J. Indian Inst. Sci., Nov.–Dec. 1996, **76**, 751–765.

Distributed problem solving in time-constrained domains

G. UMA* AND T. SIVA PERRAJU**
*Department of Computer and Information Sciences, University of Hyderabad, Hyderabad 500 034.
(e-mail:guma-cs@uohyd.ernet.in)
**Research Center, Imarat, Vignyan Kancha, Hyderabad 500 069.

Received on December 15, 1994; Revised on June 24, 1995.

Abstract

Distributed problem solving (DPS) techniques assume *a priori* partitioning of the problem and also problem-solving knowledge. In most time-constrained domains (TCDs), the problem and consequent partitioning of knowledge is not trivial. A graph-based partitioning technique for rule-based systems using individual agent capabilities was developed earlier. In TCDs, the application's time constraints are an additional factor to be considered while partitioning the knowledge. In this paper, we extend the knowledge partitioning technique by proposing heuristics to be applied in TCDs. Further, a high-level Petri net (HLPN)-based knowledge model is used instead of the graph proposed in the earlier approach. HLPN is a formal modelling technique which integrates various aspects of intelligent problem-solving like knowledge representation, verification and reasoning.

Keywords: Distributed problem solving, high-level Petri nets, knowledge partitioning, problem decomposition, and time-constrained domains.

1. Introduction

Domains in which the problem-solving activity is governed by time constraints are termed as time-constrained domains (TCDs). Solutions obtained in violation of the time constraints have little or no value. Real-time applications fall under TCDs. Very often, many real-time problem-solving tasks[1] must share the computing resource among themselves. This leads to two timing constraints, *viz.*, deadline and available computation time. In real-time control problems, meeting the deadline is important. Solutions obtained beyond this time have no value or have decreasing value with time. Obtaining a solution becomes more important, rather than obtaining the best solution. Examples are aerospace checkout diagnosis, process control, etc. In real-time planning problems, usually the solution quality improves monotonically with increasing computation time. Hence, in these problems, the problem-solving agent endeavours to increase the available computing time for the task unlike in control problems. Examples are robotic path planning and robot shooting gallery[2].

In this paper, we focus our attention on real-time control problems. To facilitate the discussion, we take the example of ground-based launch control of an aerospace vehicle[3]. Problem-solving systems for this domain should be able to

(i) accept data asynchronously;

(ii) initiate reasoning in response to changes in the environment;

(iii) respond to events while meeting the timing constraints;

(iv) explicitly represent temporal data and knowledge; and

(v) reason about and respond to multiple simultaneous faults while recognising propagated faults.

Intelligent problem-solving systems can be used to assist operators to arrive at correct decisions in time-constrained control applications where there is continually arriving and large volume of data. However, the problem-solving system has to be augmented to manage temporal data, knowledge and events and react to simultaneous disjointed events. Further physical systems are too complex and large to be modelled by a single agent. Applications in real-time domains are complex and outgrow the problem-solving capabilities of a single agent. Aerospace domain is typical example of such applications. It is more realistic to model the problem-solving system as a network of asynchronous problem-solving agents(or simply agents). This gives rise to the distributed problem solving (DPS) paradigm[4–6] with accompanying advantages like speed, modularity and ease of maintenance. Problem decomposition is the necessary first step in implementing DPS systems.

Problem decomposition helps in breaking complex problems into more simpler ones whose solutions are easier to achieve than the larger whole. Problem decomposition helps to explore alternate sets of problem-solving methods. Usually, problem decomposition is viewed as a consequence of natural decompositions like functional, geographical and spatial. In this view, problem decomposition becomes a consequence of frameworks and organisational structures used in the distributed problem-solving system. In real-time domains, usually natural decompositions do not exist or are not easily available. Hence, unlike in other domains, problem decomposition solely depends on the capabilities of the individual agents (like processing power, available knowledge). Therefore, the frameworks and organisational structures of the DPS systems should reflect problem decomposition.

