The Use of Plans in Knowledge Management Tasks

Louise Pryor

Harlequin Ltd
Technology Transfer Centre,
King's Buildings,
Edinburgh EH9 3JL, UK.
louisep@harlequin.co.uk

Abstract

This paper discusses how a plan execution system could be used to assist in a knowledge management task, such as searching a set of databases for a relevant document. PARETO is a plan execution system based on the as-needed expansion of a hierarchy of sketchy plans. It has been adapted for use in several different domains. There are several problems that would arise in adapting PARETO to assist in a knowledge management task such as searching for a relevant document in a proprietary intranet. However, there are also advantages that would be provided by the use of such a system.

Introduction

There are many tasks that involve the skilled analysis of large amounts of information. Examples include a manager analysing sales figures; an actuary setting the loss reserves for a casualty insurance company; an environmental scientist analysing data from satellites; an analyst preparing standard data files from miscellaneous inputs; and anybody searching for information on the internet or an intranet. These tasks are all performed in a similar manner. Usually, there is a (possibly very abstract) framework around which they are structured: the details are filled in as the task progresses, based on information that is learned during the task execution. Sometimes, the whole course of the task changes from that originally anticipated, again, based on circumstances encountered along the way.

This short paper describes how these tasks can be seen as *plan execution* tasks; it discusses how an opportunistic plan execution system could be used to assist in their performance. The problem of finding information stored in Lotus Notes® is used as a running example, and is described in the remainder of this section. The following sections briefly discuss how AI planning techniques are applicable to this and other knowledge management tasks, with particular reference to PARETO, a plan execution system; and discuss the advantages and disadvantages of using such a system.

Finding information in Lotus Notes

Lotus Notes organises documents in *databases*. In order to find a particular document you first have to find the relevant database and open it. You cannot tell what documents are in a database without opening it; and in order to open it you have to add it to your personal *workspace* if it is not there already.[1] Some databases may require a password. The commands that are needed in order to open a particular database thus depend on both the database and the state of the user's workspace.

Once a database has been opened there are many different ways in which it can be viewed. The possible *views* vary between databases, but typically include such options as `by author`, `by category`, `by topic`, or `by time created`. Some databases have much more specialised views. Complex views are often arranged hierarchically: a database with information about the people in a company, for example, may have views such as `Employee/by first name`, `Employee/by last name`, `Employee/by group`, `Employee/by email`, `Telephone numbers/by person` and `Telephone numbers/by room`. In such a database the information about each employee would be represented as a separate document.

It is common practice to have a database available that lists all the databases on a particular network. Again, this database catalogue can be viewed in several different ways: `by server`, `by path`, or `by title`, for example. Finding a document whose name and location are not known often involves viewing the database catalogue in various ways in order to find a likely looking database, opening it, and looking through its various views for potentially useful documents. The specific commands used will vary according to the databases that are looked at and the subject matter that is being sought.

Suppose that you are just starting a project involving knowledge management, and want to find out what techniques have been used in other projects in the company. Your overall approach is likely to consist of finding a

[1] Regular Lotus Notes users will recognize that this description is somewhat simplified.

likely database, looking at its contents through a number of different views, and then browsing some of the documents in the database (if any of them look likely). You may also follow some cross reference links to other documents (possibly in other databases), before trying the first step, of finding a likely database, again. The exact details of your search cannot be predicted in advance, although this overall outline is very stable across episodes. Moreover, there are a number of subtasks that are often performed: adding a database to your workspace, opening a database, choosing a way in which to view a database, performing full text search on a document, and so on. In fact, there are very few subtasks that are unique to a particular set of circumstances, although the exact key strokes used to perform them will indeed be unique.

Planning

A good working definition of planning is that it is the construction of a set of actions that, if executed, will lead to the achievement of a goal. Traditionally, planning in AI has been based on three major assumptions about the world:

Simple: the world is simple enough that the planner can have complete knowledge of everything in it;

Static: nothing changes in the world except through the planner's own actions;

Certain: all actions have entirely predictable, deterministic outcomes.

Under these assumptions, it is possible to perform *classical planning*: construct a sequence of actions that is guaranteed to achieve the goal, given the initial conditions. Executing a plan constructed in this way is a very simple matter: everything has been fully predicted, so there is nothing to do but execute the actions, which are guaranteed to succeed.

