed reasoning strategy to seek information from other agents for resolving local incompleteness in distributed production systems. The organization of the chapter is as follows. Section 6.1 gives an introduction to the problem and section 6.2 presents a brief review of the related work. Section 6.3 explains our distributed reasoning strategy and section 6.4 discusses the case studies. Finally, the last section presents the summary and conclusions.

6.1 Introduction

As mentioned in chapter 1, DAI systems are often geographically distributed with many natural or temporal dependencies among overlapping subproblems [67, 120]. In general, it is also difficult to decompose the main problem into nonoverlapping subproblems. Agents may not have a complete and correct view of the global situation, and knowledge or information available with an agent may not be adequate to solve all (sub)problems. Therefore, agents need to get information from others by explicit message passing or by accessing the shared memory.

However, one should see that the improved performance due to parallelism is not nullified by the additional communication required or chaos created due to incorrect solution paths. There should be coordination among agents' actions. Agents must know when they should concentrate on local problem solving and when they

should seek information or help from other agents. Since information exchange is influenced by the way data, knowledge and subproblems are distributed, an appropriate and careful distribution certainly reduces communication and increases the efficiency besides having other advantages mentioned in the previous chapters. Further, metaknowledge about the data and knowledge available with other agent;, facilitates distributed reasoning by *focussed addressing.*

Monitoring applications such as real-time aerospace vehicle checkout systems are data driven and require forward chaining as the reasoning mechanism. However, research on distributed reasoning with incomplete information for data driven systems is very little. In this direction, we propose a strategy for an agent to reason with incomplete information in distributed forward chaining systems. It answers three important questions concerned with information exchange — *when should an agent ask others for information, what exactly to ask,* and *whom to ask* [96,100]. Our assumption about incompleteness of a local knowledge/data base resulting from an agent's need for more information is similar to the proposal of Simon and Li [86] and can be resolved by allowing feedback from other agents' knowledge bases.

6.2 Related Work

Though distributed reasoning, particularly with respect to forward chaining, has not been adequately addressed in the literature, lot of work has been done on issues related to communication that may facilitate distributed reasoning. Various architectures, strategies and protocols have been suggested and used [22, 36, 37, 40, 115, 129]. Their main emphasis is on cooperation, control, task distribution, and distributed planning.

DAI communication aides developed by Huhns et al. [64] provide *low level* communication and reasoning primitives necessary for beneficial interaction between heterogeneous expert systems. Their computational agents consist of two parts: a *reasoner* and a *communication aide.* Aides also help to detect deadlocks.

HECODES [7, 131] is another architecture for problem solving by a set of heterogeneous, cooperating expert systems where parallel execution opportunities are exploited. It supports various kinds of cooperation, avoids deadlocks, and provides communication ability.

If individual expert systems use different inexact reasoning models, it is necessary to transform the uncertainties of propositions from one model to another. This involves dealing with aspects like competing hypotheses, cooperation and decision making [76]. Zhang et al.[130, 131] recognize semigroups as algebraic structures of inexact reasoning models and use homomorphic and heterogeneous transformations of these uncertainties. Different solutions are synthesized based on the mean and uniformity of the uncertainty values.

When agents are heterogeneous with respect to the domain knowledge possessed or knowledge representation schemes used by them, cooperation and interagent communication for reasoning about other agents become complex. Weihmayer et al. [121] restrict agent diversity by requiring that agents must possess common semantics of two sorts, viz., knowledge of action effects and knowledge of goal intentions, for dealing with cooperation and communication issues.

If agents that are nonmonotonic reasoners share different view points, exchange beliefs and then make inferences based on the exchanged beliefs, ensuring knowledge base integrity is important. Whenever there is a change made to a justification in one agent, consistency must be ensured among the beliefs in different agents. Huhns et al.[63] propose an algorithm for multi agent truth maintenance in this context.

Mazer [91] uses *temporal* and *epistemo logics* to examine (as external observers and designers) and characterize knowledge and its evolution among interacting agents in these systems. Reasoning about knowledge helps to understand the role of communication in achieving coherence and coordination: coherence and coordination are achieved when certain knowledge states (propositions about one set of agents to be known by other agents) are communicated.

Singh et al.[113] propose a declarative representation scheme based on *temporal logic* for specifying the acting, perceiving, communicating and reasoning abilities of agents. It specifies different kinds of protocols viz., *command, information, request, permission, prohibition* and *explanation protocols* in terms of constraints among agents for communicating at the problem solving level.

Campbell et al.[14] discuss knowledge interchange protocols to keep the knowledge interchange under control by structuring the possible acts of communication in

advance. It is similar to the way communication and actions are structured in formalized public activities like traditional ceremonies, behaviour by diplomats etc. Their *tones* of communication include many of the types discussed in [63, 64, 113, 125]; eg., *action requesting, information seeking*, and *warning.*

Woo et al. [126] propose an architecture, MOAP, for supporting knowledge communication in information systems. Three forms of knowledge communication, *knowledge acquisition* (receiving knowledge from another agent), *knowledge dissemination* (making knowledge available to another agent) and *knowledge transformation* (changing knowledge of one type into another type) are discussed. Similarly communication *within an agent itself among agents within the same organization, among agents in different organizations,* and *between users and agents* are discussed.

Bulletin board model, proposed by Lun et al. [88], combines attributes of integrative systems and blackboard models emphasizing on real-time *dialogue* and *interaction. Agent dialogue* refers to the general information transformation which does not require immediate action and leads to augment knowledge of other agents for improving the performance. *Agent interaction* is meant for imperative information that results in the form of commitment to action by senders and receivers. Both public and private communications are supported for heterogeneous agents that use different types of knowledge representation.

COSMO [125] is a general scheme for communication in cooperative knowledge based systems to notify and query agents, for executing and discarding messages. It uses organizational roles and the past performance to calculate the utility values of agents in order to query for information. It gives a definition of communicative acts and the resulting set of communication strategies and protocols. It is a general scheme in that, communication comprises of all types of action and response messages dealing with inquiries, informing, and complaints.

Genesereth et al.[53],in their agent-based approach to software inter-operation (ability of programs that can exchange information and other services with other programs or other software products and thereby solve problems that cannot be solved alone), developed an *Agent Communication Language (ACL)* for exchange of knowledge (information) and other services among agents. The ACL consists of three parts, viz., vocabulary, an inner language called KIF (Knowledge Interchange Format) and an outer language called KQML (Knowledge Query and Manipulation

Language).

In Distributed Knowledge Model(DKM) [86], agents are organized as a hierarchy with possible lateral connections among agents in different subtrees. Their knowledge in the form of Prolog predicates is classified as *local, group* and *global* Knowledge is distributed, not duplicated, and not shared among agents. Instead, inference is distributed to agents with the required knowledge. Their inference procedure assures that knowledge incompleteness problem is propagated from parent agent to grand parent agent and so on, if it cannot be resolved within a subtree. A similar scheme PARTHENON is proposed by Bose et al.[10] for parallel theorem proving.

DARES [67] experiments with various types of knowledge distribution allowing duplication and dynamic additions to the knowledge subsets. It is assumed that agents do not know whom to ask for the knowledge. If proof advancement is uncertain and the number of predicate symbols is nondecreasing in successive levels of resolution, a knowledge importation request is made by the agent. This is done by first calculating the possible likelihoods for its clause set and broadcasting requests for clause sets which resolve with the local clause sets of highest likelihood. If these requests fail to import nonlocal knowledge, it relaxes the likelihood constraint and repeats the same process until the agent is either successful in importing the required knowledge or the agent has exhausted its clause set.

However, both DKM and DARES are logic based distributed intelligent backward chaining systems. Some systems like AIDEs, HECODES deal with deadlocks, uncertain information and beliefs of multiple agents. Rest of the work including MOAP, COSMO, Bulletin Board model concentrates on communication, but reasoning for incomplete information, particularly in forward chaining systems, needs more specific strategies. Our work is related to reasoning in distributed forward chaining systems. Its emphasis is on how to determine what information is required, and whom to ask when an agent in a distributed intelligent system cannot proceed with the reasoning thread. It is general enough to take care of any type of knowledge distribution. Metaknowledge is used to reduce communication for information exchange.

6.3 Reasoning in Distributed Production Systems

Before discussing about reasoning with incomplete information in distributed production systems, we shall briefly explain the reasoning process in production systems.

6.3.1 Reasoning in Production Systems

A (forward chaining) production system [70, 71] is defined by a set of rules or productions called production memory (PM) together with an assertion data base called working memory (WM) that contains a set of *working memory elements*(WMEs). Each rule comprises of a conjunction of *condition elements* called the *left-hand side* (LHS) of the rule, and a set of *actions* called the *right-hand side* (RHS). The LHS and the RHS are also called as *premise* and *conclusion* parts of the rule. Conditional elements consist of attribute, operator and value sets. The value of an attribute can be either constant or variable. Positive condition elements are satisfied when a matching WME exists, and negative condition elements are satisfied when no matching WME is found. The RHS specifies assertions to be added to or deleted from the WM. WMEs consist of attribute value pairs(AVP's).

In a conventional (single agent) production system, an inference cycle consists of *match, select*, and *act* phases. In the *match* phase, the set of rules for which LHS parts match the current environment of WM is computed. As 90% of the total computation time may be consumed in matching patterns, several algorithms like RETE [44] and TREAT [93] are used to speed up pattern matching. Further, as the number of working memory elements increases, efficiency of the production system decreases due to cost of join operations to be performed in the match process. Ishida [68] optimizes the total cost of join operations by using statistics measured from earlier runs of the program and optimizing the sharing of join operations. Acharya et al. [1] partition and distribute hash tables of working memory elements. In this context, the rulebase partitioning along with working memory distribution discussed in this thesis also improves the performance of the system by reducing the pattern matching time. This is achieved by reducing the search space with smaller **rulebase subsets.**

A rule is said to be *enabled* when all its condition elements match with the working memory contents. (A set of WME's that satisfy the positive condition elements is called as an instantiation of that rule.) If there are only a few conditional elements of a rule matching with WME's then, that rule is said to be *partially matched.* Hence, a rule with a partial match may get enabled in the course of execution if either external input data or previous rule firings cause suitable changes to the WM.

If a single rule is enabled, the rule is fired in the *act* phase by performing additions and deletions on the WM as specified by the RHS of the selected rule. If there are many rules matched, in a single rule firing strategy, the *select* phase chooses exactly one of the matching instantiations of the rules using some predefined criterion. However, in order to fire multiple rules, interference among the rule instantiations must be checked. Interference exists among rule instantiations when the result of parallel execution of the rules is different from the result of sequential executions applied in any order. This analysis can be done using dependency graphs either at compile time or at run time [69, 70]. Rules which do not interfere may then be fired concurrently. However, firing compatible rules without taking the problem solving strategy into consideration can easily result in incorrect solutions (convergence problem). This needs to consider the rule dependencies and the context, i.e, the conditions under which the conflict resolution can be eliminated [80].

In a distributed production system, if two interfering rules are distributed to different agents, the agents must also synchronize their actions to prevent the rules from being fired in parallel and thus maintain consistency [71].

However, multiple rule firing within an agent is part of local inferencing, and synchronization of actions of different agents is concerned with parallel firing of rules in different agents. The distributed reasoning and thus the information exchange, however, are independent of the rule firing strategy used within agents. Hence, we do not discuss these aspects hereafter.

6.3.2 Reasoning with incomplete information in Distributed Production Systems

In real-time expert systems, in order to meet the deadlines, faster processing is required. For example, in an aerospace vehicle checkout application [104], health of the system has to be continuously monitored before checkout. This requires the system to fire as many rules as possible and maximize the performance. Throughput of such a system can be measured as the number of rules fired in unit time. Even in a medical diagnosis application, in case of acute pains, it is essential to arrive at the correct diagnosis fast and save life.

