International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

A Logic Based Framework for Planning for Mobile Agents

Rajdeep Niyogi

Abstract**—**The objective of the paper is twofold. First, to develop a formal framework for planning for mobile agents. A logical language based on a temporal logic is proposed that can express a type of tasks which often arise in network management. Second, to design a planning algorithm for such tasks. The aim of this paper is to study the importance of finding plans for mobile agents. Although there has been a lot of research in mobile agents, not much work has been done to incorporate planning ideas for such agents. This paper makes an attempt in this direction. A theoretical study of finding plans for mobile agents is undertaken. A planning algorithm (based on the paradigm of mobile computing) is proposed and its space, time, and communication complexity is analyzed. The algorithm is illustrated by working out an example in detail.

Keywords**—**Acting, computer network, mobile agent, mobile computing, planning, temporal logic.

I. Introduction

MOBILE agents (MAs) [14] have found wide application in information searching, retrieval, electronic commerce, and network management to mention a few. A plethora of research (both foundational and applications) has been done in mobile agents. Some of the areas include code mobility [10], minimal agent code size [9], reliable communication among mobile agents [17], communication language for agents [25], and security [4].

Research in automated planning [11] is concerned with developing efficient algorithms (or planners) for solving different types of tasks. Efficient planners [6], [13] have been developed for classical planning. The assumptions made in classical planning are: deterministic action effects, full observability, and no exogenous events; i.e., there is no uncertainty. Real-world planning, however, needs to deal with uncertainties. In planning under uncertainty there are some scenarios (see for example [23], [24]) where acting is necessary. This paper considers a real-world scenario where acting is necessary.

A type of planning problems for mobile agents, similar to the Traveling Salesman problem, is considered in [16]. The objective is to minimize the expected time to complete a task for which plans are first found assuming complete knowledge of the network. The work in this paper differs substantially from [16]. The objective of this paper is (i) to provide a logical specification scheme for a type of tasks of mobile agents, and (ii) to find plans for mobile agents, for these tasks, in an environment which is partially observable. Planning for mobile agents may be useful when the agents perform network management tasks and when they are used for information search and/or retrieval.

Rajdeep Niyogi is with the Electronics and Computer Engineering Department, Indian Institute of Technology (IIT) Roorkee, India 247667 (e-mail: rajdpfec@iitr.ernet.in).

Let the task of a mobile agent be to find some information α on a network as shown in Fig. 1. The agent starts at node A and the desired information is available at node D. Assume that faults do not occur in the network and that the information associated with a node is static. If the topology of the network is known a priori, a search algorithm can find an optimal path (in this case two steps).

It is quite realistic to assume that only a partial view of the topology is available at any node of the network. In such a scenario it is not known with certainty whether the agent really reaches node D in two steps. In the worst case it may roam across the entire network before reaching the destination node. Although the agent has, indeed, achieved the given task, the process involved may be quite costly. This stems from the fact that the *migrate* operation of a mobile agent is expensive. Moreover, while the agent is roaming in a network the problem of finding its exact location is computationally intensive. Thus it is easy to imagine the cost involved when the agent makes several wrong moves in a network that is of considerable size. One way to make the search efficient is by communication.

In this paper a partially observable network is considered. Since the model of the network is not known so plan computation cannot be done a priori. An action has to be performed to acquire knowledge about the world. For example, one should either touch the water with his toes or jump into the water to find out whether it is warm. In either case the state changes because he will become wet. Some scenarios where acting is necessary is discussed in Section VII. The related work is also discussed in Section VII.

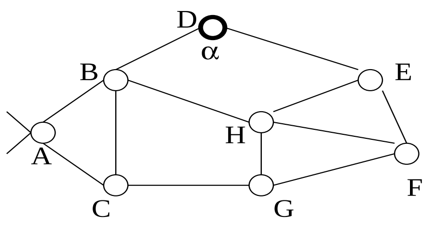


Fig. 1. Acquiring information in a network

The plan structure, for the example given above, consists of only the movement activities of an agent: (i) a sequence of hops ($hop_{A,B}$; $hop_{B,D}$), or (ii) a combination of hops, enter, and exits in a hierarchical network. The primitive activities of an agent, say, delete a file, install a software, or remove virus are not depicted in a plan.

The task (or goal) considered in the above example is called reachability goal. However more complex goals (eg., temporally extended goals) have been considered and suitable logical languages, based on temporal logics, are proposed in [1], [2]. A type of temporally extended goals is considered in this paper. Some examples of tasks of a mobile agent are given below. These tasks are similar to that arising in network management [5].

Example 1 A mobile agent has to reach a supercomputer (*sc*) in the Computer Science subnet (*CS*) of the network of an educational institution (*edu*).

Example 2 Consider an interconnection of private networks. Inside a private network there may be computers with a particular configuration (*pconfig*). The goal of a mobile agent is to hop through those networks, inside which there is at least one such computer, before reaching a network named $Home$.

Example[1] **3** A mobile agent has to reach a host computer with a specific application software (app) installed. But the mobile agent is a Java program and so it has to hop through only computers that have JVM installed (jvm).

These tasks can easily be encoded in a logic LMA that is similar in spirit to the Computation Tree Logic (CTL) [8], which is a branching-time temporal logic. The logical language LMA is discussed in Section III. For instance, Example 1 can be written in LMA as $E(true\ U\ v[edu \wedge CS[v[sc]]])$ that is to be read like a CTL formula $E(p\ U\ q)$–there exists a path where at some point q is true and prior to that at all other points on the path p is true. A term $a[p]$ denotes a location named a where a property p holds; when the property p is of interest $v[p]$ is used, where v denotes some location. Another inspiration for using the logic is from the works in [12], [19] where planning is viewed as model checking CTL goals.

Thus the problem considered in this paper is briefly stated as: *find a plan that achieves a task (goal), expressed in LMA, in a partially observable hierarchical computer network, using two mobile agents. The communication among agents should be as less as possible.* The motivation behind going for little or no communication between agents is based on the fact that the provision of a reliable communication infrastructure for mobile agents is still an open issue [17].

The approach adopted in this work is sketched below. The planning algorithm uses two cooperative mobile agents to compute a plan with minimum communication among themselves. The goals for the individual agents are obtained from the task specification given in LMA. Each agent moves in the network in a depth-first manner. The overall plan is obtained by combining the plans of the individual agents. A plan is stored in a distributed fashion over the network.

