Planning in an Open World: A Pluralistic Approach.*

Mitchell Marks, Kristian Hammond and Tim Converse
Department of Computer Science
University of Chicago
Chicago, IL 60637

Abstract

Recent work in planning has rejected the assumption of a closed, stable world, and the associated paradigm of exhaustive preplanning, which encounters serious problems trying to plan in a world where that assumption does not hold. Several alternative strategies have been proposed, responding to these new problems in a variety of ways. We review this spectrum, finding the various approaches in part incompatible but not bereft of some common themes and complementary strengths. We suggest factors in the application domain which should influence the appropriate mix, and describe the **TRUCKER** project to illustrate some of the problems and benefits in implementing such a mix.

Problems with Traditional Planning

The classical development of the theory of planning and problem-solving emphasized exhaustive preplanning, with the goal of being able to guarantee that an optimal or near-optimal plan would be found if one existed. Planners in this paradigm required certain assumptions to hold:

- The world will be stable; it will behave as projected.
- Time consumed in planning is independent of the time that can be devoted to execution, so that the efficiency of the planner has no side-effects on the feasibility of the constructed plan.
- The information available to the planner is complete, and execution will be flawless.
- Any initially correct plan will remain correct and can in fact be carried out.

In the real world, however, these assumptions simply do not hold. The world is not stable; agents must trade off planning time against execution time; and planners generally have to function under conditions of spotty rather than complete information. The simplifying assumptions were made in order to initiate progress in the serious investigation of planning. The harsh realities were consciously abstracted out of the theories initially, not merely overlooked. But with advancing theory, researchers have recently begun finding it feasible to explore the problem of planning in more realistic situations where these assumptions do not hold.

New problems in planning

In this paper we will attempt to classify the different kinds of issues that any planner must confront as we relax these traditional assumptions. We will discuss the different theories of planning that have arisen in

*This paper was submitted for presentation at the *1989 Meeting of the Cognitive Science Society*. It was also presented as a paper at the 1988 DARPA Workshop on Case-based Reasoning. This work was supported in part by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058, and the Office of Naval Research under contract N0014-85-K-010.

response to these issues. And we will present **TRUCKER**, a planner that combines features of many of these theories in an attempt to deal with the complexities of combining planning and acting in one system.

We now give a rough classification of the major sorts of problems a traditional planning system can encounter when required to deal with a more realistic domain. These problems, in turn, provide a basis for understanding what motivates the departures taken by theorists in recent work.

The Immediate Complexity Problem. This concerns the computational effort required in constructing one plan. If the planner searches for a correct and safe plan by projecting forward the effects of early steps to compare with preconditions of later steps, goals, and preservation conditions, the computational complexity can rise to a high order.

The Asymptotic Complexity Problem. As a planner interacts with the world, it is confronted with a stream of goals, or sets of conjunctive goals, rather than independent problems. If these are all treated singly, independently, the total planning and execution effort would be at least the sum of the separate costs (possibly worse, due to unfavorable interactions). But if the planner can somehow convert the interdependence from a problem to an advantage, then the average cost per goal can be reduced in the long run, offsetting the Immediate Complexity Problem.

The Execution-time Failure Problem. A plan which looked correct when it was constructed may turn out to be incorrect during execution. This may be because conditions in the world have changed in the meantime, invalidating the preconditions of a plan step, or because the plan was incorrect in the first place, based on incorrect assumptions in the planner's limited world-knowledge. To cope with this, a planner must have some facility for replanning, recovery, and repair.

The Planning/Execution Crowding Problem. If planning and execution are to be carried out by the same agent, essentially without parallelism, then time consumed in planning can deplete the time available for execution. This can create a situation where some series of actions would be a correct response to the goals and the world state, and could be feasibly executed within the total time available, but become infeasible within the time left after planning.

The Costly Information Problem. The planner cannot count on having complete information about a domain, either its underlying causal "physics" or its current state. To minimize the effects of this information shortage (namely inefficient plan construction and inaccurate plans), the planner should have information-gathering as a background goal. But understanding the world correctly, and storing new information in a usable form, can be expensive operations.