In a single agent system, as long as there are some enabled rules, the agent will be busy firing them. If there are no enabled rules, then the system might have either completed the given task and is waiting for another, or is waiting for some data that can be obtained from the user or sensors in the external world. In both the cases, as soon as new data arrives, some rules may get enabled and fired.

In a distributed production system, the situation becomes more complex because other agents may have the required information. The incompleteness of local information can be resolved with information available with other agents.

When to ask?

Knowing when exactly to ask others requires us to differentiate between the successful task completion and lack of progress due to nonavailability of required information. Therefore, we introduce a new rule type, viz., a *termination rule,* to denote rules that result in (successful) completion of a task from the system's point of view. These can be identified by domain experts. If no termination rule is fired and the agent doesn't have any enabled rules, it is necessary to identify the information useful in enabling the partially matched rules. Assuming that acquisition of data from the local user or sensor is automatic (as part of local inferencing), we consider only the distributed reasoning part to seek partial results and data from other agents. Thus, the inference cycle in a distributed production system environment has steps 1 and 3 extra as shown in the steps below.

1. If a termination rule is fired then
   report the result to the sender of the task
   (* user or another agent *)
   exit (* current task is completed *)
   endif
2. Match
3. If there are no enabled rules then
   seek information from other agents
4. Select
5. Act

Step 3 is required for distributed reasoning, i.e., deciding on when to ask, what exactly to ask and whom to ask.

What to ask?

When there are no enabled rules, the reason could be the nonavailability of the required data or partial results with the agent. The information to be sought depends on two aspects:

- which rules can be fired by requesting as less information as possible?
- what information can increase the probability of firing a large number of rules, and how easily can it be obtained?

Firstly, among the partially matched rules, fewer the unmatched Attribute Operator Value Sets (AOVS's), higher the likelihood of that rule getting enabled.

Definition 1

Let pn be the number of attributes participating in the premise part of a rule *R* and mn the number of matched attributes in it at the time of observation in a production

cycle.

Then, *likelihood* l of the rule R getting enabled in the next production cycle is defined as follows:

$$l = 1 \text{ if } mn = pn \ ;$$
$$l = \frac{mn}{pn} \text{ if } mn < pn.$$

A rule with likelihood 1 is enabled. Likelihood value of a rule can increase or decrease in the course of execution. Requesting for values of attributes, which do not have matching value in AOVS belonging to a rule with a high likelihood results in greater chances of that rule being fired.

Secondly, an attribute participating in many partially matched rules can be considered important and its chances of increasing likelihoods of a large number of rules is high. Usually, it also has a high probability of being obtained excepting a few cases where it may be very costly to get, eg., the result of a costly test or exceptional cases in medical diagnosis, or when it is involved in rules meant for exception handling.

Thus, obtaining values for this attribute results in increasing the likelihood values of a large number of partially matched rules of which some may even get enabled.

If some data that is local to the agent is obtained, some rules may get enabled and fired in the next cycle. The inference process continues like this.

However, there are a few issues applicable to domains like medical diagnosis to be considered in this context.

- Some data may have a probability of occurrence. Whether the data has a high probability or low probability of occurrence, once the data is obtained, the confidence factor(CF) of the possibility of the disease may become high.

  For example, in the diagnosis of Acute Cholecystitis, X-ray may be positive for calculus in gal bladder as most of these stones are opaque to X-ray. However, if present (only in very rare cases), gal bladder calculus confirms Acute Cholecystitis. The rules with high confidence factors (CFs) may be given high priority as they are more important.

- Some data is important for diagnosis and has to be supplied by the user when asked by the system. This data may be easily obtained without involving much cost. However, some data may not be easily available. Even if it is available, it may be costly.

  For example, the presence of shock and prostration (sp, pp) or jaundice *(j)* in *Acute Cholecystitis* may be seen only in a very few cases. Such data is not very important for the diagnosis. Similarly, The onset of pain *(po)* may not be clearly noted by the patient and hence can't always be obtained with accuracy by the doctor; it may be even ignored in the diagnosis. In contrast, data about the location or radiation of pain (pl, pr) is important for the diagnosis, and it can be easily obtained also.

- Among the rules with same likelihood value, or in general, some rules may have high priority of being fired. The data required for such rules may have to be requested from other agents even over-riding the likelihoods of the other partially matched rules.

The uncertainty associated with the incompleteness of local data (which can be obtained from the external world locally on request) may be resolved using *Bayesian Probability theory, Certainty theory* [45], and the other uncertainty management techniques discussed in [76, 130]. Multi agent truth maintenance [63] is also very important because the changes in the beliefs of an agent must be handled carefully. However, we concentrate here on information that may be obtained from other agents.

Definition 2

Let RS_l be the set of partially matched rules with likelihood /;
Dynamic count dc of an attribute *a* is the number of rules in RS_l (computed most recently) in which a appears in the premise (LHS) part.

Hence, requesting for an attribute with highest dc in a rule set with highest likelihood increases the probability of getting the required information and in enabling the rules.

Whom to ask?

This may be determined based on the number of satisfactory responses from other agents. A *satisfactory response* to a query (request) for the value of an attribute *a* is the response with a value for *a*.

Definition 3

Utility u_j of an agent *Aj* from another agent A_i's point of view for a particular attribute *a* is the ratio of number of satisfactory responses (those with values for the requested attribute) n_r from *Aj* to the number of queries n_q from A_i regarding values of *a* over a period of time.

$$u_j = \frac{n_r}{n_q} \tag{6.1}$$

When the problem solving just begins, since no queries and responses (information exchanges) would have taken place, utility values cannot be computed using the above formula. Therefore, an initial estimate can be obtained using the number of rules in A_j having *a* in the conclusion part and the premise parts respectively, and their (expected) rule firing frequencies. Suitable weights may be assigned to rules of each type depending on whether *a* is in the conclusion part or premise part.

Let RS_c and RS_p denote the sets of rules in *Aj* having *a* in their conclusion parts and premise parts respectively. Let w_c be the weight assigned to a rule R_i if $R_i \in RS_c$. Similarly, let w_p be the weight assigned to a rule R_i if $R_i \in RS_p$. Let f_i be the firing frequency of the rule R_i. Now the initial estimate of the utility u_j of an agent A_j can be calculated as

$$u_j = w_c * \sum_{R_i \in RS_c}^{i} f_i + w_p * \sum_{R_k \in RS_p}^{k} f_k \tag{6.2}$$

where rule firing frequencies are summed over all rules belonging to that category, and $w_c \gg w_p$, say $w_c = 0.9$ and $w_p = 0.1$.

Both (6.1) and (6.2) can even be combined to calculate the utility if it proves to be beneficial depending on the situation and application.

6.3.3 Algorithm

We now present our strategy for distributed reasoning in forward chaining systems.

Strategy DRFCS

1. For each partially matched rule *R*,
   compute the likelihood l
   add the rule to the set of rules RS_l with likelihood /;
   endfor;
2. Sort the rule sets RS_l on decreasing order of /;
   Let $lmax$ be the highest likelihood value computed;
   Let RS_{lmax} be the set of rules with likelihood $lmax$
3. Repeat (*For all rule sets with $lmax$*),
   3.1 for each unmatched attribute *a* present in the ruleset RS_{lmax}
   (* rules with maximum likelihood *)
   dc = number of rules in RS_{lmax} in which *a* is present in premise part;
   add a to the attribute set AS_{dc} (all attributes with the same *de* value);
   endfor;
   3.2 sort the attribute sets AS_{dc} on decreasing order of *de;*
   let $dmax$ be the highest *de* value;
   3.3 if there is a high priority rule then
   add all its unmatched attributes to an attribute list of that priority;
   endif;
   3.4 repeat
   3.4.1
   repeat (* for each attribute *a* in AS_{dmax} *)
   if there is a high priority attribute then
   a is a high priority attribute

else *a* is the first attribute in AS_{dmax};

endif;

3.4.1.1

if a $\in VS_i$ and has its *indegree* = 0 then

request locally; (* local inference *)

3.4.1.2 else if $\exists$ some *j* and an entry $(a,j) \in NRF_i$ or $(a,j) \in MRB_i$

(a* can be obtained from another agent *Aj,* second possibility being rare *)

let *umax* be the highest value of utilities of all such agents A_j that can give a

repeat

send requests to the agents with utility *umax;*

if an answer with a value for *a* is recieved then

update utility values of agents;

if an answer with a matching value is received then

update likelihoods of all rules which will get

affected by the new value of a as in step 1;

if any rule is enabled then

add it to the enabled rule list;

exit; (* execute the enabled rules *)

endif;

endif

else *umax* = next highest utility u

endif;

until an answer is received or all agents report failure;

(* to send values for a *)

endif;

endif;

3.4.1.3 a is the next unconsidered attribute in AS_{dmax}

until all attributes in AS_{dmax} are over

3.4.2 $dmax$ = next highest dc

until all dc values corresponding to RS_{lmax} are exhausted;

(* for all attribute sets of RS_{lmax} are over *)

3.5 $lmax$ = next highest /

until all likelihood values / are exhausted;

(* for all rulesets*)

The distributed inferencing strategy DRFCS is invoked by the local inferencing cycle, when some nonlocal data, possibly available with other agents, is required. Likelihood computation for all rules is done only the first time the distributed inferencing step is invoked while solving a particular problem. Later on, likelihood computation is required only for the rules that get affected by the changes to the working memory either through local user or sensor, or other agents.

When likelihood of a rule changes, it is automatically deleted from the list of partially matched rules previously it belonged to, and is added to the corresponding partially matched rule set with the new likelihood value. Dynamic count values for attributes are computed only when the corresponding rule set is being considered for getting data and enabling rules. In fact, even if *a* is present in a lower likelihood rule set, it will not be taken into account for computing dynamic count then. There is no need to compute it as soon as the likelihood of the rule (in which *a* is present) changes.

Once value of an attribute a is obtained, likelihood values of rules using this AOV set will be updated. It must be noted that the requests are sent in the decreasing order of likelihoods, but a suitable value obtained can update likelihood of rules in other rule sets irrespective of their likelihood. A value for a received previously while trying with a higher likelihood rule set does not eliminate the need for requesting it with a rule set of lower likelihood as old values might have been updated. Hence, *a* is still considered without discarding.

Also, in our strategy, requests for attribute values are sequentially processed in the order of dynamic count values of attributes and utility values agents. This can be parallelized by having all attributes with same dynamic count value requested at

once from the respective highest utility agents. Upon failure to get values for some attributes from highest utility agents, requests for these can be sent to next highest utility agents along with requests for attributes with lower dynamic count addressed to their highest utility agents.

In the absence of this knowledge about donors, i.e., NRFs and MRBs, and the utility values of agents, requests would have to be broadcast and sender has to wait till some useful information is obtained. Instead, by maintaining the metaknowledge in the form of NRFs and MRBs along with utility values, information incompleteness is resolved by sending the requests in a directed fashion. The distribution of knowledge and data with minimum dependencies using the partitioning and allocation heuristics of chapter 3 reduces communication required for information exchange and thus facilitates reasoning.

6.4 Case Studies

We shall explain the working of the distributed reasoning strategy considering the rules for aerospace and medical diagnosis applications.

6.4.1 Aerospace Vehicle Checkout Application

Consider the following partition in the ratio 1:1.

Part P_1
$VS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp, bs_3\}$
$RS_1 = \{R_1, .., R_{13}, R_{24}, .., R_{29}\}$
$NRF_1 = \{(ct_1, 2)\}$
$MRB_1 = \{(bs_3, 2\}$
$DVS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp, bs_3, ct_1\}$
$EVS_1 = \{ap_1, bs_1, n2tp, bv_1, c_1, bs_2, lp_1, n2rp\}$
$\|RS_1\| = 19$

Part P_2
$VS_2 = \{n2itp, ct1, bs_7, cm_1, bv_2, bv_3, bs_4, bs_5, bs_6, bs_8, cm_2, bs_9, bhp\}$
$RS_2 = \{R_{14}, .., R_{23}, R_{30}, .., R_{37}\}$
$MRB_2 = \{(ct_1, 1)\}$
$NRF_2 = \{(bs_3, 1)\}$
$EVS_2 = \{n2itp, bs_7, cm_1, bv_2, bv_3, bs_4, bs_5, bs_6, bs_8, cm_2, bs_9, bhp\}$
$DVS_2 = \{n2itp, ct1, bs_7, cm_1, bv_2, bv_3, bs_4, bs_5, bs_6, bs_3, bs_8, cm_2, bs_9, bhp\}$
$\|RS_2\| = 18$

Let P_1 and P_2 be assigned to agents A_1 to A_2 respectively. There is no need to apply the allocation algorithm as there are only two agents involved. Further we assume that there are no high priority rules and high priority data.

