Opportunism without Deliberation?

Louise Pryor
Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland
louisep@dai.ed.ac.uk

Introduction

Recently, increasing attention has been given to the problem of how to produce goal-directed behaviour in the real world. The approaches taken vary along a spectrum from trying to extend the scope of classical planning techniques in systems such as Cassandra (Pryor & Collins 1996) and BURIDAN (Kushmerick, Hanks, & Weld 1995); via including execution in plan construction systems such as Sage (Knoblock 1995); through developing systems such as RAPS that execute ready-made plans (Firby 1989; 1996); to the design of systems that compile plans into sets of reaction rules (Schoppers 1987; Schoppers & Shu 1996). It seems reasonable to suppose that eventually much of this work will converge somewhere near the centre and will result in systems that seamlessly integrate plan construction with plan execution, thus allowing planning, re-planning and acting to be interleaved. Indeed, this is the theme of much recent research as evidenced by many of the papers included in these Working Notes.

There is, of course, disagreement about whether this eventual aim is best approached by starting from the plan construction end and adding in more flexibility, or by starting from the plan execution end and adding in more deliberation. In this brief paper I describe an example of the latter approach and consider how much flexible goal-directed behaviour can be achieved with what appear to be minimal deliberative capabilities.

The next section describes a plan execution system, PARETO,[1] that can recognise and take advantage of a large class of opportunities (Pryor 1996b). The following section discusses the forms of deliberation present in PARETO. The final section considers the achievements and shortcomings of the system.

[1]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 this work. PARETO is described in detail in (Pryor 1994).

How PARETO works

PARETO is based on the RAPS plan execution system (Firby 1987; 1989). When PARETO acquires a new goal, it looks in its library of RAPs (sketchy plans) for one that will achieve the goal. A RAP (Reactive Action Package) specifies all the different *methods* that might be used to achieve a goal. Each method consists of subgoals that must be achieved to execute the plan.

Having chosen a RAP for the goal, PARETO adds a new task to its *task agenda*. A task consists of a goal and a RAP that will achieve it. PARETO's execution cycle consists of choosing a task and processing it:

Either If it has succeeded remove it from the agenda;

Or If it is described by a RAP choose the appropriate method, based on the state of the world at the time that the processing takes place, and add a new task to the task agenda for each new goal;

Or Perform the primitive action specified by the task.

The original task is reprocessed after all its subtasks have succeeded. Their success does not guarantee the success of the original task as some time may pass between their execution and its repeat processing.

PARETO's Opportunism

PARETO's world was built using the TRUCKWORLD simulator (Firby & Hanks 1987; Hanks, Pollack, & Cohen 1993). A delivery truck travels between locations, encountering and manipulating objects as it goes. There are three building sites whose workers use the truck to run delivery errands such as "fetch something to carry my tools in." There are over 30 types of object in PARETO's world, of which 20 are used for deliveries. At any moment, there are typically well over 100 different objects at the various locations. PARETO receives delivery orders at random intervals, with a typical run involving between seven and twelve separate deliveries. The world is unpredictable: the truck can sense only those objects at its current location; objects may spontaneously change location; and actions may fail to have the desired results.

PARETO operates opportunistically in this environment. It can recognise opportunities that involve changing the plan used to achieve a goal, and opportunities for goals that have not yet been planned for (Pryor 1996b). For example, suppose the truck is on its way to a specific location where it believes there to be a bag, which it intends to use to fulfill a goal to find something that can be used to carry tools, and that it also has another goal to find something with which to cut twine. As it passes through another location it spots a box and a pair of scissors. PARETO recognises the presence of these objects as opportunities for its two goals, even though picking up the box will involve abandoning its intention to fetch the bag. It recognises the opportunity presented by the scissors even if it has not yet chosen any plan at all for the cut-twine goal.

In taking advantage of these opportunities PARETO switches its focus of attention between its goals in such a way that it never loses track of its principal objectives (Pryor 1996a). It picks up the scissors, then the bag, and then goes off to deliver the scissors. It does not get side tracked by repeatedly abandoning its current task in order to pursue an opportunity, from which it is then distracted by a further opportunity, and so on.