The rest of the paper is structured as follows. The network model is defined in Section II. The planning problem is discussed in Section IV. The planning algorithm is given in Section V and its properties are discussed in Section VI. The conclusions and future work are given in Section VIII.

[1]This example is used as the running example in this paper.

II. A Distributed World Model

A hierarchical computer network similar to that adopted in [20] is considered. The network consists of computers located at different geographical locations. For example, the Internet is a collection of private networks, a private network in turn is a collection of VLANs, and each VLAN in turn is a collection of computers. One way to model such a network is to use a top level graph that represents a collection of private networks and each node in the graph has a subgraph that represents the network contained inside it. However, it does not capture the logical boundaries governing the structure: that a VLAN hides the identity of the computers inside it is not captured in this representation.

These boundaries may also refer to administrative or political boundaries. Consider for instance, an agent flying from London to New York and then going to Times Square (located within New York). The agent may first travel in an airplane and get to the airport at New York. But the agent may be detained at the airport and may not be allowed to enter the city of New York. Here permissions (enter, exit) are needed. Barriers or boundaries in the context of mobile computation have been discussed at length in [7].

The notion that a location hides the information and structure of the locations contained inside it is an important aspect in the world model. Each location, in the model, has a unique name, a set of properties, and some computational ability. The subgraphs are assumed to be trees and are referred to as location trees. The model $\mathcal{M}$ of a distributed world consists of a location graph (LG) where a node in LG contains a location tree. LT_l denotes a location tree with root l.

An example of a distributed world is given in Fig. 2. In the Fig. LG consists of 12 locations (1,2,...,12). The location tree for node 8 is only shown; similar trees exist for the other nodes. The properties associated with nodes is indicated by p and q. Neighbors mean the adjacent nodes in LG, eg., nodes 5 and 11 are the neighbors of node 8.

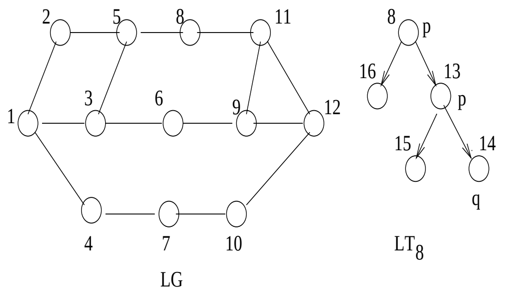
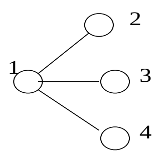


Fig. 2. A Distributed World

The following are the assumptions of the distributed world (DW). The spatial configuration of the locations do not change. DW is not dynamic (i.e., the properties of the locations are time invariant). Faults do not occur in the network. A location knows only its neighbors to which it is linked in the network, but not the properties associated with those neighbors. No location knows the total number of locations in the network. A location has limited computational power–

this is discussed in detail in Section V. A location can take part in a communication with a mobile agent.

A mobile agent working in the distributed world acquires information about the world in an incremental manner. This is illustrated in Fig. 3 with respect to the world in Fig. 2.



Agent ' s state	Information of agent
at location 1	neighbors of 1 are 2,3,4
after communicating with neighbors of 1	properties of 2,3,4

Fig. 3. Agent's information of the world

Thus when the agent is at location 1, it has partial information of the world topology. After communicating it has come to know the properties of the neighbor locations. The agent, however, at this stage has no knowledge about the location trees of the neighbors. The agent has to perform actions to gather the information. Unless it moves to a neighbor (say, location 3) it cannot figure out who the neighbors of location 3 are. Thus acting is necessary in the world. If the agent moves to location 3, its partial view of the world will be as shown in Fig. 4.

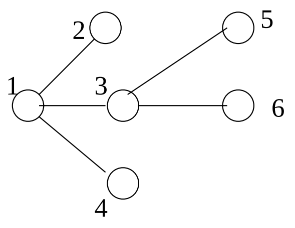


Fig. 4. Information acqusition after acting

III. A Logic for Specifying Tasks

A **l**ogical language for specifying tasks (goals) of a **m**obile **a**gent, abbreviated LMA, in the distributed world is discussed. The logic LMA is based on CTL [8]. The syntax of a CTL formula is given as $f \longrightarrow p \,|\, \neg f_1 \,|\, f_1 \wedge f_2 \,| EXf_1 \,|\, AXf_1 \,|\, E(f_1 \; U \; f_2) \,| A(f_1 \; U \; f_2)$, where $p \in AP$ (the set of atomic propositions), E and A are the existential and universal path quantifiers respectively, X is the next-time operator, and U is the until operator. A CTL formula is defined with respect to a structure $\langle S, R, L\rangle$, where S is a finite set of states, $R \subseteq S \times S$ (R is total), and L is a labeling function that labels states with truth of atomic propositions.

The logical language for specifying tasks in the distributed world should capture the fact that locations have unique names and properties are bound to locations. Moreover, a location may contain other locations. CTL does not allow the use of explicitly named locations. So the construct $C[p]$ is included. It denotes that a property p is true at a location named C. When the property p is of interest, $v[p]$ is used where v signifies some location. There is a clear distinction between a path in a location tree and a path in a location graph. The former implies containment of locations. In order to differentiate the two, the path quantifier E_s and the operator U_s are introduced to specify properties in location trees. This is explained with respect to the distributed world in Fig. 2. Suppose that the truth of a CTL formula $E(p \; U \; q)$ is to be evaluated at location 8. Assume that both p and q are false at locations 5 and 11. Now as per the semantics of CTL, the formula is satisfiable at location 8 since there is a path in the location tree (8-13-14); all the locations (5,11,13,16) are considered neighbors of location 8. However, from the definition of the structure of the world only locations 5 and 11 are the neighbors of location 8. Thus to keep the meanings of paths in location tree and paths in location graph distinct, the two operators U_s and U are used respectively.

Recall that the model $\mathcal{M}$ of a distributed world consists of the location graph LG where a node in LG contains a location tree. Thus the logic LMA has two levels of syntax: ψ for specifying properties of paths in LG and ϕ for specifying properties of paths in the location trees. In the syntax given below $p \in AP$–the set of atomic propositions in $\mathcal{M}$.

