FAST RECALL OF REUSABLE FUZZY PLANS USING ACYCLIC DIRECTED GRAPH MEMORY

Moataz Ahmed*, Ernesto Damiani**, and David Rine***

* SONEX Enterprises, Inc., 9990 Lee Highway, Fairfax, VA 22030, USA. E-mail: moataz.ahmed@sonexent.com
** Università di Milano, Polo Didattico e di Ricerca di Crema, Italia. E-mail: edamiani@crema.unimi.it
*** Computer Science Dept., George Mason Univesity, Fiarfax, VA 22030, USA. E-mail: drine@cs.gmu.edu

KEY WORDS: Planning, Similarity Metrics, Plan Reuse, Autonomous Real-Time System.

ABSTRACT

A planning capability is one of the important features that autonomous real-time systems must have. Fuzzy-based planning is more appropriate for planing in real-time dynamic environments such as vehicle navigation and patient monitoring. Planning in such environments needs to be performed as quickly as possible. Planning can be made quicker by reusing portions of similar previous plan segments to efficiently derive a new plan. Planning problems, then, include at least two sub-problems. First, the problem of efficiently and effectively generating a plan from scratch. Second, the problem of efficiently and effectively retrieving a plan suitable to be reused and then repairing it to fit the new situation. This paper presents a memory structure as well as fuzzy-based similarity metric for efficiently and effectively retrieving plans specified using fuzzy logic linguistic variables. In the paper, an *acyclic directed graph* (ADG) model is proposed for memory, such that each node represents an intermediate step in the execution of plans that are represented by other nodes connected to it. Examples of similarity measures computations are presented.

1. INTRODUCTION

The ability to act appropriately in dynamic environments is critical to the survival of all living creatures. A long standing problem in the field of automated reasoning is that of designing systems which can describe a set of actions (plan) that can be expected to allow the system to reach a desired goal. Ideally, the set of actions so produced is then passed on to a robot, a manufacturing system, or some other form of effector, which can follow the plan and produce the desired output.

Within the artificial intelligence community, there exist a number of competing paradigms for planning. This paper is concerned with one of these paradigms: case-based planning. In this paper, we present a case-based planner architecture that can effectively and efficiently uses its old experiences (plans). The paper emphasizes on plans storage and retrieval aspects.

2. BACKGROUND

2.1. Planning Systems

The problem of generating a sequence of actions to accomplish a goal is referred to as *planning*. Normally, each action has a set of *pre-conditions* which must be satisfied before the action can be applied, and a set of *post-conditions* which will be true after the action execution. A *planning problem* is characterized by an initial state and goal statement. A *state* is a collection of characteristics of an object that is sufficiently detailed to uniquely determine the new characteristics of the object that will result after an action. The initial state description tells the planning system the way the *world* is "right now". The *goal statement* tells the planning system the conditions which must be satisfied when the plan has been executed. The world in which the planning takes place is often called the *application domain*. We will sometimes refer to the goal statement as simply *the goal*. A *plan* is an organized collection of actions. A plan is said to be a solution to a given problem if it is applicable in the problem's initial state, and if after plan execution, the goal is true. A plan is applicable if all the preconditions of any action, within the plan, are satisfied before applying that action. In many planning systems, a goal may be transformed into a set of other, usually simpler, goals called *subgoals*.

Knowledge-based planners are class of AI problem solving systems that are broadly characterized by the fact that they apply a domain knowledge to search heuristically through a space of possible actions to find a sequence of actions that will achieve a goal. A brief overview on the principal paradigms for knowledge-base planning. Examples of such paradigms found within the AI community can be found in Lehner [7].

2.2. Intelligent Agents

In the last few years, there has emerged and explosion of interest in the development of *intelligent agents* (IAs) [1], [7], [10]. Although the term *intelligent agent* does

not have a precise or agreed upon definition, IAs usually refer to robust programs that communicate with other entities to gather information and make decisions. A *multi-agent system* (MAS) can be thought as: "*...several autonomous intelligent agents which 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.*" [3].