PARETO's Architecture

PARETO's architecture is shown in Figure 1. There are three levels in the architecture envisaged by Firby:

Reasoner Performs plan construction and adaptation, and any other deliberation that might be required;

Execution system Fills in the details of the sketchy plans based on the situations that are encountered;

Hardware control system Translates the actions in the sketchy plans into control processes that can be executed by the agent.

The *hardware interface* and *information sources* provide the means of communication between these layers.

The current implementation of PARETO uses a simulated hardware controller in the form of a simulated robot delivery truck. There is, however, no integrated reasoner in the system. In Firby's vision of this architecture, the reasoner would construct new plans (all PARETO's RAPs have been hand coded) and make strategic decision about how to combine plans for the various goals, what methods should be used to accomplish goals, and which goals should be pursued. The reasoner would do this through deliberation, using information from the RAP library, task agenda, and memory. It would communicate with the execution system via the same modules.

Despite the lack of a full reasoner, PARETO can perform some reasoning tasks. Its plans can specify acts of inference that should be performed, just as they can

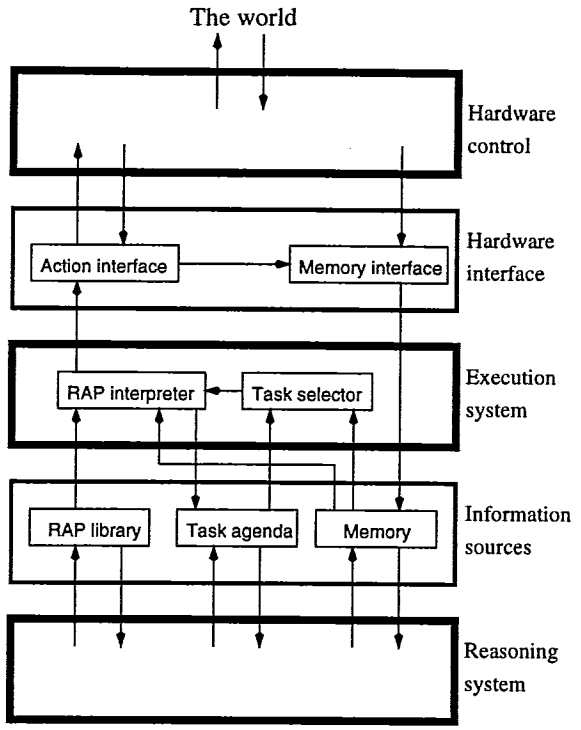

Figure 1: PARETO's architecture

specify physical actions. It might also be argued that PARETO performs reasoning in order to recognise opportunities. However, the opportunity recognition mechanism involves minimal reasoning and is performed almost entirely within the RAP interpreter and the task selector, as described in the following section.

Deliberation in PARETO

Although PARETO's architecture is envisaged as being composed of the three distinct layers shown in Figure 1, it is not nearly so clear-cut in practice. There are at least four types of activity performed by PARETO that could be said to be deliberative, only one of which — explicit inference — takes place in the reasoning layer. The others — implicit inference, opportunity recognition and opportunity pursuit — are governed by the execution layer.

Opportunity recognition

PARETO recognises and pursues opportunities through a process involving alternate filtering and analysis stages (Figure 2). All filtering and most analysis takes place in the task selector module of the execution layer.