A. Syntax and Semantics of LMA

Syntax of LMA

$\psi \longrightarrow C[\phi] \,|\, v[\phi] \,|\, \neg\psi_1 \,|\, \psi_1 \wedge \psi_2 \,|\, E(\psi_1 \; U \; \psi_2) \,|\, \top \,|\, \bot$

$\phi \longrightarrow p \,|\, C[\phi_1] \,|\, v[\phi_1] \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, E_s(\phi_1 \; U_s \; \phi_2) \,|\, \top | \bot$

Semantics of LMA

The satisfaction of an LMA formula ψ is with respect to the distributed world model $\mathcal{M}$, LG, and an initial location $l_0 \in LG$. This is denoted as $(\mathcal{M}, LG, l_0) \models_{LG} \psi$. In $(LG, l_0) \models_{LG} \psi$, $\mathcal{M}$ is understood and so omitted. The satisfaction relation $\models_{LG}$ is dependent on the satisfaction relation for ϕ, which is evaluated at the corresponding location tree. This satisfaction relation is with respect to $\mathcal{M}$, a location l, and the location tree of l. In the following $\mathcal{M}$ is omitted and $(LT_l, l) \models_{LT} \phi$, is defined inductively as:

$(LT_l, l) \models_{LT} p$ iff p holds at l
$(LT_l, l) \models_{LT} C[\phi]$ iff C is a child of l and $(LT_C, C) \models_{LT} \phi$
$(LT_l, l) \models_{LT} v[\phi]$ iff for some child l' of l $(LT_{l'}, l') \models_{LT} \phi$
$(LT_l, l) \models_{LT} \neg\phi$ iff $(LT_l, l) \not\models_{LT} \phi$
$(LT_l, l) \models_{LT} \phi_1 \wedge \phi_2$ iff $(LT_l, l) \models_{LT} \phi_1$ and $(LT_l, l) \models_{LT} \phi_2$
$(LT_l, l) \models_{LT} E_s(v[\phi_1] \; U_s \; v[\phi_2])$ iff for some path starting at l i.e., $(l = l_1, l_2, \ldots, l_j)$ in LT_l $[j \geq 1]$, the following conditions hold: (i) $(LT_{l_j}, l_j) \models_{LT} \phi_2$ and (ii) for all i $[1 \leq i < j]$ $(LT_{l_i}, l_i) \models_{LT} \phi_1$

The satisfaction relation $\models_{LG}$ is defined inductively as: (recall that l_0 is the initial location)
$(LG, l_0) \models_{LG} C[\phi]$ iff $l_0 = C$ and $(LT_{l_0}, l_0) \models_{LT} \phi$
$(LG, l_0) \models_{LG} v[\phi]$ iff for some location $l \in LG$ $(LT_l, l) \models_{LT} \phi$
$(LG, l_0) \models_{LG} \neg\psi$ iff $(LG, l_0) \not\models_{LG} \psi$
$(LG, l_0) \models_{LG} \psi_1 \wedge \psi_2$ iff $(LG, l_0) \models_{LG} \psi_1$ and

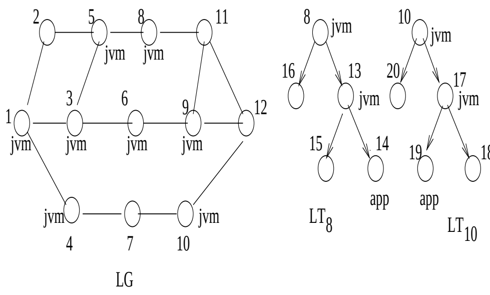


Fig. 5. Semantics of LMA

$$(LG, l_0) \models_{LG} \psi_2$$

$(LG, l_0) \models_{LG} E(\psi_1\ U\ \psi_2)$ iff for some path ($l_0, l_1, \ldots, l_j$) in LG, $[j > 0]$, $(LG, l_j) \models_{LG} \psi_2$ and for all i $[0 \le i < j]$ $(LG, l_i) \models_{LG} \psi_1$

Thus the examples given in Section I can be expressed as:

Example 1: $E(true\ U\ v[edu\ \wedge\ CS[v[super]]])$
Example 2: $E(v[E_s(true\ U_s\ v[pconfig])]\ U\ Home[true])$
Example 3: $E(v[jvm]\ U\ v[E_s(v[jvm]\ U_s\ v[app])])$

The semantics of the formula in Example 3 is now illustrated. Let the network be as shown in Fig. 5. In the Fig. only the location trees for nodes 8 and 10 are shown; similar trees exist for all the other nodes. The propositions jvm and app besides the nodes imply that it is true at these nodes. Only nodes 8 and 10 satisfy the subformula $v[E_s(v[jvm]\ U_s\ v[app])]$. For the path 1-3-5-8, it is easy to see that jvm is true at all the nodes (1,3,5). So the given formula is satisfiable at node 1. Although node 10 satisfies the subformula, there is no path from node 1 that satisfies the given property.

B. Limitations of LMA

Suppose that there is a scenario where an agent is asked to install a printer driver in all workstations of WsT make. Assume that a node in LG represents a workstation of some make. Here the task is similar to a traveling salesman problem. This task is not expressible in LMA. Now let this task be modified as: before reaching the destination node install a printer driver in all workstations of WsT make when the workstations may be within k steps from each other. For this task to be expressible in LMA, the semantics of the until operator U has to be modified. It may be noted that the semantics of U take care of the above task when $k = 1$.

IV. The Planning Problem

The planning domain is defined as $D = (\mathcal{M},\ S, A, R)$, where $\mathcal{M}$ is the distributed world model, S is the set of locations in $\mathcal{M}$, A is the set of actions, and $R : S \times A \rightarrow S$ is the transition function. Actions are of three types. For any $i, j \in LG$ such that j is a neighbor of i, execution of $hop_{i,j}$ at i transfers an agent to j. The agent uses the actions $enter$ and $exit$ for moving in the location trees. If i is a child of j, execution of $enter_i$ at j transfers an agent to i; an $exit$ at i transfers it back to j.

The freedom of roaming about in the distributed world is captured by hop. For the London-NewYork example in Section II, flying can be achieved by hop–that takes the agent from the boundary of one location to the boundary of another. A different type of action is needed to go from the airport to Times Square; for this $enter$ is needed. For returning from Times Square to the airport $exit$ is needed.