Different types of agents may compose the MAS, each of them having different reasoning capabilities on his or others' actions and beliefs. Three broad classes of agents are [9]:

- *Reactive agent*: It reacts to messages from other agents and can not reason about his or others' beliefs and actions. It acts on pre-defined plans and can send messages to other agents.
- *Intentional agent*: It can reason about the agent's beliefs and actions, create plans and execute them. It can not reason about others' beliefs and actions.
- *Social agent*: An intentional agent with explicit models of other agents and the capability to maintain these models. It can adapt and plan the agent's actions with respect to others' plans and actions.

2.3. Autonomous Real-Time Systems

Designing a system for totally autonomous operation is a difficult task which encompasses several aspects such as planning, sensory-motor controlling, and adapting. A major advantage of using fuzzy logic for designing totally autonomous systems is that fuzzy logic can allow different problem solving techniques to cooperate and exchange information to solve a global task. For example, in autonomous vehicles domain, fuzzy logic allows automated planning techniques to cooperate with fuzzy logic control techniques [8] to drive a vehicle from one location to another.

Ahmed and Rine [2] have introduced a fuzzy-based framework for designing totally autonomous real-time systems. In this fuzzy-based framework, a state of the application domain is represented by predicates with associated values in a form of fuzzy sets (as opposed to the *TRUE* and *FALSE* crisp values). Each predicate describes some property in the application domain in its present state. Each predicate may or may not have arguments. For example the predicate *SPEED(R)* with one argument *R* represents the speed of the object *R*.

The fuzzy-based framework is based on cooperative intelligent agents that communicate through a common *blackboard*. The framework is comprised of four intelligent agents. These agents are: 1) *Perception Agent*, 2) *Planning Agent*, 3) *Control Agent*, and 4) *Identification Agent*.

The Perception Agent is concerned with understanding the world (application domain) where the autonomous system operates. According to the given available resources to the autonomous system, and according to the given desired objectives of the autonomous system, the Planning Agent sets a method (a plan) that efficiently and effectively uses available resources to achieve the desired objectives. The Planning Agent transforms objectives, together with a perception of the world, into a course of action which is expected to achieve these objectives.

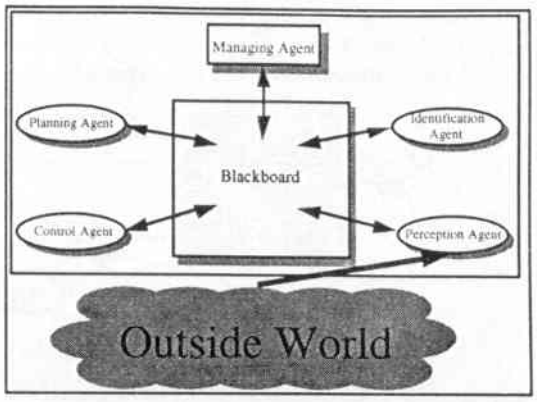

Figure 1. Agent-Based Framework for Designing Totally Autonomous Real-Time Systems.

As in *Command, Control, Communication, and Intelligence (C^3I)* systems [5], the Planning Agent operates in two modes, corresponding respectively to the preparatory and execution phases of a mission:

- *Mission planning*: Searching for and selecting a plan which is expected to achieve the objectives.
- *Effectiveness monitoring*: Continually reassessing the suitability of the promulgated plan.

On the other hand, the Control Agent transforms the required course of action (i.e., plan), together with a perception of the world, into resource manipulation orders such that the actual resource activity complies with the required resource activity. As one can see, the Control Agent is also a planner.

The purpose of the Identification Agent is to continually identify the behavior of the resources controlled by the system. This continuos identification process allows the Planning Agent and the Control Agent to continuously adapt their plans in order to accommodate unpredictable environment changes, whether these changes arise within the resources or external to them. The Managing Agent works as a managing unit (scheduler), for example for concurrency control.