Let us assume a working memory instance of A_1 as shown below.

Working memory instance of A_1 in the beginning
$ap_1 = 30, lp_1 = 3, c_1 = 0.3.$

The inference process in each cycle is explained below.

Cycle 1

Rule fired	Change to the working memory
R_1	bs1 = 'LACPR'
R_3	bs2 = 'NOK'

The list of the rules which are matched = $\{R_1, R_3\}$ and the list of partially matched rules = $\{R_6\}$. Since there are some enabled rules, local inferencing continues and the distributed inferencing algorithm DRFCS doesn't get invoked. Firing these rules results in changes to the working memory.

Working memory instance of A_1 at the end of Cycle 1
$ap_1 = 30, lp_1 = 3, c_1 = 0.3, bs_1 =' LACPR', bs_2 =' NOK'$

These changes initiate a new cycle.

Cycle 2

Rule fired	Change to the working memory
R_1	bs1 = 'LACPR' (no new change)
R_3	bs3 = 'NOK' (no new change)
R_5	Hold (no change to working memory elements' values)

The list of partially matched rules = $\{R_6, R_{24}\}$. Since rules R_1 and R_3 have been fired and do not result in new changes to the working memory, we shall not consider them again. Firing R_5 results in a *hold* condition which does not change the working memory.

Cycle 3

There are no enabled rules, and the DRFCS is invoked from the local inferencing procedure.

Partially matched rule	Likelihood
R_6	0.5
R_{24}	0.5

For rule R_6, attribute required for a full match is bs_2 and the conditional element bs_2 =' OK' should be satisfied. Since bs_2 is a local data item, there is no need for asking for this from others. An external input for $c\backslash$ in the range 0.6 to 1.5 will enable R_7 and its firing will result in changes to the working memory satisfying the above condition. Otherwise, it may be that $bs_2 =' NOK'$ because of some value of bv_1 or c_1.

The rule R_{24} has a likelihood of 0.5 of being fired. The only data item needed for this rule is ct_1. However, ct_1 is nonlocal. There is an entry $(ct_1, 2)$ in NRF_1 list of the agent implying that ct_1 belongs to A_2. Dynamic count of ct_1 is 1. It may be noted that it is present in 14 rules. Since there are only two agents, A_2 is the only highest utility agent to seek value for ct_1. A_2 is actually owning ct_1. Therefore, value of ct_1 has to be requested from A_2.

A request is sent to A_2, and let us say A_1 receives a value of 300 for ct_1. Utility of agent A_2 from the point of view of A_1 will now be increased because of this. Now, the rule R_{24} gets enabled, and with the control transferred to the local inference procedure, the rule is fired resulting in a *hold* again.

Depending on whether some external input is obtained resulting in some changes to local working memory, the next inference cycle may have some enabled rules for firing by A_1. The process continues this way.

6.4.2 Medical Diagnosis Application

Let us consider the 2 : 1 partition (for the rules corresponding to the disease *Acute Cholecystitis)* obtained in section 4.2 of chapter 4. Assuming subsets P_1 and P_2

are assigned to agents A_1 and A_2 respectively, a working memory instance of A_1 is shown below.

	Part P_1
VS_1	{ vp, vf, li, lif, liq, hpc, po, pot, pi, pl, pr, di, abt, abrt, cvt, cvts, ps, pp, xil, xcal, j, prp, sp, AC1, .., AC9, AC11, AC12, AC13, AC16, AC18, AC19,AC}
RS_1	$\{R_1, .., R_9, R_{11}, .., R_{12}, R_{13}, R_{16}, R_{18}, R_{19}, R_{25}, .., R_{29}, R_{31}\}$
$\|RS_1\|$	21
NRF_1	{(AC10,2), (AC14,2), (AC15,2), (AC17,2),(AC20,2)}
MRB_1	{(AC,2), (AC19,2)}
	Part P_2
VS_2	{afp, fc, fr, fs, prt, sgb, esr, lc, sal, ecg, sgotl, sd, abwr, abwg, ms, AC10, AC14, AC15, AC17, AC20, AC21, P̃PU, ÃCPAN, M̃I}
RS_2	$\{R_{10}, R_{14}, R_{15}, R_{17}, R_{20}, .., R_{24}, R_{30}\}$
$\|RS_2\|$	10
NRF_2	{(AC,1), (AC19,1)}
MRB_2	{(AC10,1), (AC14,1), (AC15,1),(AC17,1),(AC20,1)}

Working memory instance of agent having subset P_1
pl = RUQ, pr = around to back and angle of scapula, right shoulder,
pi = maximum, pot = early;
vp = yes, $vf = few$; li = yes; sp = yes;
di = fatty foods;
abt = present, $abrt$ = present;
cvt = present;
xil = true, $xcal$ = absent;
hpc = similar to current episode;

The inference process of agent $A\backslash$ is explained below.

Cycle 1:

The rules fired and the resulting changes to working memory are:

Rule fired	**Change to the WM**
R_2	AC_2 = true
R_5	AC_5 = true
R_7	AC_7 = true
R_8	AC_8 = true
R_{13}	AC_{13} = true

Since *xcal* = present only in 10%, $R_{\cdot 3}$ is enabled and fired. The partially matched rule sets with a likelihood > 0 and their exact likelihoods values are:

Partially matched rules	Likelihood	Attributes required
R_1	0.8	*po*
R_6	0.5	*prp*
R_9	0.5	*cvts*
R_3	0.33	*lif, liq*
R_4	0.25	AC1,AC2,AC3

The next cycle proceeds as below.

Cycle 2:

AC_2 increases likelihood of R_4 to 0.5, and AC_{13} = true enables R_{26} and creates a working memory element AC = true (CF = 0.95). CF = 0.95 indicates the confidence associated with the possibility of Acute Cholecystitis.

The new rules enabled and the resulting changes to working memory arc:

Rule fired	Change to the WM
R_{26}	AC = true (CF = .95)

The new partially matched rule sets with likelihood > 0 and their exact likelihoods values are:

Partially matched rules	Likelihood	Attribute required
R_1	0.8	*po*
R_4	0.5	AC1, AC3
R_3	0.33	lif, liq
R_6	0.5	*prp*
R_9	0.5	*cvts*
R_{31}	0.27	AC6, AC9, AC10, AC11, AC12,AC15,AC16,AC18

Cycle 3:

There are no enabled rules. Therefore, we should determine the data that has to be obtained for enabling some rules. The highest likelihood value $lmax$ = 0.8.

The partially matched rule set with highest likelihood value 0.8 has only one rule, i.e., R_1.

However, the attribute required by it, i.e., *po* has a dynamic count *de* = 1. However, this is a local data item. (Since this is an attribute which is not very important for diagnosis, this may be ignored by the local inference procedure if desired.)

Since all attributes and all dc values corresponding to $RS_{l=0.8}$ are exhausted, we examine the next likelihood rule set.

There are no more attributes in AS_{dmax} and there is no other dc value. The next highest likelihood value $lmax$ = 0.5.

$RS_{lmax} = RS_{l=0.5} = \{R_4, R_6, R_9\}$. The attributes for which values are required are $prp, cvts, AC1, AC3$. All these attributes have the same dc value, i.e., $dc = dmax = 1$. Therefore, $AS_{dmax=1} = \{prp, cvt, AC1, AC3\}$.

Among the attributes in AS_{dmax}, prp is local. (Since it has a probability of 0.1 of being true in the rule. Corresponding to the possibility of appearing in Acute Cholecystitis cases, the rule may be fired by the local inference engine without obtaining it.)

$cvts$ is also local. (Since *cuts* is not very important for the diagnosis, it may be ignored and R_9 may be fired locally.)

AC is local, but it is not an external data item. Similarly, $AC3$ is a local, intermediate result.

Exhausting all the attributes and the *de* values corresponding to $RS_{l=0.5}$, the next highest likelihood value for the partially matched rules, $lmax$ = 0.33.

The only rule in $RS_{l=0.33} = \{R_3\}$. The attributes required *lif,liq* have a dc value of 1.0; $AS_{dmax} = \{lif, liq\}$. Both are local. (R_3 may be fired by the local reasoner if these are ignored.)

The next highest likelihood value $lmax = 0.27$ and the corresponding rule set $RS_{lmax} = RS_{l=0.27} = R_{31}$.
The attributes required are $\{AC6, AC9, AC10, AC12, AC15, AC16, AC18\}$.
The highest and the only dc value = 1.
$AS_{dmax} = \{AC6, AC9, AC10, AC12, AC15, AC16, AC18\}$

AC6, AC9, AC12, AC16 and AC18 are local, intermediate results. AC10 and AC15 are nonlocal. There is an entry (AC10, 2) belonging to NRF_1 indicating that agent A_2 may be having this value. Since NRF_1 entries show that the only agent that can give $AC10$ is A_2, the request is passed on to A_2. Supposing the value obtained for $AC10$ is true, utility value of $AC10$ will be increased. This also increases the likelihood of the rule R_{31}. Since $AC15$ is also a nonlocal data item to be requested from A_2 (there is an entry (AC10,2) in NRF_1), as discussed earlier, request for both the data items may be sent at once to A_2.
Depending on the values received from A_2 and the uncertainty mangement technique used by the local inference procedure, some more rules may get enabled and the resulting changes to working memory may update likelihoods of rules and enable new rules. The diagnosis may be confirmed on firing a few more rules which increase the confidence that the disease is Acute Cholecystis.

6.5 **Conclusions**

Reasoning in distributed forward chaining systems consists of *local inferencing* and *distributed inferencing*. The distributed inferencing step is embedded in the local inferencing, and is invoked only when there are no enabled rules to be fired by an agent. The information to be obtained is decided based on the rule firing likelihood and dynamic count of attributes in the rule set with highest likelihood value. Each agent has an idea of the utilities of other agents for information that is shared and needed **for** local problem solving. Utilities of agents are calculated based on the

number rules having a particular attribute in the conclusion part of the requested agent, their rule firing frequencies and the satisfactory responses to the requests for the same.

The reasoning strategy works irrespective of whether knowledge is distributed statically or dynamically, and with or without duplication. Metaknowledge about information which should be obtained from other agents and the utility values of the concerned agents help in reducing communication required for information exchange by focussing the requests.

Chapter 7

Conclusions

This chapter summarizes the main contributions of the thesis and concludes with a discussion and future directions.

7.1 Summary

Knowledge distribution plays an important role in several aspects of Cooperative Problem Solving. It is shown that an appropriate distribution of knowledge and data leads to a good task decomposition. Minimizing the interdependencies among the knowledge and data subsets reduces the communication required for information exchange. Metaknowledge about the rules and data possessed by an agent, and about the information that would be required by other agents provides a good model of agents and helps in requesting for nonlocal information by focussed addressing. Good knowledge partitioning also achieves load balancing, speedup through faster pattern matching and concurrent processing. We considered rule-based production systems to represent our CPS systems.