Most of the existing work on distributed problem-solving systems is on functional distribution or about geographically distributed data. There is no explicit provision for distributing a given knowledge base among several agents with relevant data. The problem is assumed to be already decomposed into sub-problems with possible overlap, and allocated to agents. The solution of such a problem with overlapping sub-problems obviously requires inter-agent communication of partial results or data. As a result of improper distribution of knowledge and data, the communication overhead may be much more compared to the gain in speed. Therefore, it is essential to develop a domain-independent method to partition a given knowledge base into a given number of parts of specified size, so that these parts may be allocated to agents of suitable capacity. The data must also be suitably partitioned so that inter-agent communication for data and partial results is minimised. Besides appropriate problem decomposition, fast pattern matching and load balancing are some of the advantages of an appropriate partitioning

of knowledge and data. Further, providing meta knowledge about the data needed by other problem solvers (agents) as part of the partitioning process facilitates distributed reasoning[7].

In this paper, we propose such a heuristic-based approach which partitions a given knowledge base and associated data among different problem-solving agents based on their capabilities and the time constraints imposed by the application. In Section 2, we present a rule-based knowledge representation scheme which can capture the semantics of applications in time-constrained domains. This is followed by a high-level Petri net (HLPN) model that integrates the semantics of the knowledge representation scheme and the forward-chaining reasoning process. In Section 3, a knowledge-partitioning algorithm that works on the HLPN graph of a knowledge base to distributed knowledge (*i.e.*, rules), based on timing constraints is developed. This is followed by an example of knowledge partitioning in a real-time application. Section 4 presents the conclusions and some future directions of research in this area.

2. Knowledge representation

Rules are a widely used formalism to represent knowledge. They have been augmented to represent knowledge about temporal properties and behaviour[3,8]. There are three types of rules, *viz.*, autonomous, clock synchronous and spanning rules. Spanning rules are further divided into event- and time-spanning rules. Autonomous rules are generic rules from which the other two rule types are derived. Clock-synchronised rules model knowledge about events and the temporal relationships between them. Spanning rules model knowledge about trends based on historical data. Examples of knowledge represented by these different rule types are given below.

Example of autonomous rules

Rule A1
Premise: Hydraulic_system.pressure > 1000 Ksc and Yaw.fb > 15°
Action: Hydraulic_system.status = OK.

Examples of clock synchronised rules

Knowledge: If the auto pilot has issued a Yaw command, the Yaw feedback is to become equal to the Yaw command 2 seconds after the command is issued but before 6 seconds.
Rules:

*C*1
Event: Yaw.cmd == +10°
Time limit: 2 seconds
Time operator: "BEFORE"
Premise: Yaw.fb == −10°
Action: Yaw.Controlstatus = Fast

*C*2
Event: Yaw.cmd == +10°

Time limit: 6 seconds
Time operator: "AFTER"
Premise: Yaw.fb $\neq -10°$
Hold: Yaw.Controlstatus = Lag

Example of event-spanning rules

Knowledge: If hydraulic system pressure is less than 1000 Ksc *and its rate of decrease is more than* 10% *in the last* 100 *seconds then a leak in the pipe system is suspected.*

*S*1
Event: Hydraulic_system.pressure ≤ 1000 l
From time: 100 seconds
Spanning premise: Decrease(Hydraulic_system.Pressure) > 10%
Hold: [message "leak in pipe system suspected, check up"]

Example of time-spanning rules

Knowledge: If the increase in the hydraulic system pressure is greater than 5% *in the last five seconds then it can be concluded that the hydraulic system is overcharged.*

*S*2
Cycle time: 5 seconds
From time: 10 seconds
Spanning premise: Increase(Hydraulic_system.pressure) > 10%
Hold: [message "hydraulic system overcharged"]

These rules can be modelled using HLPN (Fig. 1). Petri nets are formal models that are simple and powerful to represent complex systems with concurrent interacting components[9]. It is possible to integrate different aspects of DPS, *viz.*, knowledge representation, communication, coordination and distributed reasoning in a Petri net model. An HLPN for modelling a distributed problem solving-system is defined. It consists of three different parts: *net structure, the declarations,* and *net inscriptions*[10].

The *net structure* is a directed graph with two kinds of nodes, *places* and *transitions*, interconnected by arcs such that each arc connects two nodes of different kinds.

There are four types of places:

(i) *ordinary places* (OP) are places as defined in elementary Petri nets.

(ii) *cumulative places* (CuP) where the tokens are not removed on firing of their output transitions; tokens in such places have time tags and stored in archives so that they can be retrieved later[4,6].

(iii) *color places* (CoP) where tokens have colors belonging to the defined color set.

(iv) *time places* (TP) which has an interval $[t_1, t_2]$. A token placed in a time place at time t_m is available for firing of an output transition from time $t_m + t_1$ to $t_m + t_2$[11].