2.4. Case-Based Planning

This paradigm is founded on the recognition that an effective planner does not enter a planning problem from scratch. The planner will usually have a store of template plans or historical cases from which a planner can retrieve plans that have worked in similar situations.

Real-time performance requires that the system respond before the environment can change substantially. Planning in real-time dynamic environments such as vehicle navigation and patient monitoring has to be continuos and needs to be performed as quickly as possible. Planning can be made quicker by reusing portions of similar previous plan segments to efficiently derive a new plan. For continuous quick planning, Ahmed and Rine [2] have introduced an architecture for a case-based planner (see Figure 2). This architecture is a modified version of the architecture proposed by Hammond [4].

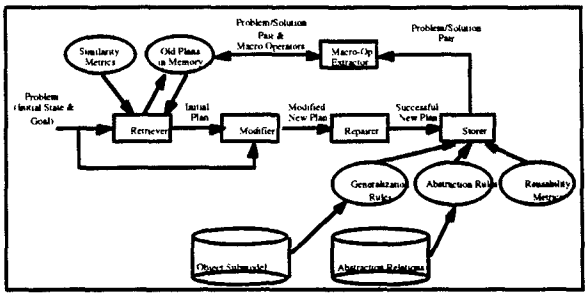

Figure 2. Case-Based Planner Architecture.

The following is a description of the major elements of the Case-Based Planner architecture:

- *Retriever* - Responsible for searching for past problem-solution pairs in the memory. The Retriever searches for a past case that is similar to the new case using similarity metrics. A similarity metric is a measurable characteristic that can be used to decide how similar two planning problems are.
- *Modifier* - Responsible for modifying the solution which has been retrieved from the memory by instantiating the solution parameters with the new problem parameters. In case the degree of similarity is not 100%, some error may arise while solving the new problem.
- *Repairer* - is responsible for fixing the error using any other planning technique.
- *Storer* - Responsible for storing successful plans into the memory for future planning. It is also responsible for assuring that the memory is not overwhelmed by an ever-increasing number of stored plans. For the stored plans to be reused efficiently and effectively, the Storer generalizes, abstracts, and measures the reusability of plans before storing them into the memory.
- *Abstraction rules* - are rules extracted from an abstraction model or abstraction relationships of the application domain. They are used to abstract the successful plans.
- *Generalization rules* - are rules extracted from the object submodel of the domain model of the application domain. They are used to generalize the abstracted successful plans.
- *Macro operator extractor* - is the process of reengineering a successful plan details (which constitute the subplans in the abstract space) so that they can work as macro operators. The rationale behind this process is the well known fact that the use of appropriate subgoals can greatly reduce the amount of search necessary to solve a problem [6]. It is often the case that given a collection of subgoals, previously satisfied subgoals must be violated in order to make further progress towards the main goal regardless of the solution order. Such a collection of subgoals are called non-serializable. The reason that human problem solvers establish and solve these subgoals is that they know what sequence of operators to apply to solve the next subgoal while not violating already satisfied subgoals. Such a sequence of primitive operators is called a macro operator. Progress toward the solution in this context is getting to a state from which the problem solver knows macro operators that will achieve the next subgoal.
- *Reusability metrics* - are measurable characteristics that can be used to measure the reusability of plans. As an example of a reusability metric is the plan cohesion.

3. PROBLEM STATEMENT

Key to case-base planning techniques is remembering. Remembering has two parts: integrating problem-solution pairs into memory when they happen and recalling them in appropriate situations later on. The problem addressed in this paper is concerned with the memory structure of the problem-solution pairs, and the similarity metrics that are used by the Retriever to find in memory the problems that are similar to the new problem.