However, there are no direct and fast domain independent techniques for heterogeneous k-way partitioning of the knowledge base. Standard graph partitioning techniques like Kernighan-Lin heuristic and Simulated Annealing concentrate on two-way partitioning and are usually oriented towards VLSI circuit design. Moreover, the techniques are computationally expensive. Therefore, fast heuristics that can obtain reasonably good solutions are preferable to their costly counterparts that produce near-optimal solutions. Further, most of the partitioning techniques ignore data distribution, the interdependencies among subsets, and heterogeneous partitioning which are important for Cooperative Problem Solving systems. In this thesis, we proposed new heuristics based on graph theory for distributing knowledge statically. The techniques consider data distribution, heterogeneous partitioning

and interdependencies among rules while partitioning knowledge. Further, these are extended to deal with dynamic distribution of knowledge.

The proposed static partitioning heuristic partitions the rules by representing the rulebase as a knowledge graph where vertices represent data elements and edges are drawn from input attributes to output attributes in each rule. The edge labels represent the corresponding rule identifiers. The partitioning process consists of two phases, viz., initial partitioning and boundary refinement. This is done by generating a spanning tree with a long chain. The chain edges and some branch edges are marked with integer numbers to enable suitable decomposition. Once the rules and data belonging to different subsets are identified approximately, subset boundaries are refined by assigning rules to subsets which possess a major portion of the data corrsponding to the rule. This minimizes the interdependencies and reduces communication. Given the ratio in which the rulebase subsets are to be obtained, the heuristic obtains the same in linear time. The partition obtained is good enough to be used as it is or can serve as a good initial partition for obtaining near-optimal parition using the Kernighan-Lin and Simulated Annealing techniques. As part of the partitioning process, metaknowledge about the data and rules possessed by each agent is abstractred. The metaknowledge also has information about data that may be required by other agents (MRBs) and data that needs to be requested from other agents (NRFs) for efficient reasoning. This also implies some estimation of other agent's requirement for nonlocal information, and helps to define agents and achieves better coordination as well.

Such a partition can improve the performance of the near-optimal paritioning techniques like Simulated Annealing and KL heuristic. This algorithm is further extended to deal with disconnected components in the knowledge graph such that it makes use of the noncommunicating components, if any, in obtaining balanced subsets. Further, ways of obtaining functional decomposition from the knowledge graph are discussed.

In addition to the partitioning heuristics, a method for allocating the subsets obtained in the given proportion (which represents the capacities of agents) to agents is also proposed. Considering the interagent distance, the information transfer between the subsets, and the capacities of agents, knowledge subsets are assigned to compatible agents (agents of corresponding capacity) only. The method guarantees

a minimal communication allocation for the given partition.

Since static partitioning is not sufficient for applications with unpredictable run time requirements, dynamic distribution(partitioning as well as allocation) becomes essential for load balancing. Dynamic distribution is necessary also for accommodating changes to knowledge base and distributing knowledge as the problems arrive at run time. We developed heuristics for distributing knoweldge dynamically. The static partitioning algorithms have been extended to incorporate the run time aspects. These make use of the information about loads on agents during the previous time interval. The length of the time interval can be changed for adaptive reorganization. Loads on the agents are calculated by considering the firing frequencies of rules assigned to them. After the load information about all the agents is obtained, imbalance with respect to the desired load is calculated for each agent. Making use of the MRBs and NRFs, and the rules in the (overlap of) boundaries of subsets, rules are transferred to lightly loaded neighbours. Two main heuristics are proposed for this.

The first heuristic makes minor changes to the present distribution in order to bring the loads closer to the ideal (or desired) loads in the previous intervals. Since the subsets assigned to the agents will have interdependencies among them, it is not possible to arbitrarily transfer the rules from a heavily loaded agent to a lightly loaded agent. Therefore, the rule transfer has to occur between and via the agents (including the heavily loaded and lightly loaded ones) in the logical network. The MRBs and NRFs are updated to reflect the changes. The second heuristic does repartitioning of the entire knowledge graph considering the run time aspects. Methods for catering to updates (additions and deletions) to the knowledge base, and dynamic problem based knowledge distribution are also discussed.

The proposed distributed reasoning strategy makes requests for nonlocal information from potential donors. This is done by selecting the data (with highest dynamic count) required for enabling the partially matched rules having highest likelihood of being fired and requesting from an agent with highest utility. However, if higher priority rules are present, the likelihoods may be ignored temporarily, and data for the high priority rule may be requested first. The information exchange is made easy and decreases the communication required with the NRFs and MRBs of agents. The utilities of agents that reply with useful information are increased.

7.2 Discussion

It is seen that the linear time heuristic works well for various input ratios with several examples considered. Case studies of aerospace vehicle checkout application and medical diagnosis applications represent two different cases of knowledge distribution and provide ample number of cases for tesing various features.

Aerospace application presents itself with multiple rules as edge labels and disconnected components. As this is a monitoring application, all rules have equal probability of being fired and hence are to be tested and fired without any discrimination. Therefore, this proves to be a perfect example where the partitions obtained using our heuristic balanced the load and implied task distribution as well. Partitions are also close to functional decomposition for appropriate input ratios. However, use of coupling and free grouping of rules and nodes as such or in the static partitioning algorithm for load balancing further enhanced the quality of the functional decomposition. However, on allowing duplication, this can exactly become a functional decomposition. There was no uncertainty associated with rules in the reasonig process.

Unlike the aerospace application, in medical diagnosis, lot of data is shared by many subsets corresponding to diseases (or concepts). Therefore, it is important to keep the rules of the closely related diseases at the same place and reduce communication. Partitioning for load balancing within a functional decomposition is better than pure load balancing as this may result in more communication for small subset sizes and inappropriate proportions. Duplication of common knowledge and the associated data reduce the information exchange between subsets. Secondly, since the entire knowledge base is not active at the same time, only some subset, not all, need to be examined *to* diagnose a problem. Dynamic invocation of modules is also important for this type of application. Another difference is the presence of uncertainty involved with the data as well as rules. This may be taken care of by the inferencing procedure.

7.3 Future Directions

The partitioning heuristics consider production rules as the knowledge graph representation scheme. These can be extended to suit other knowledge representation schemes. Similarly, suitability of Petri Nets to represent the knowledge base is to be investigated.

Further, the knowledge graph representation we have used is chosen to facilitate reasoning for incomplete information that is available with other agents, as well as the dynamic distribution of knowledge such that exact information about data and rules involved is known while transferring data and knowledge. The knowledge graph representation may be slightly modified such that rules are nodes in the graph and edges represent the amount of information transfer or dependencies between rules. In such a representation the exact data shared between the rules is not known, but it abstracts the amount of data shared in a simpler way. Performance of the respective algorithms with these two representations can be evaluated.

The dynamic distribution algorithms were simulated with some values for firing frequencies of rules. However, integration of the reasoning process with dynamic distribution of knowledge based on the actual firing of rules is not done because of the time limit. This can give more accurate idea of state of the system at a given time, thus resulting in better load balancing in a practical system. Use of probabilistic models for dynamic knowledge transfer corresponding to the excessive load can also be investigated. Possibility of optimal distribution of knowledge to maximize the number of problems solved locally as in distributed data bases needs to be looked into.

The distributed reasoning algorithm considers a single reasoning model. Dealing with inexact, heterogeneous reasoning models and multiple view points is very important for large, practical systems. Inexact reasoning models like Baye's probability theory and Certainity theory, and multi agent truth maintenance can be appropriately incorporated. Similarly, the distribted reasoning algorithm can be extended to include time factor and the answering agent's perspective also to achieve more efficient coordination.

Appendix A

Rules and Object Structure for Aerospace Application

Thirty seven rules from the rulebase of Aerospace Vehicle Checkout System are given below. Meanings of the attributes and the object structure of the systems involved are given at the end.

R1. **if** ($ap1 < 130$) **then** $bs1$ = 'LACPR'

R2. **if** ($ap1 > 170$) **then** $bs1$ = 'HACPR'

R3. **if** ($c1 < 0.5$) **then** $bs2$ = 'NOK'

R4. **if** ($c1 > 1.5$) **then** $bs2$ = 'NOK'

R5. **if** ($bs2$ = 'NOK') **then** $h1$

R6. **if** ($bs2$ = 'OK') and ($lp1 > 1$) **then** $h2$

R7. **if** ($bs2$ = 'NOK') and ($n2tp < 210$) **then** $h3$

R8. **if** ($bv1 < 25$) **then** $bs2$ = 'NOK'

R9. **if** ($bv1 > 31$) **then** $bs2$ = 'NOK'

R10. **if** ($bs2$ = 'NOK') and ($n2tp < 210$) **then** $h4$

R11. **if** ($bs2$ = 'OK') and ($n2tp < 210$) **then** $h5$

R12. **if** ($n2rp < 10$) **then** $bs3$ = 'LRJPR'

R13. **if** ($n2rp > 50$) **then** $bs3$ = 'HRGPR'

R14. **if** ($n2itp < 10$) **then** $bs3$ = 'LINJPR'

R15. **if** (*n2itp* > 10) **then** *bs3* = 'HINJPR'

R16. **if** (*bhp* < 10) **then** *bs4* = 'LHOLPR'

R17. **if** (*bhp* > 50) **then** *bs4* = 'HHOLPR'

R18. **if** (*bv2* < 25) **then** *bs5* = 'LBV'

R19. **if** (*bv2* > 31) **then** *bs5* = 'OBV'

R20. **if** (*bv3* < 25) **then** *bs6* = 'LBV'

R21. **if** (*bv3* > 31) **then** *bs6* = 'OBV'

R22. **if** (*cm1* <> *bs7*) **then** *bs5* = 'BIES'

R23. **if** (*cm2* <> *bs8*) **then** *bs9* = 'BIES'

R24. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs1* = 'LACPR') **then** *h6*

R25. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs1* = 'HACPR') **then** *h7*

R26. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs3* = 'LRGPR') **then** *h8*

R27. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs3* = 'HRGPR') **then** *h9*

R28. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs3* = 'LINJPR') **then** *h10*

R29. **if** (*ct1* <= 450) and (*ct1* > 150) and (*bs3* = 'HINJPR') **then** *h11*

R30. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs7* = 'EXT') **then** *h12*

R31. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs8* = 'EXT') **then** *h13*

R32. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs5* = 'BIES') **then** *h14*

R33. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs9* = 'BIES') **then** *h15*

R34. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs5* = 'OVB') and (*bs7* = 'INT') **then** *h16*

R35. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs9* = 'OBV') and (*bs8* = 'INT') **then** *h17*

R36. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs8* = 'INT') and (*bs9* = 'LBV') **then** *h18*

R37. **if** (*ct1* <= 150) and (*ct1* > 27) and (*bs5* = 'LBV') and (*bs7* = 'INT') **then** *h19*

Attribute codes and their actual names in the system REX

ap1 = bs.ftc.hyd-system1.acc.pressure

bs1 = bs.ftc.hyd-system1.in-acc.status

c1 = bs.sc.sc-current

bs2 = bs.sc.sc-status

lp1 = bs.ftc.hyd-system1.line-pr

n2tp = bs.sitvc.n2-system.tank-pr

bv1 = bs.sc.ext-supply.voltage

bs3 = bs.sc.status

n2rp = bs.sitvc.n2-system.reg-pr

bs4 = bs.sitvc.n2-system.reg-pr.status

n2itp = bs.sitvc.n2-system.innjectant-tank-pr

bhp = bs.sitvc.hyd oil line-pr

bs5 = bs.sitvc.n2-system.tankpr-status *bs6* = bs.sitvc.tvc-status

bv2 = bs.sc.sc-battery.voltage

bs7 = bs.sc.sc-battery.status

bv3 = bs.sc.cpif-bat.voltage

bs8 = bs.sc.cpif-bat.status

cm1 = bs.sc.sc-battery.intonoff cmd

cm2 = bs.cpif-bat.intonoff cmd

bs9 = bs.sc.sc-battery.intonoff status

ct1 = system.cdt

All of *h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15, h16, h17, h18* and *h19* stand for various **Hold** conditions.