Given the planning domain D, an initial location $l_0 \in LG$, and an LMA goal formula $f = E(f_1\ U\ f_2)$, with no occurrence of E in f_1 and f_2, the planning problem is to find a plan π for f at l_0. Henceforth the term goals shall be used instead of tasks. The result of applying π to l_0 is a plan path–$l_0, l_1, \ldots, l_k$, where each $l_i \in LG$, $[0 \le i \le k]$. A plan π satisfies f if the plan path satisfies $f_1\ U\ f_2$. Observe that a path $l_0, \ldots, l_j$ in LG satisfies $f_1\ U\ f_2$ iff $(LG, l_j) \models_{LG} f_2$ $[j > 0]$ and for all i $[0 \le i < j]$ $(LG, l_i) \models_{LG} f_1$ (refer to the semantics in Section III-A).

A plan for f consists of local plans and hops. Let loc_u denote the local plan at location u. The structure of a plan for f is: $loc_{l0}, hop_{l0,l1}, loc_{l1}, hop_{l1,l2}, \ldots, hop_{lk-1,lk}, loc_{lk}$. A hop occurs at least once. Each local plan loc_{li} is of the form $enter_{m1}, enter_{m2}, \ldots, enter_{mk'}, exit, exit, \ldots, exit,\ k' \ge 1$, where $m1 = li$ and $m2 \ldots mk'$ would be descendants of li. The number of $exit$ actions equals the number of $enter$ actions except for the localplan following the last hop. Observe that $enter$ is distinct from hop: to verify a property p at location C ($C[p]$) or to reach location D contained inside C, a hop to C is not enough; an $enter_C$ is needed. However if the goal is $C[true]$ then only a hop to C will suffice. A local plan at a location u in LG for the subformulas f_1 and f_2 is found by procedure $localplan$–that is described in the next Section. The local plan is stored at u.

V. A Planning Algorithm

Planning is done by two mobile agents (Ag_1 and Ag_2) in a distributed manner. A plan for a given goal is also stored in a distributed fashion.

An LMA goal formula $f = E(f_1\ U\ f_2)$ is decomposed to two subgoals. Each subgoal is provided to a mobile agent. The agents collaborate to find a plan for f. In the following, f_1-location is used to to mean that f_1 is true at a location; f_2-location to mean that f_2 is true at a location. The initial location of Ag_1 is l_0. The goal of Ag_1 is to go through f_1-locations until it reaches a location l' where either of the following conditions hold: f_2 is true at l' or l' is marked visited by Ag_2. The initial location of Ag_2 is l_k–that is chosen randomly. (This is done since the agent need not necessarily start at an f_2-location.) Ag_2 first reaches an f_2-location. It then goes through f_1-locations until it reaches a location l' that is marked visited by Ag_1. Each agent combines the other agent's partial plan to obtain the plan for f. Ag_1 computes the prefix of a plan and Ag_2 the suffix of a plan. Here the objective is not finding optimal plans. Both the agents perform depth-first-search for exploring the network. The agents acquire knowledge about the topology of the network incrementally.

The following assumptions are made. An agent always updates its visited information of the locations. An agent communicates only with the neighbors of its current location. The algorithm is based on the paradigm of mobile computing. The agent's mobility is represented by the operation $move$ that symbolizes the process of migration of the agent from one node to another. If an agent finds a plan, or discovers that no plan exists for a given goal, it informs the other agent. For each communication between an agent and a location, all messages are transmitted, unaltered, in exactly one unit of time. A location always responds to a query. Each location knows the location from where the message is received.

A location in LG does the following: (i) communicates with an agent, and (ii) computes the truth of an LMA subformula. A node u in a location tree has complete knowledge of the subtree rooted at u. So the truth of a subformula can be obtained via *model checking* [8]. The computation in (ii) is done exactly once at each location. A location u stores the following: (a) visited information of agents ($u.visited\_\_Ag_i$), (b) the neighbors of u that the agent finds useful (S_u^i), [$i = 1, 2$]. (c) local plan ($u.plan\_f_i$), and (d) *u.parent* and *u.son*. The *parent* and *son* fields of all locations in LG are initialized to $null$.

Description of the algorithm for Ag_1

The procedure $Ag_1.findplan$ is now described. Let Ag_1 be at l and l_1, l_2 are the neighbors of l. It $asks$ l_1 and l_2 about the truth of f_1, f_2, and the visited information of Ag_2 at those locations. If any of these conditions is true at a neighbor, Ag_1 places it in S_l^1. If a neighbor (say l_1) returns f_2 is true, then the son of l is set to l_1. Ag_1 $moves$ to l_1, sets the parent of l_1 to l, marks l_1 visited, and finds a local plan for f_2 at l_1. If a neighbor (say l_1) returns $visited\_\_Ag_2$ is true, then the son of l and parent of l_1 are set as in the previous case. The idea of marking nodes resemble how ants use pheromone to mark paths [21].

As soon as a plan for f is found, Ag_1 informs Ag_2 and terminates. If a neighbor (say l_1) returns f_1 is true, Ag_1 first *moves* to l_1, marks l_1 visited, finds a local plan for f_1 at l_1, and sets the son of l and parent of l_1. Now Ag_1 repeats the above process at l_1. If a satisfying plan for f is not found at l_1, Ag_1 backtracks to l and tries alternate paths. Ag_1 also uses the procedures $proc\_\_Ag_1$ and $localplan$.