4. THE MEMORY STRUCTURE

A plan can be recursively defined in a *Backus Naur Form (BNF)* as follows:

<PLAN> ::= <ACTION> |
<PLAN> <ACTION> |
<ACTION> <PLAN>

<ACTION> ::= Any of the available actions in the application domain.

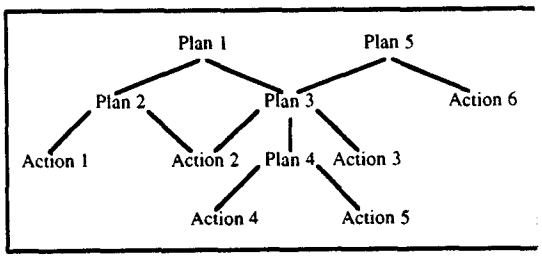

Figure 3. ADG Memory: An Example.

A plan can be stored in a form of a tree structure. However, since some actions, some plans, or some portions of plans are included in more than one plan, we propose an *acyclic directed graph* (ADG) model for problem-solution pairs memory, such that each node represents an

intermediate step in the execution of plans that are represented by other nodes connected to it.

One can look at a plan, which is a sequence of actions, as a *macro* that has its own pre-conditions and post-conditions. The pre-conditions of a plan is extracted from its initial state. These pre-conditions are the set of predicates that appear in the initial state and used during the plan execution. Hence, plans in memory can be treated exactly the same way as actions. Plans as well as actions are stored as shown in Figure 4. *Plan-ID* is either the plan or the action identifier. *Pre-Conditions* and *Post-Conditions* are represented in unquantified predicates of the *first-order predicate logic*.

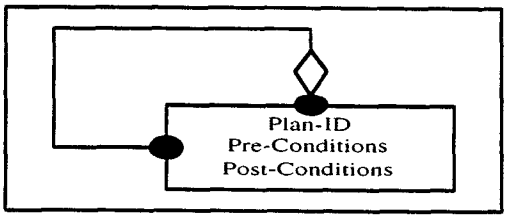

Figure 4. ADG Memory Structure.

The notation used in Figure 4 is the *Object Modeling Technique (OMT)* notation. The "◊" symbol denotes an *aggregation (part-of)* of relation. The "•" symbol denotes *"many (zero or more)"* class.

5. SIMILARITY METRICS

Let *OR* be the set of all predicates that appear in the pre-conditions of an old problem-solution pair. Let *OS* be the set of all predicates that appear in the post-conditions of an old problem-solution pair. Let *NI* be the set of all predicates that appear in the initial state of the new problem. Let *NS* be the set of all predicates that appear in the goal statement of a new problem.

In order to use the proposed metric, each past problem-solution pair is indexed by the corresponding pre-conditions predicates and post-conditions predicates. Each predicate is assigned an *importance value (IV)* in the interval [0,1]. This value, that can be modified by a feedback mechanism, expresses how much the predicate is important for the overall execution of the plan. When a new problem is presented to the system in terms of its initial state's predicates and goal statement predicates, the system looks for old problem-solution pairs that have high similarity with the new problem. Intuitively, we require that

$$OR \subseteq NI, \text{ and}$$

$$OS = NS.$$

For each old problem-solution pair PS_j in the memory, a predicate similarity measure PSM_{ji} for each predicate $P_i \in OR_j$, $P_i \in OS_j$, and $P_i \in NS$ is calculated as follows. Note that P_i represents the predicate (e.g. *SPEED (R)*). Recall that each predicates have an associated value in a form of fuzzy sets F_i. Two sets of predicates (e.g., *OS* and *NS*) may have the same predicate (e.g., *SPEED(R)*), while the fuzzy sets that represent the predicate in each one of the sets may be different. For instance, the fuzzy set $F_{i,OS}$ that represents *SPEED(R)* in *OS* may be different than the fuzzy set $F_{i,NS}$ that represents *SPEED(R)* in *NS*.