The Missed Opportunities Problem. A corollary to the Execution–time Failure problem is the Missed Opportunities problem. Because the world does not necessarily match the planner's understanding of it, it is often the case that opportunities to satisfy goals are missed at planning time and must be noticed and exploited at execution time.

NEW APPROACHES TO PLANNING

Several new directions in planning have arisen in response to problems like those enumerated earlier. All of them address the Immediate Complexity problem in one way or another, but differ in the additional emphasis they give to the various other problems. The outlook we propose here is that in most realistic environments, all of the problems will have to be addressed; so an ideal planner would combine the strengths of these different approaches, marshalled against the respective problems they most directly ameliorate.

One such new approach has come to be known as *reactive planning* (Agre & Chapman 1987) or *situated activity* in the preferred terminology of Agre and Chapman, following Suchman (1986). The major assumption of the traditional paradigm challenged by these authors is that adequate planning time is always available; along with the Immediate Complexity problem they are most directly concerned with the Planning/Execution Crowding problem. To function in a fast-changing world, a planner may have to pay more attention to execution or interpretation of plans and less to construction of plans. In the purest realizations of this idea, the planner will end up working from reflex-like stored responses for each immediate situation

rather than true goal-directed plans. These are useful ideas, even when complexity or sensitivity of the problem domain dictates that they cannot be applied in pristine, radical form.

A key point from this line of thought is that planning and execution cannot be as neatly separated as traditionally supposed. This is not just because planning and execution share the same pool of available time (though that is an important aspect); rather, this represents a change in the very fundamentals of how to think and talk about planning. There isn't planning plus execution, there is one combined activity.

Various researchers have tried to address these same problems without completely abandoning the classical orientation that views activity as the execution of plans. *Reactive planning* systems (Firby 1987, Georgeff & Lansky 1987) are an attempt to combine plan construction and plan execution under a single control structure, in such a way that the systems will be robust to changes in the world while at the same time leaving room for genuine planning. The system described by Firby (1987) works with units of action called reactive action packages or **raps**. A **rap** "is essentially an autonomous process that pursues a planning goal until that goal has been achieved." Each **rap** has a number of methods that it knows about for achieving its goal, and will try methods until it can verify that one has succeeded. **raps** that are to be executed wait in a linear queue, and execution of a **rap** may involve a sequence of primitive actions or may in turn invoke other **raps**. In the latter case this provides some hierarchical structure to the plans specified by **raps**, without demanding elaboration to the level of primitive actions before execution can begin. The combination of multiple methods and verification of success results in a system that is very robust to execution failure.

The *procedural reasoning system* described by Georgeff and Lansky (1987) is an architecture for control of a mobile robot, in which the goals, beliefs, plans, and intentions of the robot are separately and explicitly represented. The current goals and beliefs about the state of the world determine which plans are chosen to be put on the execution stack. These plans hierarchically structure sub-plans, and may be addressed either to actions in the world or to manipulations of goals, beliefs and intentions themselves. The fact that new goals may be constructed in response to information gained during execution, and in turn push new plans onto the "intention" stack, makes possible a "shift of focus" in execution in response to new information. In particular, this permits the robot to interrupt the performance of a routine task in response to the detection of an emergency, and then resume the original task once the emergency has been dealt with.

An approach generally called case-based reasoning has become an emergent paradigm in several areas of AI, including planning, natural language understanding, diagnosis/repair systems, and problem-solving (Hammond 1989, Kolodner *et al.* 1985). As an approach to planning, *case-based planning* departs from traditional methods via an emphasis on the rôle of memory; more specifically, an episodic memory of past goals and the plans that succeeded or failed in satisfying them. This emphasis is directed at taming the Immediate Complexity problem and especially the Asymptotic Complexity problem.

In the purest form of case-based planning, new plans are *always* derived from plans in old cases, and are *never* computed purely from scratch (world knowledge and inference rules). A pristine case-based planning approach is best suited to a domain where execution failures or at least suboptimal execution can be tolerated. When a less fault-tolerant domain or task requires giving up this purity of approach, the correct response, we argue, is not to retreat to a full-scale projection of the plan's effects in the world. In the first place, that would mean giving up the efficiency advantages which form part of the motivation for case-based planning. Second, such a course is not in general possible; it depends on several of the assumptions we are trying to do without—assumptions of a stable world and complete knowledge. What is needed, instead, is a capability for *plan repair*.