After $t_m + t_2$, the token will be removed from the time place, even if no output transition fires.

Transitions are of three types.

(i) *ordinary transitions* (OT) that are as defined for elementary Petri nets;

(ii) *periodic transitions* (PT) that have a frequency of firing(f)[12,13];

(iii) *timed transitions* (TT) which have an interval $[t_1, t_2]$. A transition enabled at time t_e can fire between $t_e + t_1$ and $t_e + t_2$. If the transition does not fire before $t_e + t_2$, then it cannot fire even if the corresponding input places have the necessary markings. New markings are required in the input places to enable the timed transition to fire.

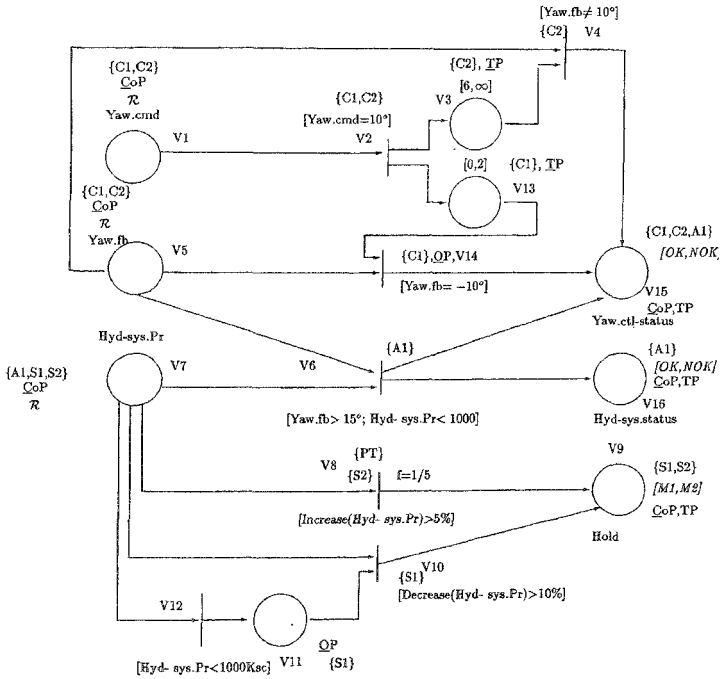

FIG. 1. High-level Petri net representation for aerospace rule base.

The evaluation function associated with a transition denotes the evaluation of an expression (premises or events in the rules) when the transition fires. If the output is a colour place, this expression must evaluate to a colour belonging to the colour set of the output colour place. The expression contains variables which will be bound to the colour of the token from the input colour place. Multiple occurrences of the same variable in such an expression will be bound to the same colour.

There are two types of arcs.

(i) *ordinary arcs* (OA) as defined in elementary Petri nets, and
(ii) *inhibitor arcs* (IA) that end with a circle rather than an arrow head and are semantically the same as the *NOT* in logic circuits.

HLPN is formally defined as a triple H = <P, T, A> where

- P is a finite set of places. Each can be one of the above types {CuP, TP, CoP, OP}
- T is a finite set of transitions. Each can be one of the above types {OT, PT, TT}
- A is a finite set of directed arcs. Each can be one of the two types {OA, IA}.

The firing rules of the three transitions are as follows:

- enabled *Ordinary transitions* follow firing rules as defined for elementary Petri nets.
- enabled *Periodic transitions* are fired once in the period.
- enabled *Timed transitions* are fired within their interval.

Net inscriptions are attached to a place, transition or arc. Places have five different kinds of inscriptions: *names, types, rule labels, time interval* and *colour sets*. Name, type, and rule labels must be given for every place but other two inscriptions may or may not be present depending on the type. Transitions have four different types of inscriptions: names, types, rule labels and evaluation functions. Arcs have only one kind of inscription: labels. All net inscriptions are positioned next to the corresponding element and to distinguish between them we write names and labels in plain text, colour sets in italics, types are underlined and evaluation functions are enclosed in square brackets. Names have no formal meaning except that they serve to identify the nodes.