$$PSM_{ji} = \begin{cases} x + \mu_i (1 - x), & \forall P_i \in (OR_j \cap NI) \\ 0, & \forall P_i \in (OR_j - NI) \end{cases}$$

$$PSM_{ji} = \begin{cases} x + \mu_i (1 - x), & \forall P_i \in (OS_j \cap NS) \\ 0, & \forall P_i \in (OS_j - NS) \cup (NS - OS_j) \end{cases}$$

where $x \in [0, 1]$ is a constant, and μ_i is the maximum degree of membership that belongs to the fuzzy set that is the result of the fuzzy intersection $F_{i,S1} \cap F_{i,S2}$. For $P_i \in (OR_j \cap NI)$, $S1 = OR_j$ and $S2 = NI$. For $P_i \in (OS_j \cap NS)$, $S1 = OS_j$ and $S2 = NS$. x is used to differentiate between the case in which $P_i \in ((OR_j \cap NI) \cup (OS_j \cap NS))$ and $\mu_i = 0$, and the case in which $P_i \in ((OR_j - NI) \cup (OS_j - NS) \cup (NS - OS_j))$. x may be adjusted according to the performance. Note that each predicate has arguments. The intersection used here is based on instantiation. For example, we may find *SPEED(O1)* in *OS* and *SPEED(L1)* in *NS*. The predicate belongs to the intersection if *O1* can be instantiated (replaced) with *L1* according to the object model of the domain model.

The overall similarity measure SM_j between the new problem and the old problem-solution pair PS_j is calculated as follows:

$$SM_j = \max_{PR} \frac{\sum_i IV_i \, PSM_{ji}}{\sum_i IV_i}$$

where *PR* is the possible permutations to map the arguments of the predicates. The system does a permutation to map the arguments of the new problem initial state and the new problem goal statement with the arguments of the old-problem preconditions and the old problem goal-statement. The combination that gives the maximum value is used and such a value represents the overall similarity measure between the two problems.

6. ILLUSTRATIVE EXAMPLE

Using fuzzy logic, states of the application domain can be described by fuzzy sets. Examples of such fuzzy sets in autonomous vehicles domain are: *SPEED(O1)*, *DISTANCE(O1,O2)*, *ICY_ROAD*, etc. Where *SPEED(object)* gives the speed of an object as a fuzzy set, *DISTANCE(object1, object2)* gives the distance between two objects as a fuzzy set, *ICY_ROAD* is a fuzzy set that represents how icy the road is. It may be the case that the vehicle can have access to an accurate measure of its speed, i.e., it can have its speed as a crisp value. However, this is not [illegible] when trying to estimate the speed of

other objects. This is an advantage of using fuzzy sets to represent the predicates that describe the state of the world as opposed to using crisp predicates (e.g., predicates of the *first-order predicate logic*).

As an illustrative example of a similarity measure computation, consider the following:

Old problem-solution pair in memory:

- Pre-conditions:
  Predicate1(O3,O2), Predicate2(O2,O1), Predicate3(O1,O4), Predicate4(O5), Predicate5(O6)
- Post-conditions:
  Predicate1(O1,O2), Predicate2(O2,O3), Predicate3(O3,O4)

New problem:

- Initial State:
  Predicate1(L1,L2), Predicate2(L2,L3), Predicate3(L3, L4), Predicate4(L5), Predicate6(L7)
- Goal Statement:
  Predicate1(L1,L2), Predicate2(L2,L3), Predicate3(L3,L4)

Assume that the values (fuzzy sets) of the predicates : *Predicate1, Predicate2,* and *Predicate3* are the same in both the *OR* and *NI*, and *OS* and *NS*. This implies that μ = 1 for those predicates. Then, using the following instantiation: *L1/O1, L2/O2, L3/O3,* and *L4/O4*, we get *PSM* = 1 for *Predicate1, Predicate2,* and *Predicate3* in *OS* and *NS*. On the other hand, we get *PSM* = 0 for *Predicate1, Predicate2,* and *Predicate3* in *OR* and *NI*. Assume that values for *Predicate4* in both *OR* and *NI* are as shown in Figure 5.