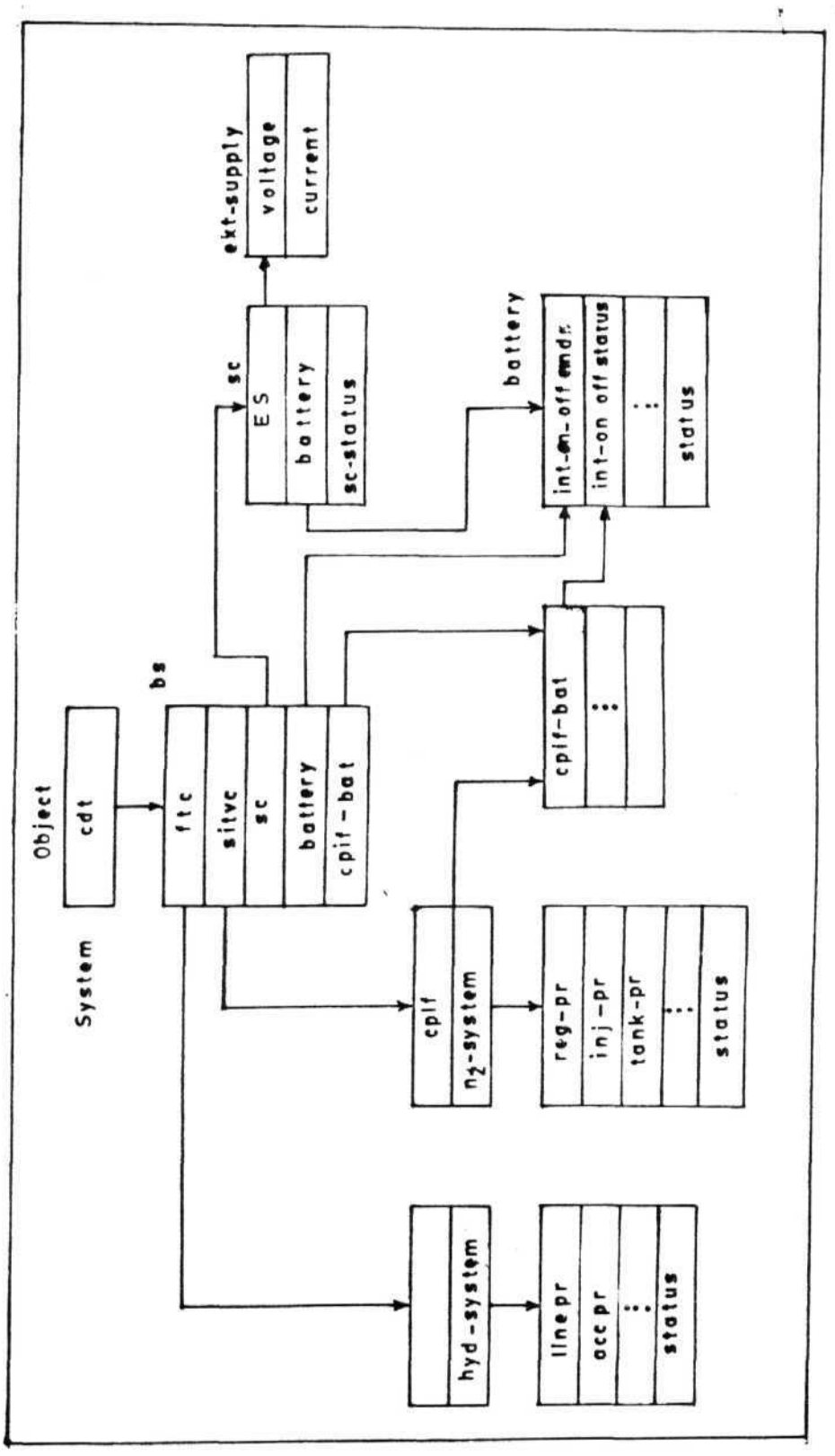

Figure A. 12: Object Structure for the Aeospace System

Appendix B

Rules for Medical Diagnosis Application

Rules corresponding to three disorders viz., Acute Cholecystitis, Acute Pancreatitis and Perforated Peptic Ulcer, pertaining to medical diagnosis of acute abdominal pains are given below. The decodification of the attributes is also given after the rules.

R1. **if** *(po* — sudden or gradual) and *(pot* = early) and *(pi* = maximum) and (pl = RUQ(0.9)) and *(pr* = around to back,angle of scapula, right shoulder) **then** AC1

R2. **if** *(vp* = yes) and *(vf* = few to many) **then** AC2

R3. **if** (*li* = yes) and (lif = occasional) and (*liq* <> heavy) **then** AC3

R4. **if** AC1 and AC2 and AC3 and *(hpc* = similar to current episode) **then** AC4

R5. **if** *(di* = fatty foods and cabbage) **then** AC5

R6. **if** *(sp* = yes(O.l)) and *(prp* = yes(O.l)) **then** AC6

R7. **if** *(abt* = present) **then** AC7

R8. **if** *(abrt* = present) **then** AC8

R9. **if** *(cvt* = present) and *(cvts* = right) **then** AC9

R10. **if** *(abwr* = present) and *(abwg* = unilateral rectus) and *(ms* = present) **then** AC10

R11. **if** *(ps* = normal to hypoactive) **then** AC11

R12. **if** *(pp* = flatsupine) **then** AC12

R13. **if** (*xil* = true) and (*xcal* = present(O.lO)) **then** AC13

R14. if *(sgb* = stones) **then** AC14

R15. *if* (*sal* = minimal elevation) and (*lc* = moderate increase) and (*esr* = increases) **then** AC15

R16. if *(j* = present(0.2)) **then** AC16

R17. if *(afp* = present) and *(fc* — present) and *(fr* = present) and *(fs* = present) and (*prl* = present) **then** AC17

R18. if *(pp* = onside) **then** AC18

R19. if AC1 and AC17 **then** AC19

R20. if ACM and AC19 **then** AC20

R21. if (ecg = normal) and (*sgotl* = insignicant) **then** MI

R22. **if** *(sd* <> profound) and *(abwr* <> boardlike) **then** PPU

R23. **if** *(sal* < 1000 units) **then** AcPan

R24. if AC20 and MI and PPU and AcPan **then** AC21

R25. **if** AC1 **then** AC (CF = 0.80)

R26. **if** AC13 **then** AC (CF = 0.95)

R27. **if** AC14 **then** AC (CF = .95)

R28. **if** AC19 **then** AC (CF = .85)

R29. **if** AC20 **then** AC (CF = 0.97)

R30. **if** AC21 **then** AC (CF = 0.98)

R31. **if** AC5 and AC6 and AC7 and AC8 and AC9 and AC10 and ACll and AC12 and AC15 and AC16 and AC18 **then** AC

R32. **if** *(po* = sudden and sharp) and (pl = epigastrium/RUQ/LUQ) and *(pr* = diffused) **then** PPU1

R33. **if** *(vp* = yes) and *(vf* = absent to few) and (*vn* = retching) **then** PPU2

R34. if (*li* = yes) and (lif = variable) then PPU3

R35. if (*hpu* = yes) then PPU4

R36. if *(di* = spices and alcohol) then PPU5

R37. if (st = early) and *(sd* = common or high) and *(prp* = yes) then PPU6

R38. if *(abt* = yes) and *(abtn* = diffused) then PPU7

R39. if *(abrt* <= 4 hours) then PPU8

R40. if *(cvt* = true) and *(cvts* = bilateral) then PPU9

R41. if (*abwr* = boardlike) then PPUIO

R42. if *(ps* = absent) then PPUll

R43. if *(pp* = flat supine) then PPU12

R44. if (*xil* = free air) and *(xgsud* = yes) then PPU13

R45. if *(sal* — elevated) and (*hc* = elevated) and (*lc* — high) then PPU14

R46. if PPUIO and PPUll and PPU12 and PPU14 then PPU15

R47. if (*vv* = blood) then PPU16

R48. if PPUl and PPU2 and PPU4 and PPU5 and PPUIO and PPUll and PPU12 then PPU17

R49. if PPUl then PPU (CF=0.7)

R50. if PPU2 then PPU (CF=0.9)

R51. if PPU6 then PPU (CF=0.7)

R52. if PPU13 then PPU (CF = 0.7)

R53. if PPU15 then PPU (CF =0.95)

R54. if PPU17 then PPU (CF=0.9)

R55. if *(po* = gradual) and (pl = epigastrium) and *(pr* - slow, spreading through back) then AcPanl

R56. if *(vp* = yes) and *(vf* = multiple) and *(vn* = persistent) and ***(fp*** = yes) **then** AcPan2

R57. if (*li* = yes) and (*liq* = heavy) and *(lit* = preceding attack) **then** AcPan3

R58. if AcPanl and AcPan2 and *(hpu* — similar) **then** AcPan4

R59. if *(di* = fatty foods) **then** AcPan5

R60. if (st = late) **then** AcPan6

R61. if (abta = epigastrium) and *(abtt* = early) and *(abtd* = late) **then** AcPan7

R62. if *(abrt* > 24hrs) **then** AcPan8

R63. if *(cvt* = true) and *(cvts* = left) **then** AcPan9

R64. if (*abwr* = moderate to severe) **then** AcPanlO

R65. if *(ps* = hypoactive) **then** AcPan11

R66. if *(pp* = hipsflexed) **then** AcPan12

R67. if (*xil* = true) and (*xsl* = true) and *(xcc* — true) **then** AcPan13

R68. if (*spm* = true) **then** AcPanl4

R69. **if** *(sal*>= 1000 units/litre) **then** AcPanl5

R70. if *(pp* = onside) **then** AcPan16

R71. if AcPanl **then** AC (CF = 0.8)

R72. **if** AcPan4 **then AC (CF** = 0.85)

R73. if AcPanl4 then (CF = 0.95)

R74. **if** AcPan5 and AcPan6 and AcPan7 and AcPan8 and AcPan9 and AcPanlO and AcPan11 and AcPanl2 and AcPanl3 and AcPan15 and AcPanl6 **then** AC

R75. **if** *(fc* = present) and (n = present(0.9)) and (*m*= present(0.9)) and *(an* = present (0.9)) and *(vp* = present (0.9)) and *(acg* — yes) and *(acgd* = severe) **then** AH1

R76. **if AH1** and *(dp* = yes) **then** AH2

R77. **if** AH1 and *(cp* = yes) **then** AH3

R78. **if** (pl = RUQ) and (*ls* = enlarged) and *(lp* = yes) and (*lt* - yes) **then** AH4

R79. **if** AH2 and AH4 **then** AH5

R80. **if** AH3 and AH4 **then** AH6

R81. **if** *(uc* = dark) and (*sct* = yellow) and *(stc* = pale) **then** AH7

R82. **if** *(sbl* = increased) and *(sail* = increased) and *(sata* > 400 units/litre) and (sap <= 250) **then** AH8

R83. **if** (*sbsl* = increased) and (*ubsl* — increased) and (*pt* = increased) then AH9

R84. if AH9 and (esr = increased) and (*lc* = normal) and (*lc* = mild to moderate) **then** AH10

R85. if (*bul* = increases) and (*ubl* — increased) then AH11

R86. if AH4 and AH5 and AH7 and AH8 and AH10 and AH11 then AH

R87. if AH4 and AH6 and AH7 and AH8 and AH10 and AH11 then AH

R88. **if** AH1 **then** AH **(CF = 0.9)**

R89. **if** AH5 **then** AH **(CF = .80)**

R90. **if** AH6 **then** AH **(CF = .80)**

R91. if AH7 **then** AH **(CF = .95)**

R92. **if** AH8 **then** AH **(CF = .95)**

R93. **if** AH9 **then** AH (CF = .95)

R94. **if** *(chp* = yes) and *(ps* = band around chest) and (pr = arms, left epigastrium, neck, head) **then** MI1

R95. **if** *(vp* — yes) and *(vn* — protracted) and (*bpl* = decreased) and (*plr* = fast and feeble) and *(pn* = thready) **then** MI2

R96. if *(bn* = short) and *(anx* — present) and *(sw* = present) and *(ha* — present) then MI3