In HLPN in Fig. 1, the places Yaw.cmd, Yaw.fb, Hyd-sys.Pr with a zero indegree represent external inputs and participate in the premises and events of various rules. The places Yaw.ctl-status and Hyd-sys.status represent attributes that participate in the actions of rules. The place Hold represents the Hold slot of different rules. A single place can have more than one input transition, *i.e.*, the corresponding parameter is participating in the actions of more than one rule. This leads to multi-colour tokens coming in these places, depending on the rule transition that has fired.

In Fig. 1*, consider the inscriptions for the node V1. V1 represents the external input Yaw.cmd. The inscriptions Yaw.cmd and V1 are names of places. Yaw.cmd can take multiple values. Hence, V1 is a colour place with inscription CoP. The range of values that can be taken by Yaw.cmd are real numbers. So, the inscription $\mathcal{R}$ indicates the

* Figure 2 shows the HLPN without net inscriptions.

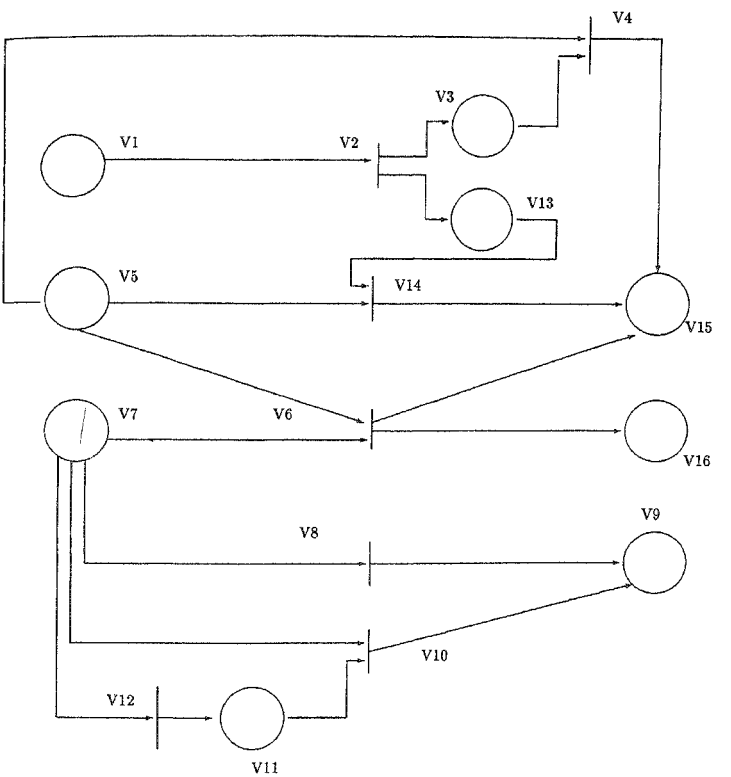

FIG. 2. HLPN of the aerospace rule base without net inscriptions.

colour set of V1. The inscription {C1,C2} indicates that the node V1 participates in two rules, *viz.*, C1 and C2.

Node V8 represents the rule transition of rule S2. Inscriptions V8, S2 are the names of the transition. Since S2 is a time-spanning rule and is to be invoked repeatedly, this transition is a periodic one. The inscription PT indicates the same. The evaluation function associated with the rule premise is inscribed below the transition. The inscription $f = 1/t$ indicates the firing frequency of the rule(transition).

3. Partitioning and allocation

A linear time heuristic for partitioning knowledge in the form of rules for a set of agents in a distributed production system has been developed[14]. It reduces the data inconsistencies and communication, and helps in reasoning. The rules are represented as a graph and the partitioning is applied on this graph. The vertices are the data elements and edges denote rules. Literature on graph partitioning usually refers to vertex partitioning, particularly two-way partitioning which is also referred to as bisection[15]. If multiple parts, say k, are required, it is done by repeatedly bisecting the graph. The emphasis in Nalinikumari[14] is to obtain a k-way graph partition such that rules are in the given proportion, without using repeated bisection. Data dependencies and adjacency of data and rules are exploited in doing this. We present below an informal discussion of the heuristic.