The *adaptive planning* approach of Alterman (1985), and the *generate-test-debug* approach of Simmons and Davis (1987) are directed especially against the Execution-time Failure problem, as is (Hammond 1987). In adaptive planning, failure leads to replanning using semantically linked features; in generate-test-debug, replanning is guided by a causal description of the failure. Case-based planning also uses a causal description of plan failure, both to repair the current plan and to discover the features that will predict the problem in the future.

We have so far been treating the issue of failure as though it exclusively meant failure of a single plan to work correctly in execution, either through simple bad planning or through confrontation with unexpected conditions in the world. A planner dealing with multiple goals must also deal with possible failures deriving

from the interactions of the plans for two or more goals. Interactions where plans may interfere with each other have long been a focus in planning research. Here we will place some emphasis on another sort of interaction between plans, interactions in which some benefit could be derived from combining plans. If a useful interaction could be obtained, but the planner does not take advantage of this possibility, it has fallen into another sort of interactive failure. This failure to take advantage of potentially beneficial plan interactions constitutes the Missed Opportunities problem.

Clearly, the necessity for making use of opportunities—avoiding the Missed Opportunities problem—springs ultimately from considerations of efficiency; that is, from the concerns we have labeled the Immediate Complexity and Asymptotic Complexity problems. Indeed, our whole taxonomy of problems has been revealed as a tangled network of mutual dependencies. The moral is obvious: they cannot be solved singly and piecemeal. Active planning in a realistic domain requires combined work on all these fronts simultaneously.

Strategy mix depends on domain

In the previous sections we listed several fundamental problems encountered in the traditional approach of full preplanning under a closed-world assumption; examined several recent directions in planning, each one aimed at ameliorating selected items from that list of problems; and called for efforts to develop an integrated approach. But we will not get very far by arguing about these issues at the level of generality in the previous sections. There is no best (and of course this could only mean "currently-best") integrated approach to planning in general, for the simple reason that there is no such thing as planning-in-general.

As happens almost anywhere in AI or computer science, we have reached the point of confrontation with trade-offs. A planner trying to operate in an unstable world, of which it has incomplete knowledge, is forced to trade correctness for efficiency, first-time success for long-run success, planning time for execution time. If the planner is trying to solve an NP-hard problem in the real world, the dilemma can only become sharper.

The best—or let us only say the least unsatisfactory—resolution of these trade-offs is not given by general considerations but instead depends on the domain, the task, and perhaps an externally-decided performance-level criterion. To make our discussion more concrete, we will focus on the choices stemming from the domain and task used in the **TRUCKER** project, whose implementation is described in the latter sections of this paper.

TRUCKER is a planner operating in the domain of messenger-service scheduling. A dispatcher controls a fleet of trucks which roam a city or a neighborhood, picking up and dropping off parcels at designated addresses. (Our implementation uses Chicago and its Hyde Park neighborhood.) Transport orders are "phoned in" by customers at various times during the simulated business day, and the parcel delivery sequence and truck routing are adjusted to efficiently accommodate the new orders. The relevant sense of efficiency here includes both the cost of the planner's own efforts and the evolving delivery sequence and routing.

TRUCKER's task involves receiving requests from customers, making decisions about which truck to assign a given request to, deciding in what order given parcels should be picked up and dropped off, figuring out routes for the trucks to follow, and monitoring the execution of the plans it constructs. A number of limited resources must be managed, including the trucks themselves, their gas and cargo space, and the planner's own planning time. **TRUCKER** starts off with very little information about the world that its trucks will be negotiating; all it has is the equivalent of a street map, an incomplete and potentially inaccurate schematic of its simulated world.

Thus, this is the sort of domain and task where the problems contemplated in our earlier list all naturally arise. Traditional approaches to planning, with emphasis on exhaustive preplanning, would therefore be inadequate to this task for a number of reasons:

- **TRUCKER** lacks perfect information about its world.
- **TRUCKER** does not know all of its goals in advance – new calls come in that must be integrated with currently running plans.