There are very few domains in which these assumptions hold true: mostly, the world is complex, dynamic, and uncertain. It is impossible to have complete knowledge of the world, there are other agents and exogenous events changing it, and actions may have uncertain results. This means that in most domains the classical planning paradigm is not applicable as it stands. It is simply impossible to construct a plan that is guaranteed to succeed and that specifies the complete details of every action that must be performed.

The Lotus Notes domain is one in which the classical planning assumptions do not hold. The principal way in which they fail is that the domain is complex; perfect knowledge of the domain would mean having complete information about all databases and all documents in them. In addition, there is some dynamism, as other users modify, add or delete databases and documents. In theory, the domain is certain, in that actions are deterministic, but in practice the complexity and dynamism lead to a perception of uncertainty.

There have been a number of approaches to the problem of constructing plans in real world domains, including contingency planning (see, for example, [1,2,3]) and decision theoretic planning (see, for example, [4,5]). Other work has addressed the problem of how to execute plans in these domains. The issues that arise during plan execution include: specifying details of actions based on the exact circumstances encountered, choosing alternative courses of action, handling unexpected plan failure, and recognising and taking advantage of opportunities [6,7,8].

```
(define-rap
  (index (open-database ?database))
  (succeed (open ?database))
  (method
    (context (in-workspace ?database))
    (task-net
      (t1 (really-open ?database))))
  (method
    (context (and (not (in-workspace ?database))
                  (know-location ?database)))
    (task-net
      (t1 (add-to-workspace ?database)
          (in-workspace ?database) for t2)
      (t2 (really-open ?database))))
  (method
    (context (not (know-location ?database)))
    (task-net
      (t1 (find-location ?database)
          (know-location ?database) for t2)
      (t2 (add-to-workspace ?database)
          (in-workspace ?database)  for t3)
      (t3 (really-open ?database))))
  )
```

Figure 1: Part of a RAP to open a database

Plan execution

An approach to plan execution that has proved productive is one that uses a hierarchy of sketchy plans, expanded on an as-needed basis. A sketchy plan is one that that provides an outline of the actions to be performed, without specifying all the details. Systems based on this notion include the RAPs system [6] and PARETO[2] [7], both of which use RAPs (Reactive Action Packages) to represent plans. A RAP consists of a number of steps, each of which expands into either another RAP or a primitive action. Execution consists of expanding steps; the exact form the expansion takes depends on the circumstances at the time of expansion. If a step expands into a primitive action, that action is performed. This execution model allows details to remain unspecified until they are required in order to decide exactly what to do. The instantiations of the sketchy plans depend on the situations actually encountered during plan execution, thus reducing the need for unreliable predictions.

Figure 1 shows a simple RAP that might be used in our

[2] Planning and Acting in Realistic Environments by Thinking about Opportunities. The economist, sociologist and philosopher Vilfredo Pareto (1848–1923) is best known for the notion of *Pareto optimality* and for the *Pareto distribution*, neither of which is used in thePARETO system.

example domain. Each RAP has an `index`, which is used to retrieve its definition from the RAP library. An index may include variables, denoted by a beginning question mark. The `succeed` clause specifies the conditions under which the RAP may be considered to have succeeded. This is necessary because simply performing a sequence of actions does not guarantee success in a dynamic environment. The RAP shown in Figure 1 has three `methods`, or different ways in which the task can be performed. The `context` clause of each method specifies the circumstances in which it is appropriate. The `context` is checked against PARETO's current world knowledge, which has been gained through observation and feedback from actions that have been performed. Each `method` contains a `task-net`, which specifies the actions to be performed and the dependencies between them. The full syntax of RAPs and the details of the execution algorithm are given in [9].

Opportunities

An important aspect of plan execution in complex domains is the handling of opportunities. If it is impossible to foresee everything that might happen, which it is, you have to be able to handle the unexpected. Unforeseen circumstances might have no effect on your goals, but they might affect them either favourably, in which case there is an opportunity, or adversely, in which case there is a threat.

Opportunity recognition is complex in two ways. First, there is an enormous number of elements in every situation that no agent can possibly predict. None of these elements can be ruled out *a priori* as never being relevant to any goal. Suppose, for example, that you have a goal to open a can of paint. The objects in your garage include your car, the shelves on the wall, engine oil, *etc.* Few of these are relevant to your current goal, but they may be relevant to other goals at other times. Second, there are many subgoals involved in achieving a goal. For instance, to pry the lid off the paint can you must find something that is the right size and shape to act as a lever, a strong rigid surface on which you can rest the can at a convenient height, and so on.