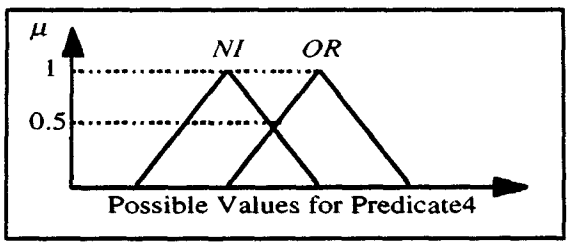

Figure 5. Values of Predicate4 in both *NI* and *OR*.

Figure 5 shows that the fuzzy intersection has a maximum degree of membership μ = 0.5. Assume that x = 0.5, then for *Predicate4* (with *L5/O5* instantiation), *PSM* = 0.5 * 0.5 * 0.5 = 0.75. On the other hand, *Predicate5* has a *PSM* = 0.0 (there is no match). This imply that the overall *SM* = (6.75/11) = O.61. Note that the instantiation (*L1/O1, L2/O2, L3/O4, L4/O4,* and *L5/O5*) gives the maximum *SM* which is 0.61.

7. Conclusion

This paper has described a mechanism for retrieval of reusable fuzzy-based plans in a repository storing successful problem-solution pairs. The proposed method is based on a memory structure that has a form of acyclic directed graph (ADG), and a reusability metric that measures how close the new problem and an already existing problem-solution pair are.

Future works will focus on conducting experimental case studies to give a feeling of how the proposed method is performing. Future work will also investigate the effect of changing the value of x on the performance. It will also investigate the effect of assigning a constant value $y \in [0, 1]$ to any two predicates $P_i, P_j \in (OR - NI)$ or $P_i, P_j \in (OS - NS) \cup (NS - OS)$, but P_i and P_j have the same name. Example of such P_i and P_j is *Predicate1(O3, O2)* in *OR* and *Predicate1(L1, L2)* in *NI* in the illustrative example. above.

8. REFERENCES

[1] ACM (1994). *Special Issue of Communications of the ACM on Intelligent Agents.* July.

[2] Ahmed, M., and Rine D., (1997). "A Reusable Intelligent Autopilot: A Framework", Accepted for publication, *International Journal of Applied Software Technology (IJAST).*

[3] Demazeau, Y., and Muller, J. (1989). "Decentralized Artificial Intelligence." In *Proceedings of the First European Workshop on Modeling Autonomous Agents in Multi-Agent World,* North-Holland, August.

[4] Hammond, K. J. (1989). *Case-Based Planning.* Academic Press, Inc.

[5] Harris, C.J. and White, I. (eds.), (1987). *Advances in Command, Control & Communication Systems.* Peter Pererinus Ltd., London, United Kingdom.

[6] Korf, R. E., (1990). "Planning as Search: A Quantitative Approach." In *Readings in PLANNING,* J. Allen, J. Hendler, and A. Tate (EDS.), Morgan Kaufmann Publishers, Inc.

[7] Lehner, P. E. (1996). *An Introduction to Knowledge-Based Planning Paradigms and Approaches.* George Mason University.

[8] Mamdani, E.H. (1993). "Twenty years of fuzzy control: experiences gained and lessons learnt." In *1993 IEEE International Conference on Fuzzy Systems* (San Francisco, CA, March 28-April 1). IEEE, Piscataway, N.J., pp.339-344.

[9] Moulin, B., and Cloutier, L. (1994). "Collaborative work based on multiagent architectures: A methodological perspective." In *Soft Computing: Fuzzy Logic, Neural Networks, and Distributed Artificial Intelligence.* Prentice Hall.

[10] Wooldridge, M., and Jennings, N. (eds). (1995). *Intelligent Agents.* Springer-Verlag.