Description of the algorithm for Ag_2

The initial job of Ag_2 is to reach an f_2-location. So it communicates with its neighbors and explores the location graph LG. Once an f_2-location (say v) is found, Ag_2 now tries to reach a location (say l'), that has been visited by Ag_1, through f_1-locations. This is done by the procedure $Ag_2.findplan$ which is similar to $Ag_1.findplan$. Once l' is found at l, it sets the parent of l to l', and the son of l' to l. Now it informs Ag_1 and terminates. If Ag_2 fails to reach such a location (l' from v), it finds an alternate f_2-location and repeats the above process. Ag_2 also uses the procedures $proc\_\_Ag_2$ and $localplan$.

```
procedure $proc\_\_Ag_1$
input: an initial location $l_0$ and the subformulas $f_1, f_2$ of an LMA goal formula $f = E(f_1\ U\ f_2)$
outcome: a plan for $f$ (if it exists), failure otherwise
begin
1 $ask$ $l_0$ whether $f_1$ is true at $l_0$
2 if $f_1$ is true at $l_0$
3   then $l_0.visited\_\_Ag_1 := 1$;
4        $l_0.plan\_f_1 := localplan(l_0, f_1)$;
5        $pred\_of\_l_0 := null$;
6        $Ag_1.findplan(l_0, pred\_of\_l_0)$
7 inform $Ag_2$ failure and terminate
end
```

```
procedure $Ag_1.findplan(v, pred\_of\_v)$
input: a location $v \in LG$ and the predecessor of $v$
outcome: move forward or backtrack or failure
begin
1 ask the unvisited neighbors of $v$ about the truth of $f_1, f_2$,
  and $visited\_\_Ag_2$; select the neighbors based on the answers
  and place them in a list $S_v^1$    // list stored at $v$
2 if $S_v^1$ is empty then if $pred\_of\_v = null$ then inform $Ag_2$
  failure and terminate else move to $pred\_of\_v$; (say $v'$)
  // backtrack and resume $Ag_1.findplan(v', pred\_of\_v')$
3 if there is an $u \in S_v^1$ where $f_2$ is true
4   then $v.son := u$;  move to $u$;  $u.parent := v$;
         $u.visited\_\_Ag_1 := 1$;
         $u.plan\_f_2 := localplan(u, f_2)$; inform $Ag_2$ success
  and terminate
5 else if there is an $u \in S_v^1$ where $u.visited\_\_Ag_2 = 1$
6   then $v.son := u$;  move to $u$;  $u.parent := v$;
  $u.visited\_\_Ag_1 := 1$; inform $Ag_2$ success and terminate
7 else for each unvisited neighbor $u \in S_v^1$ where $f_1$ is true
8          $v.son := u$;  move to $u$;  $u.parent := v$;
           $u.visited\_\_Ag_1 := 1$;
           $u.plan\_f_1 := localplan(u, f_1)$;
           $Ag_1.findplan(u, v)$
        // $Ag_1.findplan(v, pred\_of\_v)$ is suspended here
9     end for
10    if $pred\_of\_v = null$ then inform $Ag_2$ failure and
  terminate else move to $pred\_of\_v$ (say $v'$)
  // backtrack and resume $Ag_1.findplan(v', pred\_of\_v')$
end
```

```
procedure $proc\_\_Ag_2$
input: an initial location $l_k$ and the subformulas $f_1, f_2$ of an
LMA goal formula $f = E(f_1\ U\ f_2)$
outcome: finds a plan for $f$ (if it exists), failure otherwise
begin
1 $cur := l_k$;  set $cur$ as $explored$;  $ask$ $cur$ whether $f_2$ is
  true at $cur$
2 if $f_2$ is true at $cur$
3   then $cur.visited\_\_Ag_2 := 1$;
4        $cur.plan\_f_2 := localplan(cur, f_2)$;
5        $pred\_of\_cur := null$;
6        $Ag_2.findplan(cur, pred\_of\_cur)$
7 if there is no more location to be $explored$ then inform $Ag_1$
  failure and terminate
8 else get to a new location $l$ in $LG$;  goto 1
end
```

```
procedure $Ag_2.findplan(v, pred\_of\_v)$
input: a location $v \in LG$ and the predecessor of $v$
outcome: move forward or backtrack or failure
begin
1 ask the unvisited neighbors of $v$ about the truth of $f_1$ and
  $visited\_Ag_1$; select the neighbors based on the answers and
  place them in a list $S^2_v$   // list stored at $v$
2 if $S^2_v$ is empty then if $pred\_of\_v = null$ then inform $Ag_1$
  failure and terminate else $move$ to $pred\_of\_v$ (say $v'$)
  // backtrack and resume $Ag_2.findplan(v', pred\_of\_v')$
3 if there is an $u \in S^2_v$ where $u.visited\_Ag_1 = 1$
4   then $v.parent := u$;  move to $u$;  $u.son := v$;
  $u.visited\_Ag_2 := 1$; inform $Ag_1$ success and terminate
5 else for each unvisited neighbor $u \in S^2_v$ where $f_1$ is true
6         $v.parent := u$;  move to $u$;  $u.son := v$;
          $u.visited\_Ag_2 := 1$;
          $u.plan\_f_1 := localplan(u, f_1)$;
          $Ag_2.findplan(u, v)$
         // $Ag_2.findplan(v, pred\_of\_v)$ is suspended here
7     end for
8     if $pred\_of\_v = null$ then inform $Ag_1$ failure and
  terminate else $move$ to $pred\_of\_v$ (say $v'$)
  // backtrack and resume $Ag_2.findplan(v', pred\_of\_v')$
end
```

```
procedure $localplan(l, \phi)$
input a location $l$, an LMA subformula $\phi$
output a plan for $\phi$
begin
case: $\phi = l[true]$     return emptyplan
case: $\phi = l[p]$        return $enter_l, exit$
case: $\phi = l[\phi_1]$   // $\phi_1$ is other than $true$ or $p$
          return  $enter_l, localplan(l', \phi_1), exit$   where $l'$
  is a child of $l$
case: $\phi = l[E_s(v1[\phi_1]\ U_s\ C[\phi_2])]$
  Traverse a path through nodes $u$ where $\phi_1$ is true, concatenate
  $enter_u$ and $localplan(u, \phi_1)$ to the $partial\_plan$ (initially
  empty), until a node $C$ is reached where $\phi_2$ is true; return
  $partial\_plan + enter_C + localplan(C, \phi_2) + exit + exit + \ldots + exit$.
  The number of $exit$ actions would be one more
  than the number of $enter$ actions in $partial\_plan$.
  [The operator $+$ signifies concatenation of plans.]
case:  $\phi = E_s(v1[\phi_1]\ U_s\ v2[\phi_2])$
  Traverse a path through nodes $u$ where $\phi_1$ is true, concatenate
  $enter_u$ and $localplan(u, \phi_1)$ to the $partial\_plan$, until a node
  $l'$ is reached where $\phi_2$ is true; return $partial\_plan + enter_{l'} +$
  $localplan(l', \phi_2) + exit + exit + \ldots + exit$. The number of
  $exit$ actions would be one more than the number of $enter$
  actions in $partial\_plan$.
case:  $\phi = \phi_1\ \wedge \phi_2$
  Find the plans for $\phi_1$ and $\phi_2$ using the above cases; return
  the concatenation of these plans.
end
```

The cases in the above procedure are as per the different forms of ϕ given in the syntax of LMA.

Upon termination of the algorithm, any location that lies in the plan path for f will know its parent and son; except the initial location whose parent is null, and the closing location whose son is null. Based on this information, a hop can be easily determined at each location. This is illustrated with an example in Section V-B.