R97. if *(ot* = early) and *(ecgst* = elevated) then MI4

R98. if *(ot* = moderately late) and *(ecgrws* = diminished) then MI5

R99. if *(ot* = late) and *(tw* = inverted) then MI6

R100. if *(ot* >= 4hrs) and *(ot* <= 6hrs) and *(ckl* = raises) then MI7

R101. if *(ot* = 12hrs (approx.)) and *(ckl* = peak) and *(sgotl* = raises) then MI8

R102. if *(ot* >= 24hrs) and *(ot* <= 48hrs) and *(sgotl* = peak) then MI9

R103. if *(ot* >= 48hrs) and *(ot* <= 72hrs) and *(ckl* = normal) then MI10

R104. *if (ot* — 168hrs (approx.)) and *(sgotl* — high) and *(sgotn* — prolonged) then MI11

R105. if *(cprwr* = yes) then MI 12

R106. if *(cprwr* = no) then MI13

R107. if MI7 and MIS and MI9 and MI10 then MI (CF = .98)

R108. if MI1 then MI (CF = .90)

R109. if MI2 then AP (CF = .95)

R110. if MI4 then MI (CF = .90)

R111. if MI5 then Ml (CF = .90)

R112. if MI12 then AP (CF = .98)

R113. if MI13 then AP

Attribute codes and their meanings

po — onset of pain (abdominal pain)

pot = (abdominal) pain onset time

pi — (abdominal) pain intensity

pl — (abdominal) pain location

pr = (abdominal) pain radiation

vp — whether vomiting present

vf — vomiting frequency

vn = vomiting nature

vv = vomitus

fp — presence of fever

fc = fever with chills

fr = fever with rigors

fs = fever with sweating

li — alcohol intake

lif = alochol intake frequency

liq = alcohol intake quantity

lit = time of alcohol intake (eg., preceding attack)

hpc = history of similar attacks (of cholecystitis)

hpu = history of previous attacks (of ulcer)

di = dietary intolerance

sp — presence of shock

st = time of shock

sd = degree of shock

prp = presence of prostration

abt — abdominal wall tenderness

abrt = abdominal wall rebound tenderness

abtn = abdominal tenderness nature

abta = abdominal tenderness area

abtt = time at which abdominal tenderness is felt

abtd = abdominal tenderness degree

cvt = costoverterbal angle tenderness

cvts = costoverterbal angle tenderness side

abwr — abdominal wall rigidity

abwg — abdominal wall guarding area

ms = Murphy's sign

ps = peristaltic sounds

(*ps* = Psoa's sign)

pp = patient position

xil = X-ray positive for ileus (eg. air, fluid)

xcal = X-ray positive for calculus in gall bladder

xgsud — X-ray showing gas shadow under diagphram

xsl = X-ray Sentinel loop

xcc = X-ray Colon Cutoff sign

spm — Sonogram positive for Pancreatic mass

sgb — Sonogram positive for stones in gall bladder

sal = Serum Amylase level

lc = Leukocyte count

he = Haematocrite count

esr = ESR

j = jaundice

n — nausea

m = malaice

an = anarexia

acg = aversion to cigaretts

acgd = degree of aversion to cigaretts

dp = Diarrhoea presence

cp — constipation presence

ls = size of liver

lp = liver palpability

lt = liver tenderness

uc = urin colour

sct = tint of sclera

stc = stools colour

ubsl — bile salts level in urine

bul = Biluribin urea level

ubl =Urobilinogen level

chp = pain in chest

cprwr = chest pain relief with rest

bpl = BP level

p/r = pulse rate

pn = pulse nature

bn = breath nature

anx — anxiety

ot — observation time

sbl = Serum Biluribin level

satl = SAT (SGOT/SGPT) level

sata = SAT activity

sbsl = SB salts level

ecg = ECG

ecgst = ECG ST

ecgrws = ECG RWS

tw — T wave

ckl = CK level

sgotl = SGOT levels

sgotn = SGOT nature

afp = facies pallor

sw = sweating

pt = Prothrornbin present

prl = presence of restlessness

ha = hyperacidity

AC, AcPan, PPU, Ml and *AH* stand for the disease names Acute Cholecystitis, Acute Pancreatitis, Perforated Peptic Ulcer, Mycardial Infarction and Acute Hepatitis respectively. The numerals after the disease code indicate the satisfaction of some symptoms related to the disease. For example, AC1 and AC2 indicate the presence of some symptoms related to Acute Cholecystitis.

Appendix C

Test Results

The partitioning heuristic is implemented in Turbo Pascal. Results of some sample runs are given below. Partitions obtained for three different input ratios are shown after generating the spanning tree and preparing it for the initial decomposition.

Input rulebase:

ab– >c

bde– >f

bg– >h

i– >j

fg– >k

ag– >i

$

Spanning tree generated:

1 j 1 0–– >i2

2 i 2 1–– >g3

3 g 2 2–– >h4

4 h 2 3–– >b5

5 b 3 4–– >c6–– >f8

6 c 2 5–– >a7

7 a 1 6

8 f 4 5-- >d9-- >e10-- >k11

9 d 1 8

10 e 1 8

11 k 1 8

spanning tree after processing

node structure : attr sdeg prev plen dir rule

1 j 1 0 0-- >i 2 0 -4

2 i 2 1 0-- >g 3 0 -6

3 g 2 2 0-- >h 4 0 +3

4 h 2 3 0-- > *b* 5 0 -3

5 b 3 4 2-- >c 6 2 +1-- >f 8 0 +2

6 c 2 5 1-- >a 7 1 -1

7 a 1 6 0

8 f 4 5 1-- >d 9 1 -2-- >e 10 1 -2-- >k 11 0 +5

9 d 1 8 0

10 e 1 8 0

11 k 1 8 0

spanning tree array elements:

j i 0

i g 0

g h 0

h b 0

b c b 2

c a b 1

b f 0

f d b 1

f e b 1

f k 0

spanning tree array elements with information about **edge, branch, length, dir, rule, arclabel**

j i 1 - 4 0

i g 2 - 6 0

g h 3 + 3 0

h b 4 - 3 0

b c b 2 + 1 0

c a b 1 - 1 0

b f 5 + 2 0

f d b 1 - 2 0

f e b 1 - 2 0

f k 6 + 5 0

Rules in the set RS excluding branches

4 6 3 2 5

spanning tree array elements after labelling with information about **edge, branch, length, dir, rule, arclabel:**

j i 1 - 4 1

i g 2 - 6 2

g h 3 + 3 3

b b 4 - 3 4

b c b 2 + 1 5

a b 1 - 1 0

b f 5 + 2 6

f d b 1 - 2 0

f e b 1 - 2 0

f k 6 + 5 7

Ratio of rules in which rule base parts are to be made : 1 1

arc to be cut 4

Vertex set for partition 1 : j i g h

Proposed rule set : 4 6 3

Vertex set for partition 2: b c a f d e k

Proposed rule set : 1 2 5

Conflict rule set 3 (1,2); (2,1); 5 (1,1); (2,2); 6 (2,1); (1,1);

1 2;2 1; 2 2;1 1; 2 1;1 1;

CRS nodes sorted on degree of the rules :

Rule 3with degree 3;

and its cp nodes sorted on the attribute count : 1 2;2 1;

Rule 5with degree 3;

and its cp nodes sorted on the attribute count : 2 2;1 1;

Rule 6with degree 3;

and its cp nodes sorted on the attribute count : 2 1;1 1;

Final Partition

Ratio of rules in which rule base partitions are to be made : 1 1

PART P1:

VERTEX SET, VS1: j i g h

RULE SET, RS1: 4 6 3

RULECOUNT = 3, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 1: g (2)

Attributes that NEED TO BE REQUESTED FROM OTHERS: b(2)a(2)

PART P2:

VERTEX SET, VS2: b c a f d e k

RULE SET, RS2: 1 2 5

RULECOUNT = 3, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 2: b (l)a (1)

Attributes that NEED TO BE REQUESTED FROM OTHERS: g(1)

Required partitioning obtained

Ratio of rules in which rule base parts are to be made : 2 1

arc to be cut 6 Vertex set for partition 1 : j i g h b c a

Proposed rule set : 4 6 3 1 2

Vertex set for partition 2: f d e k

Proposed rule set : 2 5

Conflict rule set 2 (1,1); (2,3); 5 (1,1); (2,2);

2 3;1 1; 2 2;1 1;

CRS nodes sorted on degree of the rules :

Rule 2with degree 4;
and its cp nodes sorted on the attribute count : 2 3;1 1;

Rule 5with degree 3;
and its cp nodes sorted on the attribute count : 2 2;1 1;

Final Partition

Ratio of rules in which rule base partitions are to be made : 2 1

PART P1:

VERTEX SET, VS1: j i g h b c a

RULE SET, RS1: 4 63 1

RULECOUNT = 4, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 1: b (2)g (2)

Attributes that NEED TO BE REQUESTED FROM OTHERS:

PART P2:

VERTEX SET, VS2: f d e k

RULE SET, RS2: 2 5

RULECOUNT = 2, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 2:

Attributes that NEED TO BE REQUESTED FROM OTHERS: b(l)g(1)

Required partitioning obtained

Ratio of rules in which rule base parts are to be made : 1 1 1

arc to be cut 3

arc to be cut 6

Vertex set for partition 1: j i g

Proposed rule set : 4 6 3

Vertex set for partition 2: h b c a

Proposed rule set : 3 1 2

Vertex set for partition 3: f d e k

Proposed rule set : 2 5

Conflict rule set 3 (1,1); (2,2); 2 (2,1); (3,3); 5 (1,1); (3,2); 6 (2,1); (1,1);

2 2;1 1; 3 3;2 1; 3 2;1 1; 2 1;1 1; CRS nodes sorted on degree of the rules :

Rule 2with degree 4;

and its cp nodes sorted on the attribute count : 3 3;2 1;

Rule 3with degree 3;

and its cp nodes sorted on the attribute count : 2 2;1 1;

Rule 5with degree 3;

and its cp nodes sorted on the attribute count : 3 2;1 1;

Rule 6with degree 3;

and its cp nodes sorted on the attribute count : 2 1;1 1;

Final Partition

Ratio of rules in which rule base partitions are to be made : 1 1 1

PART P1:

VERTEX SET, VS1: j i g

RULE SET, RS1: 4 6

RULECOUNT = 2, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 1: g (2)g (3)

Attributes that NEED TO BE REQUESTED FROM OTHERS: a(2)

PART P2:

VERTEX SET, VS2: h b c a RULE SET, RS2: 3 1

RULECOUNT =2, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 2: b (3)a (1)

Attributes that NEED TO BE REQUESTED FROM OTHERS: g(1)

PART P3:

VERTEX SET, VS3: f d e k

RULE SET, RS3: 2 5

RULECOUNT = 2, OKAY = TRUE

Attributes that MAY BE REQUIRED BY OTHERS, MRB 3:

Attributes that NEED TO BE REQUESTED FROM OTHERS: b(2)g(1)

Required partitioning obtained

Bibliography

[1] A. Acharya and M. Tarnbe. Production Systems on Message Passing Computers: Simulation Results and Analysis. In *Proc. Int. Conf. Parallel Processing*, pages 246-254, 1989.

[2] M. R. Adler and E. Simoudis. Integrating Distributed Expertise. In *Proc 10th International Workshop on Distributed Artificial Intelligence*, chapter 24. MCC, Bandera, Texas, 1990.

[3] R. K. Arora and S. P. Rana. Heuristic Algorithms for Process Assignment in Distributed Computing Systems. *Information Processing Letts*, 11(4-5):199–203, Dec 1980.

[4] A. Basu, A. K. Mazumdar, and S. Sinha. An Expert System Approach to Control System Design and Analysis. *IEEE Trans. on Systems, Man and Cybernetics*, 18(5):685-694, Sep/Oct 1988.