The idea behind this process is that there is an opportunity for a goal if the goal is easy to achieve and if pursuing it will not interact adversely with any other goals (Pryor & Collins 1994). Because of the complexity of both the agent's environment and its goals, it is

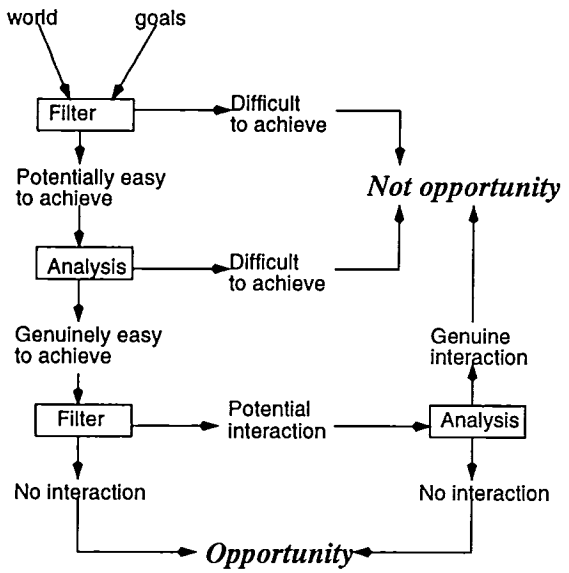

Figure 2: Recognising opportunities

impossible to perform a detailed analysis of all possible ways of achieving every goal (Pryor 1996b). Instead, PARETO uses a heuristic filter to indicate those goals that are potentially easy to achieve. The goals that have been indicated are then analysed in more detail to see whether they are genuinely easy to achieve. Finding problematic interactions is also extremely complex, and a similar heuristic filter is used to indicate those that should be analysed in more detail.

Neither filter involves any reasoning. They both involve comparing two sets of *reference features*[2] and looking for matches. The analysis for easy achievement is also simple to perform. A task on the task agenda is considered to be easy to achieve either if it has already succeeded or if it is not waiting for any other tasks to complete. Both these determinations are performed routinely by the task selector in other contexts: whenever a task is selected for processing a check is made to see whether it has already succeeded, and except for unexpected opportunities only those tasks not waiting for any others are considered for expansion.

The only stage in the opportunity recognition process that involves true deliberation is the analysis of potentially problematic interactions, and this is performed by the reasoning layer. When a potential opportunity is indicated as having a potentially problematic interaction with another task, a new task is created to reason about the interaction. This new task is placed on the task agenda, and constraints added to ensure that it is expanded before the tasks whose interaction is to be analysed. The new task uses the explicit inference mechanism discussed below. The reason for adding a new task is that detailed analysis of interactions may be arbitrarily complex: it is a significant activity that should be treated on a par with other planned activities (Pryor & Collins 1991).

[2]Simple functionally relevant characterisations such as *sharp* (Pryor 1996b).

Opportunity pursuit

The pursuit of opportunities is governed by the task selector, whose job is to choose a task for expansion from the task agenda. At any time during execution PARETO's task agenda may hold tasks at varying levels of abstraction, ranging from tasks that have not been expanded at all to those consisting of single primitive actions. Some of these tasks may be waiting for other tasks to complete, or for specific situations to obtain. However, there are usually many tasks that are eligible for processing, and PARETO must choose one of them. The mechanism that it uses to make this choice is based on the notion of *opportunities* (Pryor & Collins 1994); whenever PARETO must choose a task to process, it chooses from among those tasks for which there are good opportunities.

The way in which PARETO decides what to do next takes four important factors into account. These factors can be expressed as desiderata for reasonable behaviour:

- Concentrate on important goals;
- Seize opportunities;
- Avoid unnecessary interruptions;
- Don't get side tracked.

In general, no single factor can be taken as being more crucial than the others; the balance between them varies according to the circumstances. None of them is desirable in all circumstances. PARETO uses simple heuristics, based partly on annotations attached to its goals, to balance the conflicting factors (Pryor 1996a). It does not analyse the likely effects of any of the possible courses of action.