The k-way partitioning[14] consists of two phases: *initial k-way decomposition* and *boundary refinement*. In the first phase, knowledge graph has to be initially cut into k disjoint components by cutting the graph at k-1 places using cut sets. To identify the cut sets, it is necessary to generate a spanning tree of the graph. Usually, the spanning tree covers most of the rules. In the spanning tree the chain** is identified and the edges on the chain are labelled sequentially. The total number of rules and the proportions in which parts are to be obtained give us the number of rules in each part. This is used to determine the edges (on the chain) to be cut for obtaining the graph components. Once we roughly determine which rules belong exclusively to each part, we can check if the boundaries must be smoothened† to get the required partitioning without disturbing the inner portions of the parts. There are several alternatives to be considered in boundary smoothening. In the heuristic proposed[14,16], the total number of data elements participating in a rule on the partition boundary, and the number of data elements in each part only are considered[14]. A boundary rule is assigned to the part that contains maximum number of its data elements. Directories for maintaining attributes to be shared among different partitions are created, and are used by agents during the reasoning process. In TCDs, when the partition boundary cuts a time-constrained rule (TCR) (*e.g.*, clock-synchronised rules with BEFORE operator and time-spanning rules)[17] the above heuristic alone does not suffice. It is necessary to consider the associated time constraints when deciding in which partition a rule is to be placed. The following possibilities exist when such rules are on the boundaries of the partitions obtained by the algorithm.

(i) a time place in one partition and its output time transition is in another partition;
(ii) a time place in one partition and its input transition is in another partition;
(iii) a periodic transition in one partition and its input place in another partition;
(iv) a timed transition in one partition and its corresponding input place in another partition; and

** A path connecting one node to another in a spanning tree is termed a semi-path. The longest semi-path in a spanning tree is termed a chain.

† While partitioning it is possible that some rules are on the boundary and can be put in one of the partitions; the process of determining to which partition these should be assigned is boundary smoothening.

(v) a time transition in one partition and its corresponding output place in another partition.

Case 1

In this case, when a marking appears in the time place with interval $[t_{p1}, t_{p2}]$, it takes a certain communication time t_c, before the transition in the other partition gets enabled. So the effective time interval of the place is now $[t_{p1} + t_c, t_{p2} + t_c]$. Now to satisfy the time constraint on the output time transition, its interval would now become $[t_{t1} + t_c, t_{t2} - t_c]$, where $[t_{t1}, t_{t2}]$ is the interval of the time transition in the other partition. If t_f is the firing time of the transition, then the following relation must hold good

$t_c + t_f < t_{t2}$, or
$t_c < t_{t2} - t_f$.

If the above relation is not satisfied, it should be ensured that both the time place and time transition are in the same partition.

Case 2

In this case, the interval of the time place becomes $[t_{p1} + t_c, t_{p2} - t_c]$, thus reducing the lifetime of the marking in the time place. In order to satisfy the time constraint on the output time transition it is necessary that,

$t_{t2} - t_c > t_c + t_f$, or
$t_c < (t_{t2} - t_f)/2$.

If the above relation is not satisfied, it should be ensured that both the time place and the input transition are in the same partition.

Case 3

In this case, we assume that t_{ctot} is the total communication time that includes the time for transmitting the request + the time for processing the request + the time for reply. Also, the update rate of the input place is U. If t is the time over which data is required and f is the firing frequency, then

$Ut \times t_{ctot} + t_f < 1/f$, or
$t_{ctot} < (1/f - t_f)/Ut$.

If the above relation is not satisfied, it should be ensured that both the place and output periodic transition are in the same partition.

Case 4

In this case, the interval of the timed transition becomes $[t_{t1} + t_c, t_{t2} - t_c]$. In order to satisfy the time constraint on the output time transition it is necessary that,

$t_{t2} - t_c > t_c$, or
$t_c < t_{t2}/2$.

If the above relation is not satisfied, it should be ensured that both the time place and input transition are in the same partition.

Case 5

In this case, the output marking is available in the place after a delay of t_c.

All the above cases[††] can appear in a knowledge graph either individually or in conjunction. If they appear in conjunction it is necessary that all constraints are satisfied.

3.1. *Knowledge partitioning algorithm*

The algorithm for knowledge partitioning using the HLPN graph representation is given below.

Inputs

(i) High-level Petri net for the rule base consisting of N rules.

(ii) Proportion $p_1:p_2:...:p_k$ in which rules are to be distributed among agents (this proportion reflects the problem-solving capabilities like processing power of the agents).

Outputs

(i) Rule base subsets $P_1, P_2,..,P_k$ with rules in the given ratio.

(ii) Directories for the rule base subsets with attributes details—owned or shared.

Steps

1. (* spanning tree generation and marking *)

   (i) Generate a spanning tree for the given HLPN graph (all places and transitions are considered as nodes in the graph). The tree generation starts at node with a degree of 1.