The analysis of each situation element of a complex environment in the light of each of the agent's many goals would involve huge numbers of subgoals and situation elements and would preclude a timely response to unforeseen situations. Moreover, such an analysis would require the determination of each goal's subgoals, thus demanding the existence of a plan to achieve that goal. However, the recognition of an opportunity may trigger a radical change of plan or the construction of a plan for a hitherto unplanned-for goal. PARETO can recognise and take advantage of opportunities in these circumstances. It uses a filtering mechanism that indicates those situations in which there are likely to be opportunities and that will therefore repay further analysis. For a detailed description of this mechanism, see [10,7].

PARETO's opportunity recognition mechanism relies on *reference features* for its effectiveness. Reference features represent the functional tendencies of elements in PARETO's world: for example, an object with the reference feature *sharp* tends to cut soft objects, scratch harder ones, burst membranes and sever taut strings. The term *sharp* labels a collection of related effects. Knowledge about causal effects is thus represented in PARETO by attaching reference features to objects in the world and to its goals. PARETO uses this knowledge in its heuristic filters, without ever having to reason explicitly about the associated causal effects.

The concept *sharp* is useful just because many commonly arising human goals involve structural integrity. If, however, we lived in a world in which structural integrity was unimportant, we might well not even have such a concept, let alone find it useful. The reference features that an agent finds useful depend on the tasks that it habitually performs. Agents performing different tasks in the same world may attach completely different reference features to objects in their environments. Properties that are significant to one agent may be completely meaningless to another.

Plan execution in knowledge management

PARETO was originally developed in a simulated domain: it managed a robot delivery truck in a world developed using the TRUCKWORLD simulator [9,11,12]. Firby has used a later version of the RAPs system from which PARETO was developed to manage a mobile robot [13]. PARETO itself has been applied to two software domains: environmental planning [14] and UNIX [15]. It is not especially difficult to write RAPs for new domains; in fact, they have provided a surprisingly natural framework within which to analyse tasks. It is likely that the RAPs framework would be equally suitable for a knowledge management domain.

Advantages of a tool based on plan execution

The obvious question to ask at this stage of the discussion is what the point of applying a plan execution system to a knowledge management task would be. How would the user gain? Why bother? There are several ways in which a tool based on plan execution could make knowledge management tasks easier to perform. Before discussing these advantages, however, we must consider the form that such a tool would take. For reasons discussed in the following section, ir is likely that it would not autonomously perform knowledge management tasks but would instead be a mixed-initiative tool: one that interacts with the user as they jointly work towards the user's goals. Ideally, the level of autonomy in the tool would be adjustable to suit the user's wishes.

The tool would provide a structure within which to perform the task. Depending on the task in question, this structure might provide an element of quality control in the task; ensuring that certain necessary stages are passed

through, for example, or that items on a check-list are all covered. Like many knowledge-based systems, the tool would help novice users by supporting them as they perform the task. It could offer explanations of the decisions taken in terms of the RAP-based task analysis. By adapting itself to the user's preferences and level of expertise, it could gradually reduce the level of support offered as a novice user became more competent.

In addition, the use of a tool based on a system such as PARETO would enable the user to perform the task at a more abstract level; the tool would essentially form an interface between the user and the other systems being used, such as Lotus Notes, allowing the user to ignore the nitty gritty details of exactly how the Lotus Notes environment works or the precise commands needed in order to open a given database. It could in fact provide a transparent interface to several different proprietary systems, thus reducing or even eliminating the user's learning curve. It would do this by providing the user with executable commands at a much higher level than "add this icon to my workspace". Commands such as "open this database" would be treated by the tool as goals to be accomplished through the use of sketchy plans. The sketchy plans would then be expanded into the required low-level commands.

Overall, the tool's main contribution would be to reduce the effort required on the part of the user in performing the routine parts of the task at hand, allowing them to concentrate on the more interesting parts. In particular, as will be discussed in the next section, it would allow the user to concentrate on the portions of the task that require judgement and domain expertise.