Implicit inference

PARETO's inference consists entirely of various types of plausible (*i.e.*, non-deductive) reasoning, much of it implicit. The principle type of implicit inference that is performed is through property inheritance. If an object is known to be of a particular class, such as a `safe`, it can be assumed to have all the properties that `safe` usually have. For example, PARETO assumes that examining the `front` of the `safe` will inform it whether the `safe` is `open` or `locked`; that the `safe` will have a `combination`; that it can be opened using the `combination`; that it can contain other objects; and so on. These implicit inferences about `safe`s and other objects are built into the RAPs that it uses to achieve

```
(define-rap
    (index (find-safe-combination ?safe => ?comb))
    (succeed (and (combination ?safe ?comb)
                  (not (= ?comb unknown))))
  (method
   (task-net
    (t1 (discover-owner ?safe => ?owner)
        (for t2))
    (t2 (discover-birthday ?owner => ?date)
        (for t3))
    (t3 (mem-assert-p (combination ?safe ?date))))
  ))
```

Figure 3: Finding the combination of a safe

its goals. For example, consider the RAP for finding out a safe's combination shown in figure 3. It uses the (admittedly unusual) property of all safes in PARETO's world: the combination is always the of the owner. This RAP also demonstrates PARETO's ability to make deliberate assertions to its memory (assertions are usually made as a result of the receipt of sensory data).

RAPs may also sanction other inferences. One subtask of finding an object is checking that it meets the criteria for the delivery goal: part of the RAP that does this is shown in Figure 4. There is no explicit inference step. Although this is definitely deliberation that PARETO is performing, it is not taking place in the reasoning layer. Instead, this deliberation is planned in advance and is performed implicitly as the plan is executed.

Explicit inference

As well as the forms of implicit inference described in the previous sections, PARETO can perform explicit inference. A RAP method can consist of a reasoning request, signifying that a request should be sent to the reasoner for the appropriate reasoning to be performed. This is analogous to the way in which a RAP method consisting of a primitive action signifies that an instruction should be sent to the hardware controller. PARETO can currently handle four types of reasoning request, using special-purpose Lisp code for each.

The `infer-default` reasoning request is used to infer a default value for a property of an object. It is used if it is for any reason impossible to observe the property in question and an assumption must be made about the value. The way in which this reasoning request is implemented allows fully general reasoning to be used: for example, default values may depend on a combination of other properties of the object, or on its current location.

The `infer-examine-for-match` reasoning request is used to determine how to examine an object when trying to determine whether it can be identified with one that has been observed previously. PARETO, like the RAPS system on which it is based, cannot automatic-

```
(define-rap
    (index (check-object-ok ?object ?type
                            ?color ?size))
 ...
 (method
  (context (and (or (= ?type dont-care)
                    (type ?object ?type true))
                (or (= ?color dont-care)
                    (color ?object ?color))
                (or (= ?size dont-care)
                    (size ?object ?size))))
  (task-net
   (t1 (mem-assert-p (object-ok ?object ?type
                                ?color ?size
                                true))))
 ) ... )
```

Figure 4: Checking an object against goal criteria

ally distinguish one object from another of the same type (Firby 1987). For example, it may see a box at the location base, having previously put a hammer into one at the same place. If it is to recover the hammer, it must identify the box as being the same as the one it previously encountered. To perform such identifications, PARETO tries to match all the previously observed properties of the object in question, such as its size and colour. PARETO's powers of observation are limited, and the properties it observes depend on the way in which it examines an object. For example, whether a safe is open or locked can be observed by examining its front, but its top must be examined in order to find out who its owner is.

The `infer-examine-for-match` reasoning request uses information about the object to be identified and the properties of any other objects that are currently believed to be at the same location to decide what observation should be made. There are two cases: either the object that PARETO is trying to identify has a property that hasn't been checked for all the candidate objects, or there is an object of the right type with a property value other than that required, and that property has not been checked for all candidate objects.

The `analyze-potential-interaction` reasoning request is used to determine whether there is a genuinely problematic interaction between two tasks. The filtering stage mentioned above indicates the type of interaction whose possibility should be analysed. For example, if PARETO is considering picking up a roll of wallpaper when it is already carrying a knife there may be a cut interaction. The analysis of potential interactions may require further information about the current circumstances: *e.g.*, a cut interaction will only take place if the knife's blade is extended. If the blade is retracted, the wallpaper is safe. A reasoning request may therefore add further tasks to the agenda, such as one to examine the blade of the knife.