- Planning time is limited. **TRUCKER**'s world does not wait for it to complete plans before new events occur.
- Even given perfect advance information, an optimal solution to the problem **TRUCKER** faces is computationally intractable. Even scheduling the pickup and dropoff points for a single truck to minimize travel time is a variant of the traveling salesman problem, which is known to be NP-complete.

To stave off the effects of the Planning/Execution Crowding problem, we might try a situated-activity approach. But the complexity of the domain rules out a pristine situated-activity/reactive-planning approach to this task. Very few pickups and deliveries would get done if the trucks were commanded by a frenetic dispatcher constantly issuing new instructions based only on the current locations of the trucks and the last transport-request received.

However, an appropriate modification within the reactive-planning school of thought, we believe, can be derived along the lines taken in (Firby 1987). We separate out classes of actions which require temporally-extended control (routing) from those which can be but the matter of a moment (navigation), and assign execution of the latter to semi-autonomous agents.

To deal with Asymptotic Complexity, we might want to cast our planner in a case-based mold, storing and re-using plans. The relevant plans here are the routes for driving from one given block to another. That's fine, as far as it goes, and indeed **TRUCKER** has a route-memory. But by itself this technique doesn't go nearly far enough. A delivery truck cannot afford to work sequentially through its list of orders, driving directly from the pickup point of each request to the corresponding dropoff point before dealing with the next request, even if the list has been put in some rational order. There will be a clear case of interaction failure, contributing to the Missed Opportunities problem, if the planner is not able to combine nearby stops. On the other hand, the planner will be swamped in the Immediate Complexity problem if it checks for all route-combination possibilities every time a new request is phoned in. Clearly, it needs something beyond the situated-activity and case-based components demanded so far, something to help it select reasonable occasions for making the computational effort to detect advantageous combinations.

To deal with these combinations, then, we would want to give the planner an opportunistic component, whose job it is to detect apparent opportunities for route-combination. But if hammered together in isolation, such a facility would introduce its own new computational costs. It would be at least problematic, and perhaps no net gain at all, if this component were introduced as a new planning expense on top of everything else—say, as a collection of daemons attached to each pending delivery request.

This consideration takes us from *opportunism* in general to the more specific model of *opportunistic memory*.

A pending delivery goal gets attached to memory structures which will be used or activated anyway when an appropriate opportunity for dealing with that goal arises in the normal course of other activity. In the **TRUCKER** domain, the relevant normal activity is simply that of (simulated) driving. With that step, we have asked for another component, an observing/understanding component which "parses the world" from the raw stream of incoming information. In the course of interpreting the presence of a certain recognizable building or other landmark as meaning that the truck has reached a now-identifiable location, it must find and access a memory structure corresponding to that location. Waiting in that memory structure is a notation about other delivery goals associated with that place—but waiting quietly, as a notation, not waiting busily, as a daemon. Thus our response to the Missed Opportunities problem has demanded that we deal with the Costly Information problem at the same time.

When an opportunity for route-combination arises, the planner must be able to take advantage of it by reorganizing the delivery plans. (And of course it should store the combination for later re-use.) Besides the potential interaction failures represented by missed opportunities, **TRUCKER** must also be prepared to deal with direct execution failures. For both these reasons, it requires a replanning component.

Structure of the TRUCKER Program

The **TRUCKER** planner is embedded in a demonstration program consisting of three modules: the world simulation, the map, and the planner itself. Trucks move through the world, along routes constructed by the

planner, assuming that the directions are valid, that there is gas in the tank, and so on (they may find out otherwise). The map is a schematic of the simulated world, but with considerably less information, lacking buildings, "visual" cues, one-way streets and other features. The simulated world, on the other hand, does contain cues of those sorts, which play an essential role in navigation. Though the program has to know where all the trucks are at any given moment, this information is not directly available to the planner; instead, it must construct and maintain this information as it goes along, "parsing the world." This intimate connection with locality provides the basis for having places remind the planner of possible opportunities.