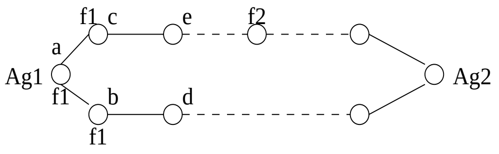


Fig. 6. A Network topology

TABLE I
COMPUTATION STAGES

state	location	information	data
active at a	a	$b : f_1$ $c : f_1$	$S^1_a = \{b, c\}$ $a.son = b$ $a.visited\_Ag_1 = 1$ $local\_plan\_f_1$ at a
suspended at a	moves to b		
active at b	b	$d : \neg f_1, \neg f_2$	$S^1_b = nil$ $b.parent = a$ $b.visited\_Ag_1 = 1$ $local\_plan\_f_1$ at b
terminates at b	moves to a		
suspended code at a resumes	a		
suspended at a	moves to c		$a.son = c$
active at c	c	$e : \neg f_1, \neg f_2$	$S^1_c = nil$ $c.parent = a$ $c.visited\_Ag_1 = 1$ $local\_plan\_f_1$ at b
terminates at c	moves to a		
suspended code at a resumes	a	terminates, reports failure to Ag_2	

A. Computation Stages of the Algorithm

The computation stages for the procedures of the agents is illustrated with a simple network structure as shown in Fig. 6. The agents start from the two ends of the network. There is only one f_2-location and it is located far away from the starting point of Ag_2; f_1 is true only at the locations a, b, and c. Suppose that no plan exists for the goal $E(f_1 U f_2)$. The procedures $Ag_1.findplan, Ag_2.findplan$ are executed concurrently at the locations. The Table I shows the different stages of the computation for $Ag_1.findplan$, where the information is that obtained by the agent after the ask operation, and "data" implies the data at the location. The computation stages of $Ag_2.findplan$ is similar.

B. An Example Illustrating Plan Construction

The working of the algorithm to find a plan is illustrated with respect to a distributed world (DW) shown in Fig. 7 for Example 3 given in Section I. The goal formula is

$$E(v[jvm]\ U\ v[E_s(v[jvm]\ U_s\ v[app])])$$

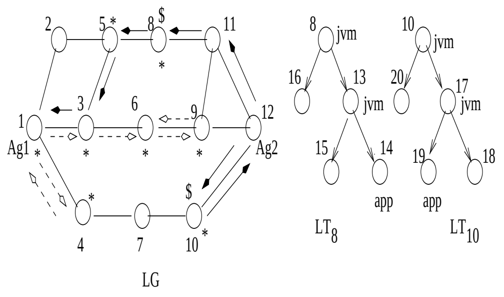


Fig. 7. A Distributed World

DW consists of the locations $l_1, \ldots, l_{12}$ in LG. In the Fig. only the location tree of l_8 and l_{10} are shown. jvm is true at $l_1, l_3, l_4, l_5, l_6, l_8, l_9, l_{10}, l_{13}, l_{17}$. So f_1 is true at these locations (shown by $*$). app is true only at l_{14} and l_{19}. f_2 is true at l_8 and l_{10} (shown by $).

Ag_1 starts from l_1 and Ag_2 from l_{12}. The movement of Ag_2 is shown by lines with bold arrowheads, and that of Ag_1 by broken lines. Ag_1 computes the local plan for f_1 and stores the plan at l_1 ($enter_1, exit$). It marks l_1 visited. Now Ag_1 can select either l_3 or l_4. Let Ag_1 first select l_4. So $l_1.son$ is set to l_4. At l_4 it sets $l_4.parent$ to l_1, finds local plan for f_1. Since no satisfying path exists from l_4, so it returns to l_1. It then selects l_3, updates $l_1.son$ to l_3, moves to l_3, and repeats the steps done at l_1. At l_3 it can select either l_5 or l_6. Let it move to l_6. At l_6 it can move only to l_9, which is a dead-end. So it backtracks. During this time it has not yet come across a location that is visited by Ag_2. Now consider Ag_2's movements. It first moves to l_{10}, where f_2 is true, but comes back to l_{12} since l_{10} is a dead-end. Let it now move to l_{11}. At l_{11} it finds an f_2-location (l_8). The local plan at l_8 is $enter_8, enter_{13}, enter_{14}$. From l_8 it can move only to l_5. At l_8 it sets $l_8.parent$ to l_5. At l_5, it sets $l_5.son$ to l_8, and comes to know that Ag_1 has already visited l_3. Thus it discovers that a plan exists for the goal. So it sets $l_5.parent$ to l_3. It moves to l_3 and updates $l_3.son$ to l_5. It informs Ag_1, who is now at l_6, that a plan is found and terminates.

Thus, a plan is stored in a distributed manner. In order to execute the plan for the given goal proceed follows. First execute the local plan at l_1. Since the son of l_1 is l_3, so perform $hop_{1,3}$. Similarly, execute the local plan at l_3 followed by $hop_{3,5}$. Now execute the local plan at l_5 followed by $hop_{5,8}$, and then execute the local plan at l_8. Thus, the overall plan is $loc_{l_1}, hop_{1,3}, loc_{l_3}, hop_{3,5}, loc_{l_5}, hop_{5,8}, loc_{l_8}$.

It is easy to see that Ag_1 always moves through f_1-locations and a prefix of a plan for f exists in at least one of those locations. Ag_2 after finding an f_2-location always moves through f_1-locations and a suffix of a plan for f exists in at least one of those locations. Thus, if an agent combines its plan path with that of the other agent, the resulting is always a plan path for f.

VI. Properties of the Algorithm

A. Complexity

Theorem [Time complexity] The worst case time complexity of the algorithm is $O(e + M \cdot |f|)$ where e is the number of edges in LG, $|f|$ is the length of an LMA goal formula, and M is the total number of locations in all the location trees.

Proof: The worst case time complexity of the algorithm is the sum of the worst case time complexity of the agent's procedures. Since the procedures of the two agents are identical so the time taken by any one agent can be computed. Now the time spent by an agent is in three ways: (i) time spent in communicating, (ii) time spent in traversing LG, and (iii) time spent in location trees.

It is assumed that it takes at most one unit of time to send (receive) a query (an answer) over each edge, for the communication between an agent and a location. Thus communication takes constant time.

The total number of $move$ operations made by an agent in the worst case is $2 \cdot e$, because of backtrack. So this becomes $O(e)$.

For (iii) the following is to be considered (a) the time taken to compute the truth of subformulas, and (b) the time taken to compute local plans.