An `analyze-potential-interaction` reasoning request not only determines whether a potential interaction is genuinely problematic, it also finds and initiates a method of involving those interactions that are problematic. For example, if PARETO determines that the potential `cut` interaction is likely to take place, it will add a task to retract the blade of the `knife` to the task agenda, together with the appropriate constraints to ensure that the new task is expanded before the potentially interacting tasks. There are four simple avoidance strategies used by PARETO:

- Postpone one of the task involved in the interaction until the other has finished;
- Delay one of the tasks for a specific period of time;
- Perform actions to prevent the occurrence of the interaction;
- Note that the interaction is unavoidable.

PARETO uses the same avoidance strategy every time in encounters a particular type of interaction. In general, strategies to handle interactions could be highly contingent and involve arbitrarily complex changes to the tasks on the task agenda. McDermott 1992 describes some of the transformation that are possible. Perhaps the most obvious omission in PARETO is that it is unable to rule out the use of a particular object for the furtherance of a given goal.

The `solve-problem` reasoning request is used to find a RAP for a particular goal. There is currently just one problem-solver in PARETO, which finds a route from one location to another using simple breadth-first search. As well as being called explicitly via reasoning requests problem solvers are called automatically whenever PARETO fails to find a RAP in its library.[3]

Conclusion

PARETO is capable of surprisingly flexible behaviour considering the fact that it uses very little deliberation. Much of this flexibility is produced by a combination of Firby's plan execution algorithm and a significant body of knowledge that is in effect compiled in to PARETO's domain and plan representations. The little deliberation that is performed is not, on the whole, directed towards the analysis of cause and effect or towards projecting the course of future events.

Compiled knowledge

PARETO's ability to operate robustly in a dynamic world is founded largely on its plan representation language and plan execution algorithm, which both come from Firby's RAPS system. The plans themselves embody a large amount of object-level domain knowledge, enabling many implicit inferences along the lines of those described above. The hierarchical structure of the plans embodies meta-knowledge about how tasks should be structured in the domain, and the plan execution algorithm embodies domain-independent meta-knowledge about how to cope with plan failure (Firby 1989). Investigations which have been made into the possibility of adapting PARETO for use in domains other than the one for which it was originally designed have supported the idea that PARETO's architecture and plan representation language are essentially domain-independent (Cheng 1995; Green 1996). In other words, the RAP language is sufficiently expressive to represent knowledge about a range of different domains.

The second type of knowledge on which PARETO relies is to be found in its reference features, which are used extensively in the opportunity recognition mechanism. Reference features represent the functional tendencies of elements in PARETO's world (Pryor 1996b): 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 analysis of potential easy achievement also uses compiled knowledge: this time, knowledge that is represented in the RAPs about the circumstances under which tasks have been achieved and about the interdependencies between tasks.

Finally, PARETO's mechanism for choosing between tasks relies partly on annotations attached to its goals about their priority (Pryor 1996a). For example, some goals are labelled as being *urgent*, others as being *preservation* goals, and so on. These annotations are a type of reference feature and are heuristic in operation.

Focussed deliberation

The little explicit deliberation that PARETO does perform is very focussed. For example, when PARETO analyses a potentially problematic interaction between tasks it already knows the form the interactions will take and the elements involved. It does not examine any other possibilities. The other types of explicit deliberation are equally specific. Deliberation in PARETO is therefore implemented through special-purpose Lisp functions: there is no general-purpose reasoning facility. Because reasoning is treated like any other planned activity it is fairly simple to add in new reasoning cap-

[3]This functionality was provided in Firby's RAPS system, but was undocumented. The mechanism in PARETO is a slightly altered version of that used by Firby.

abilities: all that is needed is a new type of primitive reasoning action, which can then be used in writing new RAPs. It was mentioned above that PARETO cannot find an existing RAP for a goal it can use a problem solver to construct a new one. Currently, only one type of problem solver exists, but it would certainly be possible to write others. Again, the reasoning performed would be specific to the type of goal.