The high-level agenda for a truck is a sequence of instructions about where to travel and what to do there. Typically it consists at any one time of alternating instructions for travel-steps and parcel transactions:

```
(GOTO (5802 S-WOODLAWN))
(PICKUP PARCEL-3)
(GOTO (920 E-55TH))
(DROPOFF PARCEL-5)
```

Plans of this sort are created as needed, and consumed piecemeal as each portion is executed. Each truck has such a plan. Portions not yet executed are available for reordering, cancellation, addition of new steps along the way, or transfer to another truck.

The planner also provides specific plans for the routes that trucks follow when executing a GOTO step. A route is represented as a series of turns, using street names and compass directions (with a start and stop instruction at the beginning and end). In particular, it is *not* a series of step-by-step or block-by-block instructions; a truck driving under the guidance of a route can travel several blocks without using a new portion of the route, until it must turn or make a stop. The route expanding the travel step `(GOTO (920 E-55TH))` in the delivery plan given above would be the following:

```
(START NORTH (5802 S-WOODLAWN))
(TURN EAST E-57TH)
(TURN NORTH S-CORNELL)
(TURN EAST E-55TH)
(STOP (920 E-55TH))
```

These pieces of knowledge are indexed by the place in the world with which they are associated.

Together, these provide the material on which the several active components demanded in the previous section do their work.

Reactive-planning component: central planning agenda

At the center of the **TRUCKER** implementation lies a "main loop" planning and execution supervisor, corresponding to the dispatcher in the domain model. The implementation is intended to connect with the ideas of (Firby 1987). The basic task of this component is to answer the phone, examine each new delivery order, and either assign it to a truck immediately or else decide to temporarily lay it aside.

The planner controls its own agenda by means of a request-based action queue, ordered by predefined priorities for various types of tasks. The central planning component is treated as time-bound. That is, almost all of the actions it can take, both those involving its own state and those involving the domain more directly, have costs in the simulated time-stream. As a consequence, it is designed to act in units of atomic actions that require little time singly; the atomic components of a complex action are placed in a priority queue, to be carried out when time allows. The contents of this queue at any given moment constitutes a tentative plan for the planner's actions in the near term. To achieve this atomization, most of the action-types built into the planner are molecular.

The most important molecular action is to try assigning a delivery request to a truck which will be able to handle it well. These are "or" packets. When a new order is received, the planner's only immediate response is to place such an assign-delivery-to-truck action somewhere in its queue. At some point that action is interpreted, and the result of that interpretation is to place four new packets into the queue.

```
handle-new-req-expansion:
      try-assign-to-truck-going-near    ; if reminded
      try-assign-to-idle-truck
      try-combine-with-unassigned-reqs ; if reminded
      dump-in-unassigned-reqs
```

By keeping the atomic actions generally inexpensive individually, even at the cost of multiplying their number, we prevent the planner from being tied up and uninterruptible when it should be noticing events in the world, a particular version of the Planning/Execution Crowding problem. But the main contribution of this architecture is against the Immediate Complexity problem, by avoiding projection of effects and enforcing an early (indeed, immediate) linearization of implicitly hierarchical structures.

Case-based component: route and combination memory

When a truck, working through its delivery agenda, completes one pickup or dropoff and prepares to drive on to the next, it requires a driving route from the dispatcher. If necessary, the dispatcher will consult the map and its memory of road conditions and typical speeds (a memory quite distinct from the map) in order to compute, by two-way best-first search, a near-optimal route from the truck's current location to its next stop. This computation is one of the two inherently expensive operations in **TRUCKER**, and the planner will avoid undertaking it if possible. The computation can be skipped if the planner already knows the desired route, having previously computed it and stored it. This, of course, is the core idea of case-based planning and we employ it very directly here: the planner cannot entirely avoid this expensive search, but it can avoid repeating it for the same locations. [1]

When two delivery requests are opportunistically combined and their routes are merged, the merged route is stored, along with the fact that these two pickup-dropoff location pairs proved combinable. This provides some interesting challenges in memory indexing, but otherwise the basic idea is still the same, and addresses especially the Asymptotic Complexity problem.