   (ii) Identify the chain (longest semi-path) of the spanning tree.

   (iii) Starting at the initial node of the chain, label the edges of the chain with consecutively increasing integers. If a branch is encountered at any node, then label the edges of the branch if the nodes in the branch have rule labels other than those present on the nodes of the chain. Let MAXLABEL be the maximum label on the edges.

2. Using the proportion in which rules have to be partitioned, determine the number of rules in each partition.

   $nr_i = N \times p_i/\Sigma p_i$

3. (*initial decomposition*)

   (i) Divide the labelled edges of the spanning tree in the partition proportions. $e_i =$ MAXLABEL $\times\, p_i/\Sigma p_i$.

[††] More cases have been detected after reporting of this work.

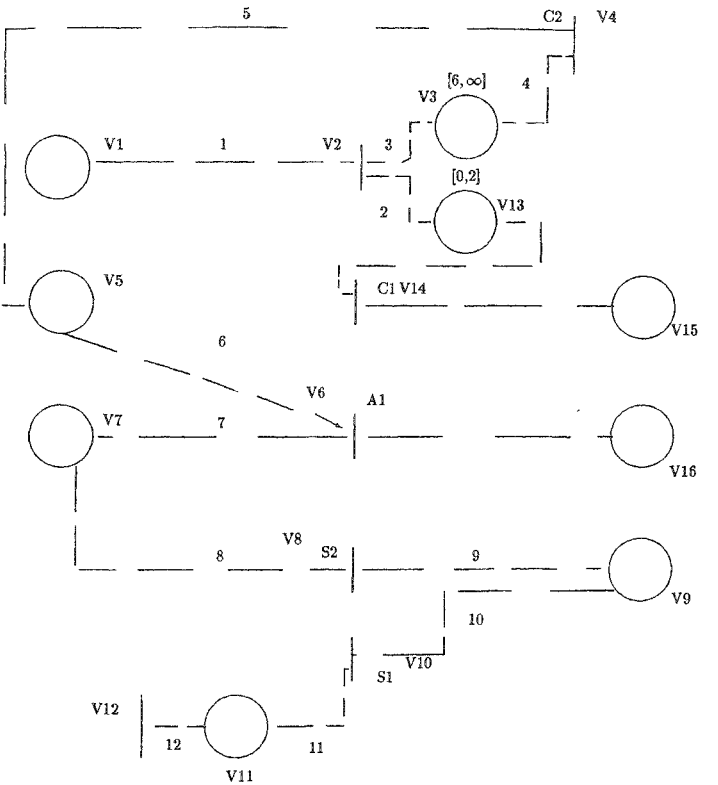

FIG. 3. Spanning tree(dashed lines) of the HLPN knowledge graph.

(ii) For each partition form the vertex sets VS_i
$VS_i = \{x/x$ is the second vertex of edge e_{i-1}, or
x is the first vertex edge of e_i, or
x is a vertex incident on edges between e_{i-1} and $e_i\}$

For each partition determine proposed rule sets (PRS)
$PRS_i = \{x/x$ is a rule label for a vertex in $VS_i\}$.

(iii) Form the cut-set rule-set (CRS)
$CRS = \{x/x \in PRS_i \ \forall x \in \text{in } PRS_j \ \forall i \neq j\}$

4. (The data required for matching and firing rules in CRS are present in more than one partition. Hence these rules can be allocated to either of the partitions in whose vertex sets the data nodes of the rule are present.)

For each rule r in CRS

(i) determine the partitions in which r is present;

(ii) from these partitions find a partition such that timing constraints enunciated in Section 3 are satisfied, and allot r to that partition.

5. For each partition determine the following sets

$MRB_i = \{x/x$ is a data item present in partition p_i and may be requested by rules in another partition$\}$
$NRF_i = \{<x, p_j> /x$ is a data item present in partition p_j and is required for rules in partition $p_i\}$

6. stop

3.2. *Example*

The steps of the algorithm are explained with the help of an example. Figure 3 gives the spanning tree of the HLPN knowledge graph.

The algorithm is traced below.

1. (i) The spanning tree edges are: <V1,V2>, <V2,V13>, <V13,V4>, <V2,V3>, <V3,V4>, <V4,V5>, <V5,V6>, <V6,V16>, <V6,V7>, <V7,V8>, <V8,V9>, <V9,V10>, <V10,V11>, <V11,V12>

(ii) The path {V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12} forms the chain (longest semi-path) of the spanning tree.