We can see, then, that there is plenty of scope for expansion in PARETO's reasoning layer through the addition of further *ad hoc* modules. It would also be possible to add more general-purpose capabilities that would use the same interface to the rest of the system. More interesting is the notion of extending the reasoning layer to perform a more central role in the execution process, for example by being involved in the process of selecting tasks to be expanded or choosing a method for a particular task. Extending PARETO in this way would move it further towards the deliberation end of the spectrum discussed in the Introduction.

Limitations

The implementation of PARETO in a simulated domain has demonstrated that the opportunity recognition filter based on reference features is feasible, but not that it will be effective in a real-world domain. The potential problems of scaling-up have not been addressed.

Moreover, the range of opportunities that PARETO can recognise is limited. It can only recognise opportunities whose pursuit does not involve substantial changes to its plans for other goals. This is because it cannot combine its plans in advance: it cannot, for example, decide to use a less efficient route to get somewhere because doing so would enable it to further another goal in some way. In this limitation to *immediate* opportunities, PARETO is very much staying within the original RAPS philosophy of being guided by the actual circumstances that it encounters.

Another limitation of the limited deliberation in PARETO is the lack of flexibility in threat avoidance. PARETO has a small number of crude strategies for avoiding problematic interactions between tasks, but has no general threat detection or avoidance mechanisms. Again, this is consistent with being guided by the circumstances that are encountered. The addition of more deliberative power to PARETO would certainly increase its range of behaviour. One reason for eschewing an over-reliance on deliberation is that deliberation may be computationally costly, and may reduce real-time responsiveness in dynamic environments. To date, PARETO has not been implemented in such environments so this has not been an issue.

References

Cheng, B. F. 1995. Plan execution in an environmental domain. MSc thesis, Dept. Artificial Intelligence, University of Edinburgh.

Firby, R. J., and Hanks, S. 1987. The simulator manual. TR YALEU/CSD/RR 563, Dept. Computer Science, Yale University.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *Proc. 6th Nat. Conf. on AI*, 202–206. AAAI Press.

Firby, R. J. 1989. Adaptive execution in complex dynamic worlds. TR YALEU/CSD/RR 672, Dept. Computer Science, Yale University.

Firby, R. J. 1996. Modularity issues in reactive planning. In *Proc. 3rd Int. Conf. on AI Planning Systems*, 78–85. AAAI Press.

Green, S. 1996. Planning actions in dynamic environments: A UNIX assistant. MSc thesis, Dept. Artificial Intelligence, University of Edinburgh.

Hanks, S.; Pollack, M. E.; and Cohen, P. R. 1993. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine* 14(4):17–42.

Knoblock, C. 1995. Planning, executing, sensing, and replanning for information gathering. In *Proc. 14th Int. Joint Conf. on AI*, 1686–1693. IJCAI.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

McDermott, D. 1992. Transformational planning of reactive behavior. TR YALEU/CSD/RR 941, Yale University, Dept. Computer Science.

Pryor, L., and Collins, G. 1991. Information gathering as a planning task. In *Proc. 13th Ann. Conf. Cognitive Science Soc.*, 862–866. Lawrence Erlbaum Associates.

Pryor, L., and Collins, G. 1994. Opportunities: A unifying framework for planning and execution. In *Proc. 2nd Int. Conf. on AI Planning Systems*, 329–334. AAAI Press.

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

Pryor, L. 1994. Opportunities and planning in an unpredictable world. Technical Report 53, Institute for the Learning Sciences, Northwestern University.

Pryor, L. 1996a. A heuristic approach to deciding what to do next. In *Notes 15th Workshop of the UK Planning and Scheduling Special Interest Group.*

Pryor, L. 1996b. Opportunity recognition in complex environments. In *Proc. 14th Nat. Conf. on AI*, 1147–1152. AAAI Press.

Schoppers, M., and Shu, R. 1996. Generalizing indexical-functional reference. In *Proc. 13th Nat. Conf. on AI*, 1153–1159. AAAI Press.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proc. 10th Int. Joint Conf. on AI*, 1039–1046. IJCAI.