Opportunistic memory and replanning components: detecting and constructing plan combinations

TRUCKER merges requests in an effort to optimize over travel time. But it does so only when it encounters an opportunity to satisfy one request while it is actually running the route for a previous one, or if it has learned from a previous such opportunity that two requests are combinable. Initially, the effect of the queued action packets in the dispatcher is that **TRUCKER** runs requests in order of "call-in," assigning them singly to idle trucks until all trucks are occupied. It also links each new request with the memory nodes in its representation associated with the locations that would serve as opportunities for satisfying the request, *i.e.*, the pickup and delivery location. As the planner executes each stage of its plan, recognizing locations, it sometimes finds requests associated with locations that it is passing. When this happens, **TRUCKER** considers the possibility that the new request could be merged with the current plan—as well as the possibility that the resulting route should be stored and re-used.

Situated-activity and Understanding components: driving and navigating

Like these real drivers, **TRUCKER** cannot preplan all the driving steps involved in carrying out a sequence of deliveries. It first supplies the trucks with highest-level plans, a sequence of the locations where they are to stop for pickups and dropoffs. Only when such a step is ready for execution is it expanded into a plan at the next lower level, a sequence of major travel legs punctuated by turns or change of street name.

[1] This aspect was not our main theoretical emphasis in **TRUCKER**, so we did not implement certain interesting variations which suggest themselves as additional time-saving measures. New routes, for example, could be constructed by extending old ones; or the planner could model the city in terms of neighborhood centers, major intersections, and local "feeder" streets, and try to adapt any route found in memory which has the same start and end neighborhoods as the desired new route, or the same nearby major intersection.

This gives the planner the flexibility to rearrange the higher-level plans as needed for repair or opportunity, without wasting the effort of repeatedly changing the expansions at the lower level when the major steps get interleaved differently.

Summary

Recent developments in planning have separately addressed various of the major problems that arose when traditional planning ideas were presented with domains and tasks for which the assumptions of a closed world and complete knowledge do not hold. The new theories have shown considerable success in taming some of those problems, by regarding plan construction and execution as intimately tied together, and thus monitoring and guiding the execution of their plans. But even with this measure of success, no one of these theories can claim complete success, and none can be taken as the unique best direction in which planning research should go. We call for a pluralistic spirit and close attention to the dictates of the particular domains and tasks as a way of developing suitable planning systems.

References

Agre, P. E. and D. Chapman (1987). Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, AAAI, Seattle, WA, July 1987, 268-272.

Alterman, R. (1985). Adaptive planning: refitting old plans to new situations. In *Proceedings 7th Cognitive Science Society.*

Birnbaum, L., and G. Collins (1984). Opportunistic Planning and Freudian Slips. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, Boulder, CO, 1984.

Chapman, D. (1985). *Planning for Conjunctive Goals*, Technical Report TR 802, MIT Artificial Intelligence Laboratory.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of AAAI-87*, AAAI, Seattle, WA, July 1987, 202-206.

Georgeff, M. P. and Lansky, A. L. (1987). Reactive Reasoning and Planning. In *Proceedings of AAAI-87*, AAAI, Seattle, WA, July 1987, 677-682.

Hammond, K. (1987). Explaining and Repairing Plans that Fail. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

Hammond, K., T. Converse and M. Marks (1988). Learning from opportunities: Storing and re-using execution-time optimizations. In *Proceedings of AAAI-88.*

Hammond, K. and N. Hurwitz (1988). Extracting diagnostic features from explanations. AAAI Symposium on explanation-based learning, Palo Alto, CA, March.

Hammond, K., *Case-based Planning: Viewing planning as a memory task.* Academic Press, Cambridge, MA, 1989.

Hayes-Roth, B., and F. Hayes-Roth (1979). A cognitive model of planning. In *Cognitive Science*, 2, 1979, 275-310.

Kolodner, J. L., R. L. Simpson, and K. Sycara-Cyranski (1985). A process model of case-based reasoning in problem solving. In *The Ninth International Joint Conference on Artificial Intelligence.*

Simmons, R. F., and R. Davis (1987). Generate, test, and debug: combining associational rules and causal models.

Suchman, L. (1987). *Plans and Situated Actions.* Cambridge University Press, 1987.