[5] A. Basu, T. K. Nayak, and S. Mukherjee. Design and Simulation of A Parallel Inference Machine Architecture for Rule Based Systems. *Data and Knowledge Engineering*, 4:267-285, 1989.

[6] D. Beasley, D. R. Bull, and R. R. Martin. An Introduction to Genetic Algorithms. *Vivek*, pages 3-19, Jan. 1994.

[7] D. A. Bell and C. Zhang. Description and Treatment of Deadlocks in The HECODES Distributed Expert System. *IEEE Trans. on Systems, Man and Cybernetics*, 20(3):654-664, May/Jun 1990.

[8] R. Bisiani, F. Alleva, A. Forin, R. Lerner, and M. Bauer. The Architecture of the AGORA Environement. In M. N. Huhns, editor, *Distributed Artificial Intelligence Vol I*, chapter 4, pages 99-118. Pitman / Morgan Kaufmann, 1987.

[9] A. H. Bond and L. Gasser. An Analysis of Problems and Research in DAI. In A. H. Bond and L. Gasser, editors, *Readings In Distributed Artificial Intelligence*, chapter 1. San Mateo, CA: Morgan Kaufmann, 1988.

[10] S. Bose, E. M. Clarke, D. E. Long, and S. Michaylov. PARTHENON: A Parallel Theorem Prover for Non-Horn Clauses. *Journal of Automated Reasoning*, 8:153-181, 1992.

[11] N. Botten, A. Kusiak, and T. Raz. Knowledge Bases : Integration, Verification and Partitioning. *European Journal of Operational Research*, 42:111-128, 1989.

[12] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the Performance of the Kernighan Lin and Simulated Annealing Graph Bisection Algorithms. In *86th ACM/IEEE Design Automation Conference*, pages 775-778, 1989.

[13] R. Calinescu and D. J. Evans. A Parallel Simulation Model for Load Balancing in Clustered Distributed Systems. *Parallel Computing,* 20:77-91, 1994.

[14] J. A. Campbell and M. P. Dinverno. Knowledge Interchange Protocols. In Y. Demazeau and J. P. Muller, editors, *Decentralized A.I.* Elsevier Science Publishers, B.V., 1990.

[15] C. H. Cap and V. Strumpen. Efficient Parallel Computing in Distributed Workstation Environments. *Parallel Computing,* 19:1221-1234, 1993.

[16] B. Chandrasekaran. Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design. *IEEE Expert,* l(3):23-30, Fall 1986.

[17] Y.-C. Chang and K. G. Shin. Optimal Load Sharing in Distributed Real-Time Systems. *Journal of Parallel and Distributed Computing,* 19:38-50, 1993.

[18] I.-R. Chen and B. L. Poole. Performance Evaluation of Rule Grouping on a Real-Time Expert System Architecture. *to appear in IEEE Trans. on Knowledge and Data Engineering,* 1994.

[19] Y. Cheng and K.-S. Fu. Conceptual Clustering in Knowledge Organization. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 7(5):592-598, Sep 1985.

[20] A. N. Choudhary, B. Narahari, and R. Krishnamurti. An Efficient Heuristic Scheme for Dynamic Remapping of Parallel Computations. *Parallel Computing,* 19:621-632, 1993.

[21] R. E. Condon and L. M. Nyhus. *Manual of Surgical Therapeutics.* Little Brown and Company, Boston, 5 edition, 1982.

[22] S. E. Conry, R. A. Meyer, and V. R. Lesser. Multistage Negotiation in Distributed Planning. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence,* pages 367-384. San Mateo, CA: Morgan Kaufmann, 1988.

[23] D. D. Corkill. Design Alternatives for Parallel and Distributed Blackboard Systems. In V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors, *Blackboard Architectures and Applications,* pages 99-136. Academic Press Inc, 1989.

[24] N. S. Coulter, R. B. Cooper, and M. K. Solomon. Information-Theoretic Complexity of Program Specifications. *The Computer Journal*, 30(3):223-227, 1987.

[25] G. Cybenko. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279-301, 1989.

[26] K. Das. *Clinical Methods in Surgery.* Eastern Type Foundry & Oriental Printing Works (P) Ltd, Calcutta, 9 edition, 1976.

[27] C. J. Date. *An Introduction to Database Systems*, volume 1. Narosa Publishing House, India, 3 edition, 1986.

[28] R. Davis. Interactive Transfer of Expertise. In B. G. Buchanan and E. H. Shortliff, editors, *Principles of Rule Based Expert Systems,* pages 171-205. Addison Wesley Publishing Company, 1985.

[29] R. Davis and R. G. Smith. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence,* 20(l):63-109, 1983.

[30] Y. Demazeau. Decentralized Artificial Intelligence. In Y. Demazeau and J.-P. Muller, editors, *Decentralized A.I. (Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multiagent World),* volume 2. Elsevier Science Publishers B.V., The Netherlands, 1990.

[31] N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice Hall of India Private Limited, New Delhi, 1986.

[32] V. V. Dixit and D. I. Moldovan. The Allocation Problem in Parallel Production Systems. *Journal of Parallel and Distributed Computing,* 8:20-29, 1990.

[33] A. E. Dunlop and B. W. Kernighan. A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Trans. on Computer Aided Design,* 4(l):92-98, Jan 1985.

[34] E. H. Durfee and V. R. Lesser. Using Partial Global Plans to Coordinate Distributed Problem Solvers. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence,* pages 285–293. San Mateo, CA: Morgan Kaufmann, 1988.

[35] E. H. Durfee and V. R. Lesser. Negotiating Task Decomposition and Allocation Using Partial Global Planning. In M. N. Huhns, editor, *Distributed Artificial Intelligence,* volume 2, chapter 10, pages 229-243. London: Pitman/Morgan Kaufmann, 1989.

[36] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent Communication Among Communicating Problem Solvers. *IEEE Trans. on Computers,* 36(11):1275-1291, Nov 1987.

[37] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Cooperation Through Communication in A Distributed Problem Solving Network. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, volume 1, chapter 2, pages 29-58. Pitman Publishing / Morgan Kaufmann Publishers, 1987.

[38] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in Cooperaive Distributed Problem Solving. *IEEE Trans. on Knowledge and Data Engineering,* l(l):63-83, Mar. 1989.

[39] F. Ercal, J. Ramanujam, and P. Sadayappan. Task Allocation onto A Hypercube by Recursive Mincut Bipartitioning. *Journal of Parallel and Distributed Computing,* 10:35-44, 1990.

[40] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys,* 12(2):213-253, Jun 1980.

[41] L. D. Erman and V. R. Lesser. A Multi-Level Organization for Problem Solving using Many, Diverse, Cooperating Sources of Knowledge. In *Proc. of IJCAI4*, pages 483-490, 1975.

[42] D. J. Evans and W. U. N. Butt. Dynamic Load Balancing Using Task-Transfer Probabilities. *Parallel Computing,* 19:897-916, 1993.

[43] C. M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *19th Design Automation Conference,* pages 175-181, 1982.

[44] C. L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence,* 19:17-37, 1982.

[45] R. A. Frost. *Introduction to Knowledge Base Systems.* Collins Professional & Technical Books, William Collins Sons & Co Ltd., 8 Grafton Street, London W1X 3LA, 1986.

[46] S. Gaglio, R. Minciardi, and P. P. Puliafito. Multiperson Decision Aspects in the Construction of Expert Systems. *IEEE Trans. on Systems, Man and Cybernetics,* 15(4):536-539, Jul/Aug 1985.

[47] B. R. Gaines and M. L. G. Shaw. Induction of Inference Rules for Expert Systems. *Fuzzy Sets and Systems,* 18:315-328, 1986.

[48] K. Q. Gallagher and D. D. Corkill. Performance Aspects of GBB. In V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors, *Blackboard Architectures and Applications,* pages 323-346. Academic Press Inc, 1989.

[49] L. Gasser, C. Braganza, and N. Herman. MACE: A Flexible Testbed for Distributed AI Research. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, volume 1, chapter 5, pages 119-152. Pitman/Morgan Kaufmann, San Mateo, CA, 1987.

[50] L. Gasser and M. N. Huhns, editors. *Distributed Artificial Intelligence, Volume II.* London : Pitman/Morgan Kaufmann, 1989.

[51] L. Gasser and T. Ishida. A Dynamic Organizational Architecture for Adaptive Problem Solving. In *Proc. AAAI-91*, pages 185-190, 1991.

[52] M. R. Genesereth, M. L. Ginsberg, and J. S. Rosenschein. Cooperation without Communication. In A. H. Bond and L. Gasser, editors, *Readings In Distributed Artificial Intelligence,* pages 220-226. Morgan Kaufmann, 1988.

[53] M. R. Genesereth and S. P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7):48–53 *k* 147, Jul 1994.

[54] M. L. Ginsberg. Decision Procedures. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, chapter 1, pages 3-28. Pitman/Morgan Kaufmann, 1987.

[55] F. Gomez and B. Chandrasekaran. Knowledge Organization and Distribution for Medical Diagnosis. *IEEE Trans. on Systems, Man and Cybernetics,* ll(l):34-43, Jan 1981.

[56] P. E. Green. AF: A Framework for Real-Time Distributed Cooperative Problem Solving. In M. N. Huhns, editor, *Distributed Artificial Intelligence,* chapter 6, pages 153-175. Pitman/Morgan Kaufmann, 1987.

[57] A. Gupta. *Parallelism in Production Systems.* PitmanMorgan Kaufmann Publishers, 1 edition, 1987.

[58] W. Harvey, D. Kalp, M. Tambe, D. McKeown, and A. Newell. The Effectiveness of Task-Level Parallelism for Production Systems. *Journal of Parallel and Distributed Computing,* 13:395-411, 1991.

[59] S. Henry and D. Kafura. Software Structure Metrics Based on Information Flow. *IEEE Trans. on Software Engineering,* 7(5):510–517, 1981.

[60] C. E. Hewitt. Offices are Open Systems. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Aritificial Intelligence,* pages 321-329. San Mateo, CA: Morgan Kaufmann, 1988.

[61] C. E. Hewitt. Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial Intelligence,* 47:79-106, 1991.

[62] C. E. Hewitt and J. Inman. DAI Betwixt and Between: From "Intelligent Agents" to "Open Systems Science". *IEEE Trans. on Systems, Man and Cybernetics*, 21 (6): 1409-1419, Nov/Dec 1991.

[63] M. N. Huhns and D. M. Bridgeland. Multiagent Truth Maintenance. *IEEE Trans. Systems, Man and Cybernetics*, 21 (6): 1437-1445, Nov/Dec 1991.

[64] M. N. Huhns, D. M. Bridgeland, and N. V. Ami. A DAI Communication Aide. In M. N. Huhns, editor, *Proceedings of 10th International Workshop on Distributed Artificial Intelligence*, MCC, Bandera, Texas, 1990.

[65] M. N. Huhns, U. Mukhopadhyay, L. M. Stephens, and R. D. Bonnell. DAI for Document Retrieval: The MINDS Project. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, chapter 9, pages 249-283. Pitman/Morgan Kaufmann, 1987.

[66] B. Indurkhya, H. S. Stone, and L. Xi-Chcng. Optimal Partitioning of Randomly Generated Distributed Programs. *IEEE Trans. on Software. Engineering,* 12(3):483-495, Mar 1986.

[67] D. J. M. Intosh, S. E. Conry, and R. A. Meyer. Distributed Automated Reasoning: Issues in Coordination, Cooperation, and Performance. *IEEE Trans. on Systems, Man and Cybernetics,* 21(6):1307-1316, Nov/Dec 1991.

[68] T. Ishida. Optimizing Rules in Production Systems Programs. In *AAAI-88*, pages 699-704, 1988.

[69] T. Ishida. Methods and Effectiveness of Parallel Rule Firing. In *Proc. of IEEE Conf. on Artificial Intelligence Applications,* pages 116-122, 1990.