The (a) part: The truth of the subformulas is obtained using the model checking algorithm [8], that takes $O(m' \cdot |f|)$ time, where m' is the maximum number of nodes in any location tree. A location computes the truth of a subformula once, and since there are at most n locations in LG, so the total time spent is $O(M \cdot |f|)$, where $M = n \times m'$.

The (b) part: i.e., the time taken by $localplan(l, \phi)$.
Let $\phi = \phi_1 \wedge \ldots \wedge \phi_k$. To find a plan for each ϕ_i the entire location tree has to be traversed in the worst case. Since $|\phi|$ is at most $|f|$, so the total time taken in any location tree is $O(|f| \cdot m')$. Since this may be performed at the n locations, so the total time becomes $O(M \cdot |f|)$.

Thus the time complexity is $O(e+M\cdot|f|)$. Q.E.D

Theorem [Space complexity] The space needed by any location in LG in the worst case is $O(n + |f| \cdot m')$, where m' is the maximum number of nodes in any location tree.

Proof: The space is needed for recording visited information of the agents, parent/son information, local plans, and information of the neighbors.

Visited information is one bit of information. The information of parent/son requires one bit of information each. Each location can have at most $n - 1$ neighbors. Each local plan has size $O(|f| \cdot 2 \cdot m')$ where m' is the maximum number of nodes in any location tree. Thus the total space needed by any location is $O(n+|f|\cdot m')$. Q.E.D

Theorem [Communication complexity] The total number of messages transfered in the algorithm is at most $6 \cdot e + 2 \cdot n + 2 \cdot M \cdot |f|$.

Proof: This involves computing the number of messages passed by the agents. The agent Ag_1 does message passing for three purposes: (1) [for communication] The number of neighbors of a node is equal to the degree of a node. Since at each step the agent communicates with only the unvisited neighbors, so the sum of all such neighbors is e. This is because an agent needs to talk with a location more than once to know the visited information of Ag_2. So the total number of messages(sending and receiving) is $2 \cdot e$;

(2) [for movement in LG] the agent uses a depth first search, and in the worst case it needs to explore the entire world. Then the number of movements become $2 \cdot e$;

(3) [for movement in location trees] to find a local plan each location tree is traversed $m' \cdot |f|$ times and finally the agent comes back to the root of the tree. So the total number of messages needed is $2 \cdot m' \cdot |f|$. So the total number of messages in the location trees is the sum of the messages needed in each tree. This becomes $2 \cdot M \cdot |f|$.

Thus total number of messages for Ag_1 is $4 \cdot e + 2 \cdot M \cdot |f|$.

For Ag_2 messages are passed for: (1) talking with at most n locations to know the truth of f_2: so the number of messages is $2 \cdot n$, and (2) movement for locating an f_2-location: which is $2 \cdot e$ Thus the total number of messages needed is at most $6 \cdot e + 2 \cdot n + 2 \cdot M \cdot |f|$. Q.E.D

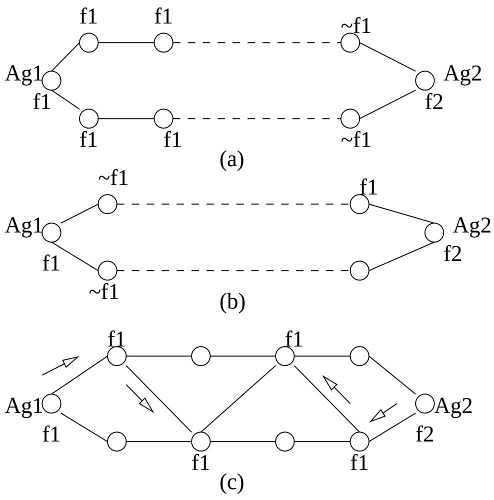


Fig. 8. Cooperative agents

B. Some Other Features of the Algorithm

Each mobile agent performs a distributed computation to obtain a plan. Thus the overall algorithm is distributed in nature. A plan is stored in a distributed manner. A plan found by the algorithm is sound, i.e., the plan path, for the plan generated, satisfies $f_1\ U\ f_2$. The planning algorithm is complete, i.e., if there exists a plan for an LMA goal formula, then the algorithm is guaranteed to find the plan. The algorithm obtains plans by using little communication among the mobile agents (Ag_1, Ag_2). However it has to communicate with the locations.

Some drawbacks of the approach are: (i) not all local plan computations are useful and (ii) in the worst case, a single agent can find a plan all by itself. Some situations are listed, below, where both the agents play significant roles and thus influence the total time of completion of the algorithm. This is illustrated with a simple topological structure of the network, that can easily be generalized to other structures. Refer to Fig. 8, where only the location graph is shown. In (a) and (b) it is assumed that there is only one f_2-location, and that the agents start from the two ends. In (a) and (b) failure is detected after one step. If a single agent is used, then in both (a) and (b) Ag1 and Ag2 would have taken the worst case time respectively to detect failure. In (c) both the agents perform equal work. So in this case the total time taken is halved. In (c) the movements are shown by arrows.

C. Extension of the Algorithm

The adaptability of the algorithm, given in Section V, for slightly more complicated LMA goals is now considered. Let a goal formula with nested Es be of the form: $E(v[\phi_1]\ U\ (C[\phi_2]\ \wedge\ E(v[\phi_3]\ U\ C'[\phi_4])))$
This is handled in the following manner. First compute the subgoals: by parsing the formula and then reducing it to an equivalent formula. In this case the subgoals are $g_1 = E(v[\phi_1]\ U\ C[\phi_2 \wedge \phi_3])$ and $g_2 = E(v[\phi_3]\ U\ C'[\phi_4])$. Now Ag_1 works as follows: it first tries to achieve g_1, so the closing location is named C; now from C it tries to achieve g_2. This can be easily done by $Ag_1.findplan$.

Ag_2 works as follows: it first gets to a location named C', if ϕ_4 holds at C', it goes through locations where ϕ_3 holds until it comes across a location that is already visited by Ag_1 or that is named C. This can be easily done by $Ag_2.findplan$.

VII. Related Work

In [16] plans are pre-computed and fed to the agent. Complete knowledge about the topology of the network is assumed in [16] to compute plans. A partially observable network is considered in this paper where plan computation cannot be done a priori. However, a planning problem like the traveling salesman problem is considered in [16]. Such types of goals are not expressible in LMA and thus cannot be solved by the algorithm proposed in this paper.