(iii) The arcs in the spanning tree are consecutively numbered if they belong to new rules (Fig. 2).

2. Assuming that the partitions are to be made in the ratio of 2:3(p_1, p_2), the size of the first partition $z_1 = 2$ (no. of rules $\times p_1/p_1 + p_2$), $z_2 = 3$.

3. (i) The edge at which the partitions is to be made is 5 (max. label no. on the chain $*p_1/p_1 + p_2$).

(ii) So, now all nodes beginning at edge 1 till edge 5 fall in the first vertex set, similarly are rules associated with transitions beginning at edge 1 till edge 5 fall in the first rule set. The remaining nodes and rules go to the second sets. So,

$VS_1 = \{V1, V2, V3, V4, V13\}$
$VS_2 = \{V5, V6, V7, V8, V9, V10, V11, V12, V14, V15, V16\}$
$PRS_1 = \{C1, C2\}$
$PRS_2 = \{A1, S1, S2, C1, C2\}$

(iii) Since the rules C1 and C2 are found in both the partitions, they form the cut-set rules. So, CRS = {C1, C2}

4. In the graph, the arcs cut in the partitioning are <V4,V5>, <V13,V14>, <V4, V15>. These cuts come under cases 4, 1 and 5, respectively, as discussed in Section 3.

Consider the arc <V4,V5>. For V4 and V5 to be present in different partitions it is necessary that $t_c < \infty(t_{r2}$ of V5). Since any communication delay is finite, this constraint is always satisfied. So, rule C2 can be in partition 1. Consider the arc <V13,V14>. Here, t_{p1} and t_{r1} are 0, while t_{p2} and t_{r2} are 2. If t_c and t_f are the average communication and rule firing times, then the constraint $t_c < 2-t_f$ must be satisfied for V13 and V14 to be in separate partitions. In this case, rule C1 will be in partition 2. However, if the constraint is not satisfied, it is necessary that V14 be shifted to VS_1 (C1 is shifted to partition 1). If the shift is effected, then the arcs <V5, V14> and <V14, V15> are cut. The cut on the latter arc corresponds to case 5. In this case, the value is available at V15 after a time delay of t_c. Since there are no further constraints on V15 this is acceptable. The cut on the former arc corresponds to case 4. In this case, it is necessary that $t_c < 1$ ($t_{r2}/2$ of V14) for V5 and V14 to be different partitions.

Consider the cut on <V4,V15>. In this case the value is available at V15 after a time delay of t_c. Since there are no further constraints on V15 this is acceptable.

Summarising, rule C2 will be in partition 1, while C1 can be either in partitions 1 or 2 depending on the constraints to be satisfied. We will assume that C1 is also in partition 1. The final sets are

VS_1 = {V1, V2, V3, V4, V13, V14}
VS_2 = {V5, V6, V7, V8, V9, V10, V11, V12, V15, V16}
PRS_1 = {C1, C2}
PRS_2 = {A1, S1, S2}

Based on the above partitioning, rules C1 and C2 require the value of place V5 and they modify the value of V15. This information should be maintained as meta knowledge. Two directories are maintained with each partition, *viz.*, need to be requested from (NRF) and may be requested by (MRB).

NRF_1 = {(V5,2)}
$MRB_1 = \phi$
$NRF_2 = \phi$
MRB_2 = {(V5,1)}

In some cases, it is possible that the smoothening rules generate partitions which are not in the designated proportions. Clearly, this is a case of mismatch between the individual node capabilities and the problem requirements. It is practically not possible to obtain partitions in the right proportions for every problem-solving activity. There are two factors which affect the performance under such circumstances. They are the individual computing power of the nodes and the communication bandwidth.

Usually, every node will have a margin and can take additional load to a limited extent and yet meet the time constraints. We hope that this margin would take of imbalances in the partitions. However, if the imbalances grow beyond the margins of individual nodes, then the given distributed problem-solving system needs to be augmented with additional nodes to meet the time constraints of the application.

If the communication bandwidth is low, then it is possible that there will be a single equivalent class. This is because the slow communication medium does not permit distributed problem solving. This necessitates an increase in the communication capacity and the speed before partitions can be made and distributed problem solving be used.