Problems with plan execution systems

The previous section described some of the advantages that would accrue through the use of a tool based on a plan execution system. However, in order to capitalise on these advantages the system must actually be built. There are two main tasks in building a knowledge-based system; choosing the architecture and representing the knowledge. We have seen how an architecture based on the hierarchical expansion of sketchy plans could be used; we must now consider the knowledge representation issues.

There are two types of knowledge that would have to be represented: knowledge about the knowledge management task, which I shall term *task knowledge*, and knowledge about the domain in which that task is being performed, which I shall term *domain knowledge*.

Task knowledge is represented in the system in the raps that are used to specify how tasks are to be performed. This knowledge does not depend on the domain in which the task is being performed; the methods used to find a document in Lotus Notes, for example, do not depend on the subject matter of the document. Rather, they depend on the way in which Lotus Notes works.

Representing domain knowledge, on the other hand, presents much more of a problem. The successful performance of a knowledge management task such as using Lotus Notes relies heavily on a great deal of information about the contents of the documents, possible synonyms, relationships between concepts, the fact that document titles may be meaningful, *etc.* In order to build a system that can assist in this task, we need to be able to represent at least some of this knowledge. Of course, in attempting to build a system to assist in the task rather than one to perform the task autonomously we are not cutting ourselves off from the enormous amount of knowledge that the user can bring to bear on the problem; but in order for the system to be of any use it must perform at least some of the task itself.

There are two principal problems posed by the representation of domain knowledge. First, what domain or domains are we talking about? A useful tool should be able to operate in many different knowledge domains. For example, a tool that can help track down documents in Lotus Notes would not be much use if it could only track down documents on a single topic, or even if there were a set of predefined topics. The type of tool under consideration would be useful precisely because it would obviate the need for the comprehensive indexing of documents in advance. Users would not be limited to queries that had been anticipated by the authors of documents; the system would help find documents in response to queries about topics that may not even have existed when the documents were created. The difficulty here, then, is the *scope* of the domain knowledge that would be required.

The second problem with domain knowledge concerns *how* it should be represented. As usual, the criteria used to judge the representation method should take account of both how the knowledge is used by the system and how it is acquired when the system is built. Ideally, the representation should facilitate the efficient operation of the system and should be easy for the knowledge engineers building the system to use. In PARETO, the domain knowledge is used by the system to guide task execution by choosing which sketchy plans to expand and how to expand them.

Figure 2 shows part of a RAP (the control information is omitted) to find a document that is relevant to some topic specified by the user. The domain knowledge is represented by such predicates as `relevant-category` and `relevant-name`. Predicates such as these, used in RAP definitions in such places as the `success` and `context` clauses (and elsewhere), are used to guide task execution according to PARETO's beliefs about the world. To use them, PARETO must be able to judge whether they apply in the situation in which it finds itself. In other words, it must be able to compare the actual state of the world, which it discovers through interacting with it through information-gathering and other actions, with functional requirements imposed by the goals it is pursuing. The problem arises because of the necessity of translating between the two very different natural characterisations that arise.

```
(define-rap
  (index (find-relevant-document ?topic
                                 => ?document))
  (succeed (and (open ?database)
                (contains ?database ?document)
                (relevant ?topic ?document)))
...
  (method
    (context (and
               (open ?database)
               (contains ?database ?document)
               (relevant-name ?topic ?document)))
    (task-net
      (t1 (check-contents ?topic ?document))))
  (method
    (context
      (and (open ?database)
           (relevant-category ?topic ?category
                              ?database)
           (contains ?database ?document)
           (category ?category ?document)))
    (task-net
       (t1 (check-contents ?topic ?document))))
  )
```

Figure 2: Part of a document-findingRAP

The first characterisation arises from the requirement to guide task execution through consideration of the state of the environment. PARETO must characterise the world in terms of its perceptual features — the features that it can observe directly.[3] For example, it might observe the presence or absence of particular words or phrases in a document, or of particular views in a database.

The second characterisation arises through the requirement to guide task execution in order to achieve certain goals, such as the discovery of a document that is relevant to a particular topic. To meet this requirement, PARETO must characterise the world in terms of its functional features — features that affect goal achievement.

The trouble is that these two characterisations are not usually the same. Translating between the two is a non-trivial task that may require an arbitrary amount of reasoning using an arbitrary amount of domain knowledge. If a tool based on a system such as pareto is to be effective, it must run efficiently, which means that the reasoning must be limited. Obviously, as machines get faster the absolute level of reasoning that is possible without adversely affecting performance increases, but the problem remains. Moreover, it is exacerbated by the problem of domain scope referred to above and the problem of knowledge engineering. Is it possible to decide in advance what knowledge should be represented in order to judge relevance to arbitrary future queries?