[70] T. Ishida. Parallel Rule Firing in Production Systems. *IEEE Trans. on Knowledge and Data Engineering,* 3(1):11–17, Mar 1991.

[71] T. Ishida, L. Gasser, and M. Yokoo. Organization Self-Design of Distributed Production Systems. *IEEE Trans. on Knowledge and Data Engineering,* 4(2):123-133, Apr 1992.

[72] R. J. K. Jacob and J. N. Froscher. A Software Engineering Methodology for Rule-Based Systems. *IEEE Trans. on Knowledge and Data Engineering,* 2(2):173–189, Jun 1990.

[73] N. R. Jennings. *Joint Intentions as a Model of Multi-Agent Cooperation.* PhD Thesis, Dept. of Electronic Engineering, Queen Mary and Westfield College, University of London, Mile End Road, London El 4NS, UK, Aug 1992.

[74] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal,* 49:291-307, Feb 1970.

[75] J. D. Keyser and D. Roose. Load Balancing Data Parallel Programs on Distributed Memory Computers. *Parallel Computing,* 19:1199–1219, 1993.

[76] N. A. Khan and R. Jain. Uncertainty Management in a Distributed Knowledge Based System. In *Proc. 9th International Joint Conference on Artificial Intelligence,* pages 318–320, 1985.

[77] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science,* 220(4598):671~680, May 1983.

[78] W. A. Kornfeld and C. E. Hewitt. The Scientific Community Metaphor. *IEEE Trans. on Systems, Man and Cybernetics,* 11(1):24 33, Jan 1981.

[79] H. F. Korth and A. Silberschatz. *Database System Concepts.* Mc-Graw Hill Inc., 2 edition, 1991.

[80] S. Kuo and D. Moldovan. Implementation of Multiple Rule Firing Production Systems on Hypercube. *Journal of Parallel and Distributed Computing,* 13:383-394, 1991.

[81] D. B. Lenat. BEINGS: Knowledge as Interacting Experts. In A. II. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence,* pages 161-168. San Mateo, CA: Morgan Kaufmann, 1988.

[82] V. R. Lesser. A Retrospective View of FA/C Distributed Problem Solving. *IEEE Trans. on Systems, Man and Cybernetics,* 21 (6):1347–1362, Nov/Dec 1991.

[83] V. R. Lesser and D. D. Corkill. Functionally Accurate, Cooperative Distributed Systems. *IEEE Trans. on Systems, Man and Cybernetics,* 11(1):81-96, Jan/Feb 1981.

[84] V. R. Lesser and D. D. Corkill. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine,* 4:15-33, Fall 1983.

[85] V. R. Lesser and L. D. Erman. Distributed Interpretation: A Model and Experiment. *IEEE Trans. on Computers,* 29(12):1144–1163, Dec. 1980.

[86] Y. P. Li. *A Distributed Knowledge Model for Multiple Intelligent Agents.* PhD Thesis, Computer & Information Systems Research Program, Graduate School of Manangement, UCLA, Sep 1987.

[87] F. T. Lin, C. Y. Kao, and C. C. Hsu. Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems. *IEEE Trans Systems, Man and Cybernetics,* 23(6):1752-1767, Nov/Dec 1993.

[88] V. Lun and I. M. MacLeod. Strategies for Real-Time Dialogue and Interaction in Multiagent Systems. *IEEE Trans. Systems, Man and Cybernetics*, 22(4):671-679, Jul/Aug 1992.

[89] J. MacLeod, editor. *Davidsons Principles and Practice of Medicine.* English Language Book Society/Churchill Livingstone, 14 edition, 1984.

[90] T. W. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87-119, Mar 1994.

[91] M. S. Mazer. Reasoning About Knowledge to Understand Distributed AI Systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21(6):1333-1346, Nov/Dec 1991.

[92] J. Mills, M. T. Ho, P. R. Salber, and D. D. Trunkey, editors. *Current Emergency Diagnosis & Treatment.* Lange Medical Publications, Los Altos, California 94023, 1985.

[93] D. P. Miranker. TREAT: A Better Match Algorithm for AI Production Systems. In *Proc. AAAI-87*,pages 42-47, 1987.

[94] W. R. Murray. Dynamic Instructional Planning in the BB1 Blackboard Architecture. In V. Jagannathan, R. Dodhiawala, and L. S. Baum, editors, *Blackboard Architectures and Applications*, pages 455-480. Academic Press Inc, 1989.

[95] G. Myers. *Reliable Software through Composite Design.* Petrocelli/Charter, Newyork, 1975.

[96] O. Nalini Kumari, G. Uma, and B. E. Prasad. Distributed Reasoning in Cooperative Problem Solving. *communicated to IEEE Trans. Systems, Man and Cybernetics.*

[97] 0. Nalini Kumari, G. Uma, and B. E. Prasad. Dynamic Knowledge Distribution in Distributed Production Systems. *to be communicated to IEEE Trans. Knowledge and Data Engineering.*

[98] O. Nalini Kumari, G. Uma, and B. E. Prasad. Knowledge Distribution in Distributed Production Systems. *communicated to IEEE Trans. Knowledge and Data Engineering.*

[99] 0. Nalini Kumari, G. Uma, and B. E. Prasad. Knowledge Base Partitioning in Distributed Intelligent Systems. In *Canadian Workshop on DAI (CWDAI'94)*, May 1994.

[100] O. Nalini Kumari, G. Uma, and B. E. Prasad. Reasoning with incomplete information in Distributed Forward Chaining Systems. In *ECAI Workshop on Decision Theory for DAI Applications,* Aug. 1994.

[101] S. Niizuma and T. Kitahasi. A Problem Decomposition Method Using Differences or Equivalence Relations Between States. *Artificial Intelligence*, 25:117-151, 1985.

[102] J. Y. C. Pan and J. M. Tenenbaum. An Intelligent Agent Framework for Enterprise Integration. *IEEE Trans. on Systems, Man and Cybernetics*, 21(6):1391-1408, Nov/Dec 1991.

[103] D. L. Parnas. On the Criteria To Be Used In Decomposing Systems Into Modules. *Communicationsof the ACM*, 15(12):1053-1058, Dec 1972.

[104] B. E. Prasad, T. S. Perraju, G. Uma, and P. Umarani. An Expert System Shell for Aerospace Applications. *IEEE Expert,* pages 56-64, Aug 1994.

[105] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed Scheduling of Tasks with Deadlines and Resource Requirements . *IEEE Trans. on Computers,* 38(8):1110-1123, Aug. 1989.

[106] J. S. Rosenschein and M. R. Genesereth. Deals Among Rational Agents. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence,* pages 227-234. San Mateo, CA: Morgan Kaufmann, 1988.

[107] P. Sadayappan and F. Ercal. Nearest Neighbor Mapping of Finite Element Graphs onto Processor Meshes. *IEEE Trans. on Computers,* 36(12):1408–1424, Dec 1987.

[108] P. Sadayappan, F. Ercal, and J. Ramanujam. Cluster Partitioning Approaches to Mapping Parallel Programs onto a Hypercube. *Parallel Computing,* 13:1-16, 1990.

[109] M. J. Shaw. Mechanisms for Cooperative Problem Solving and Multi-Agent Learning in Distributed Artificial Intelligence Systems. In M. N. Huhns, editor, *Proc, 10th International Workshop on Distributed AI*, Bandera, Texas, 1990.

[110] K. G. Shin and Y.-C. Chang. Load-Sharing in Distributed Real-Time Systems with State-Change Broadcasts. *IEEE Trans. on Computers,* 38(8):1124–1142, Aug 1989.

[111] B. Shirazi and A. R. Hurson. Guest Editor's Introduction to Special Issue on Scheduling and Load Balancing. *Journal of Parallel and Distributed Computing,* 4(16):271-273, Dec 1992.

[112] J. S. Sichman, Y. Demazeau, and 0. Boissier. When can Knowledge Based Systems be called Agents? In *9th Brazilian Symposium on Artificial Intelligence,* Rio De Janeiro, Sep 1992.

[113] M. P. Singh, M. N. Huhns, and L. M. Stephens. Declarative Representations of Multiagent Systems. *IEEE Trans. on Knowledge and Data Engineering,* 5(5):721-739, Oct 1993.

[114] R. G. Smith. A Framework for Distributed Problem Solving. In *Proc. IJCAI,* pages 836-841, 1979.

[115] R. G. Smith and R. Davis. Frameworks for Cooperation in Distributed Problem Solving. *IEEE Trans. on Systems, Man and Cybernetics,* 11(1):61-70, Jan. 1981.

[116] A. Sohn and J. L. Gaudiot. A Survey on the Parallel Distributed Processing of Production Systems. *International Journal on Artificial Intelligence Tools,* 1(2):279-331, 1992.

[117] L. E. Stanfel. Applications of Clustering to Information System Design. *Information Processing and Management,* 19(1):37-50, 1983.

[118] M. Stefik and L. Conway. Towards the Principled Engineering of Knowledge. *The AI Magazine,* 3(2):4-6, 1982.

[119] K. R. Tout and D. J. Evans. Parallel Forward Chaining Technique with Dynamic Scheduling, for Rule-Based Expert Systems. *Parallel Computing,* 18:913-930, 1992.

[120] G. Uma, B. E. Prasad, and O. Nalini Kumari. Distributed Intelligent Systems: Issues, Perspectives and Approaches. *Knowledge Based Systems,* 6(2):77-86, Jun 1993.

[121] R. Weihmayer and R. Brandau. Modes of Diversity : Issues in Cooperation among Dissimilar Agents. In M. N. Huhns, editor, *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence,* MCC, Bandera, Texas, 1990.

[122] E. Werner. Distributed Cooperation Algorithms. In Y. Demazeau and J.-P. Muller, editors, *Decentralized A.I. (Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multiagent World), Vol 1,* pages 17-31. Elsevier Science Publishers B.V., North-Holland, 1990.

[123] E. Werner and A. Reinefeld. Distributed Algorithms for Cooperating Agents. In M. N. Huhns, editor, *Proceedings of 10th International Workshop on Distributed Artificial Intelligence,* MCC, Bandera, Texas, 1990.

[124] R. B. Wesson, F. A. Hayes-Roth, J. W. Burge, C. Stasz, and C. A. Sunshine. Network Structures for Distributed Situation Assessment. *IEEE Trans. on Systems, Man and Cybernetics,* 11(1):5-23, Jan. 1981.

[125] S. T. C. Wong. COSMO: A Communication Scheme for Cooperative Knowledge-Based Systems. *IEEE Trans. on Systems, Man and Cybernetics*, 23(3):809-824, May/Jun 1993.

[126] C. C. Woo and F. H. Lochovsky. Knowledge Communicatin in Intelligent Information Systems. *International Journal of Intelligent and Cooperative Information Systems*, **1(1)**:203-228, 1991.

[127] J. Xu and K. Hwang. Mapping Rule-Based Systems onto Multicomputers Using Simulated Annealing. *Journal of Parallel and Distributed Computing*, 13:442-455, 1991.

[128] J. Xu and K. Hwang. Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer. *Journal of Parallel and Distributed Computing*, 18:1-13, 1993.

[129] J. D. Yang, M. N. Huhns, and L. M. Stephens. An Architecture for Control and Communications in Distributed Artificial Intelligence Systems. *IEEE Trans. on Systems, Man and Cybernetics*, 15(3):316-325, May/Jun 1985.

[130] C. Zhang. Cooperation under Uncertainty in Distributed Expert Systems. *Artificial Intelligence*, 56:21-69, 1992.

[131] C. Zhang and D. A. Bell. HECODES: A Frame Work for HEterogeneous Cooperative Distributed Expert Systems. *Data and Knowledge Engineering*, 6:251-273, 1991.

[132] G. Zlotkin and J. S. Rosenschein. Cooperation and Conflict Resolution via Negotiation Among Autonomous Agents in Noncooperative Domains. *IEEE Trans. on Systems, Man and Cybernetics*, **21** (6):1317-1324, Nov/Dcc 1991.