This paper did not consider finding shortest paths. However, this may be useful in some applications. Now a previously established result is discussed. The problem of finding shortest paths with unknown geography is considered in [18]. They consider a layered graph of width two–for example the network in Fig. 8(c). It is proved that for such a graph the ratio of distance traveled to the optimal length is nine. Such a ratio is an important criterion to evaulate the performance of an agent. For layered graphs of unbounded width no fixed ratio is possible [18]. Real world networks are considered in this paper that are not necessarily layered graphs of width two. So the above performance criterion cannot be applied here.

In some situations the only way to decide for a plan is by performing actions (acting) in the world. This is best illustrated by the popular *Omelette* domain that consists of some good and bad eggs (at least 3 are good), a bowl and a saucer; the task is to get 3 good eggs. So an egg has to be broken to find out whether it is good or bad. Classical planners–that produce a sequence of actions–cannot address this problem. Sensing actions is proposed in [15] to attack problems of similar nature. In reactive planning acting may sometimes be necessary [3], [23]. In [23] failure of actions is considered and a dynamic logic is proposed to capture failure and sensing. This is beyond the scope of this paper. A real-world challenging power supply restoration domain [24] is another scenario where acting is a necessary precondition for information acquisition. Thus the work in this paper is similar to these works since a mobile agent has to roam across the network to acquire information.

Distributed depth-first-search (DDFS) algorithms have been widely studied. One of the best known DDFS algorithm is reported in [22] that has both time and message complexity $O(n)$, where n is the number of nodes in the network. The planning problem considered in this paper is different from the DDFS problem. The comparison is made for two reasons: (i) since the search techniques are DFS and (ii) to illustrate why the complexities are more in the algorithm presented here. The algorithm in [22] obtains the complexity by using messages of length $O(n)$ which allows global information of the network. The algorithm presented here uses fixed size messages, performs additional steps beside exploration– for example computation of truth of subformulas and local plans. It also uses communication with neighbors to chose the next move for the agents, which is not needed in the DDFS algorithm. Thus more messages are needed by this algorithm.

VIII. Conclusions and Future Work

A hierarchical computer network is considered where mobile agents are supposed to achieve network management tasks. The logic LMA is presented that can elegantly specify such tasks. This paper made an endeavor to justify the importance of finding plans for mobile agents. A distributed planning algorithm using two collaborative mobile agents is developed when the tasks are expressed in LMA.

Although there has been some work on planning for mobile agents, those works did not really address or deal with planning issues. This paper makes a first attempt in this direction. The hope is that the theory developed will lead to the implementation of a fully functioning system. Some interesting issues that need to be addressed are listed below. First, to come up with a logical language to express a much wider variety of tasks that a mobile agent is usually asked to achieve. Second, faults are common in networks. How can the planning problem be addressed in such scenarios?

Some future works that are planned include: (i) automatic generation of sub-formulas for complicated LMA formulas with nested Es, (ii) to obtain optimal number of agents needed for a task, and (ii) to obtain the lower/upper bounds on the communication complexity to achieve a task.

Acknowledgement

The author thanks Prof R. Ramanujam of Institute of Mathematical Sciences (IMSc) Chennai, India, for his comments and suggestions on a preliminary version of the work.

References

[1] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.
[2] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning with temporal goals. In *Proceedings of IJCAI*, pages 509–514, 2001.
[3] M. Beetz and D. McDermott. Improving robot plans during their execution. In *Proceedings of AIPS*, 1994.
[4] P. Bellavista, A. Corradi, C. Federici, R. Montanari, and D. Tibaldi. Security for mobile agents: issues and challenges. *Handbook of mobile computing: I. Mahgoub and M. Ilyass (eds)*, 2004.
[5] A. Bieszczad, B. Pagurek, and T. White. Mobile agents for network management. *IEEE Communications Surveys*, 1998.
[6] A. Blum and L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
[7] L. Cardelli. Wide area computation. In *Proceedings of ICALP, LNCS 1644*, pages 10–24, 1999.
[8] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions of Programming Languages and Systems*, 8(2):244–263, 1986.
[9] J.E. Cook. Software engineering concerns for mobile agent systems. In *Proceedings of the Workshop on Software Engineering and Mobility*, 2001.
[10] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.
[11] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, 2004.
[12] F. Giunchiglia and P. Traverso. Planning as model checking. In *Proceedings of European Conference on Planning*, pages 1–20, 1999.
[13] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *Artificial Intelligence*, 14:253–302, 2001.
[14] D. Kotz and R. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(2):7–13, 1999.
[15] H. Levesque. What is planning in the presence of sensing? In *Proceedings of AAAI*, 1996.
[16] K. Moizumi. Mobile agents planning problem. In *PhD thesis, Dartmouth College*, 1998.
[17] A. Murphy and G. Picco. Reliable communication for highly mobile agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 5(1):81–100, 2002.
[18] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.
[19] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proceedings of IJCAI*, pages 479–486, 2001.
[20] P. Rodriguez, C. Spanner, and E.W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE Transactions on networking*, 9(4):404–418, 2001.
[21] J.A. Sauter, R. Matthews, H. van dyke Parunak, and S. Brueckner. Evolving adaptive pheromone path planning mechanisms. In *Proceedings of AAMAS*, 2002.
[22] M. Sharma, S. Iyengar, and N. Mandyam. An efficient distributed depth first search algorithm. *Information Processing Letters*, 32:183–186, 1989.
[23] L. Spalazzi and P. Traverso. A dynamic logic for acting, sensing, and planning. *Logic Computation*, 10(6):727–821, 2000.
[24] S. Thiebaux and M. Cordier. Supply restoration in power distribution systems–a benchmark for planning under uncertainty. In *Proceedings of ECP*, 2001.
[25] P. Wojciechowski. Algorithms for location-independent communication between mobile agents. In *Proceedings of AISB '01 Symposium on Software Mobility and Adaptive Behaviour*, 2001.

Rajdeep Niyogi was born in Calcutta, India, in 1971. He received his Bachelor of Engineering degree in Electrical Engineering with Honours in 1994 from Jadavpur University, India. He received his Master of Engineering degree with specialization in Computer Engineering in 1998 from Jadavpur University, India. He received his Ph.D. in Computer Science and Engineering in 2004 from Indian Institute of Technology (IIT) Kharagpur, India. He is currently working as an Assistant Professor at the Electronics and Computer Engineering Department, IIT Roorkee, India. He is a member of ACM. His fields of research are automated planning, formal methods, and distributed computing.