An area of future work that may point towards a solution to this problem is the continued investigation of the *reference features* that are used in PARETO's opportunity recognition mechanism. Reference features are intended to provide a bridge between perceptual and functional representations (see [9,16] for a discussion of this aspect of them), and it may be possible to use them more generally than in the recognition of opportunities. Another approach might be to use some form of reinforcement learning to allow the system to learn to associate functional characteristics with perceptual features through user feedback.

Although knowledge representation issues are the most important problems that arise in the use of plan execution systems for knowledge management tasks, they are not the only ones. For instance, a potential advantage of the use of such tools would be transparency: the user could remain ignorant of the details of the particular proprietary system being used to store the knowledge being managed. However, this might be difficult to achieve in practice because of the fairly high level constraints placed on task performance by the architecture of the underlying system. For example, Lotus Notes has a very different information structure to the world wide web; although both have hyperlinks, embedded graphics and so on, Lotus Notes imposes a much more rigid framework.

Conclusion

In this paper I have argued that it will prove productive to think of some aspects of knowledge management in terms of plan execution. Moreover, the hierarchical expansion of sketchy plans, as used in the RAPs system and its offshoot PARETO, provides a natural framework in which to represent knowledge management tasks.

The big challenge is the representation of the world knowledge. The challenge can be partially overcome by abandoning the idea of an autonomous system in favour of a mixed-initiative system, *i.e.*, one that works interactively with the user to accomplish its tasks. However, in order for such a system to be useful it must not leave everything up to the user; in particular, it must itself have some world knowledge that allows it to judge potential relevance.

References

1 Peot, Mark A and Smith, David E. 1992. Conditional Nonlinear Planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 189–197. College Park, Maryland.

2 Goldman, Robert P and Boddy, Mark S. 1994. Conditional Linear Planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 80–85. Chicago, IL.

3 Pryor, Louise and Collins, Gregg. 1996. *Planning for contingencies: A decision-based approach.* Journal of Artificial Intelligence Research 4:287–339.

4 Haddawy, Peter and Suwandi, Meliani. 1994. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of the Second International Conference on Artificial Planning Systems*, 266–271. Chicago, IL.

[3] Obviously the term "perceptual" is used somewhat loosely here.

5 Kushmerick, Nicholas, Hanks, Steve and Weld, Daniel. 1995. *An Algorithm for Probabilistic Planning*. Artificial Intelligence 76:239–286.
6 Firby, R James. 1989. *Adaptive execution in complex dynamic worlds*. Technical Report YALEU/CSD/RR 672, Department of Computer Science, Yale.
7 Pryor, Louise. 1996. Opportunity recognition in complex environments. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1147–1152. Portland, OR.
8 Firby, R. James. 1996. Modularity Issues in Reactive Planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, 78–85. Edinburgh.
9 Pryor, Louise. 1994. *Opportunities and Planning in an Unpredictable World*. Technical Report 53, Institute for the Learning Sciences, Northwestern University.
10 Pryor, Louise and Collins, Gregg. 1994. A unifying framework for planning and execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 329–334. Chicago, IL.
11 Firby, R James and Hanks, Steve. 1987. *The simulator manual*. Technical Report YALEU/CSD/RR 563, Department of Computer Science, Yale University.
12 Hanks, Steve, Pollack, Martha E and Cohen, Paul R. 1993. *Benchmarks, testbeds, controlled experimentation, and the design of agent architectures*. AI Magazine 14(4):17–42.
13 Firby, R James. 1994. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 49–54. Chicago, IL.
14 Cheng, Boon Fooi. 1995. *Plan Execution in an Environmental Domain*. MSc thesis, Department of Artificial Intelligence, University of Edinburgh.
15 Green, Shaw. 1996. Planning Actions in Dynamic Environments: A UNIX Assistant. MSc thesis, Department of Artificial Intelligence, University of Edinburgh.
16 Pryor, Louise. 1994. Perceptual and functional representations: Bridging the gap. In *Notes of the AISB Workshop on Computational Models of Cognition and Cognitive Functions*, Leeds, AISB.