4. Conclusions

Many domains have clear functional or geographical separation of knowledge so that they can be implemented easily as distributed problem-solving systems. However, in TCDs no natural distributions are available. In these domains, problem and knowledge decomposition is solely dependent on the capabilities of individual problem-solving agents. Further, results must be obtained in a given time limit, communication delays due to implementation as a distributed problem-solving system should not be prohibitive. Knowledge partitioning by considering the temporal constraints of the rules falling on the boundaries is necessary for TCDs. We have shown that a high-level Petri net is suitable to represent different types of rules encountered in a TCDs such as aerospace application. Boundary smoothening criteria are developed for heuristic partitioning of HLPN knowledge graphs. The rules are distributed such that all time constraints are met even after communication delays are considered. This has been done by taking a static estimate of the communication delays and the processing time.

A dynamic repartitioning can be done by periodically performing boundary smoothening with actual measurements of these two parameters instead of a static estimate. A suitable distributed reasoning strategy is being formulated incorporating these features. This distributed reasoning algorithm[7] will consider the priorities and timing constraints of the partially matched rules, while requesting for nonlocal information.

References

1. STROSNIDER, J. K. AND PAUL, C. J. — A structured view of real-time problem solving, *AI Mag.*, Summer 1994, 45–66.
2. BODDY, M. AND DEAN, T. L. — Deliberation scheduling for problem solving in time-constrained domains, *Artif Intell.*, 1994, **67**, 245–285.
3. PRASAD, B. E., PERRAJU, T. S., UMA, G. UMARANI, P. — An expert system shell for aerospace applications, *IEEE Expert*, 1994, 9(4), 56–64.
4. UMA, G. — *A framework for modelling and analysis of distributed intelligent systems*, Ph. D. Thesis, Department of Computer Science, University of Hyderabad, 1991.
5. UMA, G., PRASAD, B. E. AND NALINIKUMARI, O. — Distributed intelligent systems: Issues, perspectives and approaches, *Knowledge-Based Systems*, 1993, 6, 77–86.

6. UMA, G., PRASAD, B. E. AND REDDY, P. G. — Framework for modelling and analysis of distributed problem solving systems, *Knowledge-Based Systems*, 1992, **5**, 295–304.

7. NALINIKUMARI, O., UMA, G. AND PRASAD, B. E. — Reasoning with incomplete information in distributed forward chaining systems. In *Proc. Workshop on Decision Theory for DAI Applications*, Amsterdam, 1994.

8. PERRAJU, T. S., UMA, G. AND PRASAD, B. E. — A scheme for knowledge representation, verification and reasoning in real time asynchronous production systems. In *Proc. IEEE Int. Conf. on Tools with AI*, IEEE CS Press, 1994.

9. PETERSON, J. L. — *Petri net theory and modelling of systems*, 1981, Prentice-Hall.

10. JENSEN, K. — Coloured Petri nets: A high level language for system design and analysis. In *High level Petri nets. Theory and applications* (K. Jensen and G. Rozenberg (eds)), 1991, pp. 44–119, Springer-Verlag.

11. DAVID, R. AND ALLA, H. — Continuous Petri nets, In *Proc. 8th Eur. Workshop on Applications and Theory of Petri nets*, 1987, pp. 275–294.

12. DAVID, R. AND ALLA, H. — Autonomous and timed continuous Petri nets, In *Proc. 11th Int. Conf. on Applications and Theory of Petri nets*, 1990, pp. 367–385.

13. BAIL, J. L., ALLA, H. AND DAVID, R. — Hybrid Petri nets, In *Proc. 1st Eur. Control Conf.*, 1991, pp. 175–194.

14. NALINIKUMARI, O. — *Cooperative problem solving—knowledge base partitioning approach*, Ph. D. Thesis, Department of Computer Science, University of Hyderabad, 1994.

15. KERNIGHAN, B. W. AND LIN, S. — An efficient heuristic procedure for partitioning graphs, *Bell Systems Tech J.*, 1970, **49**, 291–307.

16. NALINIKUMARI, O., UMA, G. AND PRASAD, B. E. — Knowledge base partitioning in distributed intelligent systems. In *Proc. Canadian Workshop on DAI*, 1994.

17. PERRAJU, T. S. — *Object oriented real time asynchronous production systems*, Ph. D. Thesis, Department of Computer Science, University of Hyderabad, 1993.