*Proceedings of the 1987 Winter Simulation Conference*
A. Thesen, H. Grant, W. David Kelton (eds.)

# The Role of Simulation in Planning

**David P. Miller**
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

## ABSTRACT

Theoretical work in automated planning research has shown that the popular methods used for doing automated planning either cannot be extended into real world domains, or are computationally intractable. This paper discusses the paradigm of simulation-based planning which attempts to overcome these difficulties. Simulation-based planning uses some simulation techniques to gather information for guiding the planner. The resulting system can handle more complex domains with much improved performance over previous systems, while making only a small compromise in completeness. Implemented systems are described.

## 1. INTRODUCTION: THE NEED FOR SIMULATION

There are two major techniques used in current AI planning systems. These techniques are *situation-based* planning, typified by the GPS [Ernst69] and STRIPS planners [Fikes71], and *hierarchical least-commitment* planning, such as that done by the NOAH [Sacerdoti77] and NONLIN [Tate77] planners.

Situation-based planning is usually intricately linked to a first-order predicate logic (fopl) representation of the state of the world. In other words, the world situation and the actions being planned are represented as conjunctions of first-order statements. The planning system has available to it a list of the primitive actions which may be undertaken. These actions, usually referred to as *operators*, contain fopl statements specifying the preconditions needed to exist before this operator may enacted, the positive effects the operator will have on the world (fopl statements to be added to the conjunction representing the state of the world) and the negative effects (fopl statements to be deleted from the state of the world). The planner then runs a trivial simulation involving stringing together a list of operators that successfully transforms the initial state of the world into the goal state.

Using a graph theory-based form of analysis called region analysis, it has been shown [Joslin86] that the search involved in deriving a successful plan (string of operators) is intractable. In addition, plans that must satisfy a conjunction of two or more goals may be insolvable by situation-based planners, even in cases where a legal string of operators to satisfy all the goals does exist. These problems have been classified as *nonlinear*. By using macro-operators, [Korf85] it is possible to reduce the number of nonlinear problems that are unsolvable by situation-based techniques. However, the macro-operator technique is only feasible in domains that contain a high amount of symmetry (where an operator's effects may be removed by repeating an operator sequence). Finally, in cases where the planner cannot find a solution, it often will not terminate.

The problems above were largely eliminated by the advent of hierarchical least-commitment planning. Planners based on this paradigm used pieces of pre-compiled plans at different levels of abstraction as their building blocks for creating a new plan. Given a certain problem, the planner would search its library for a top-level plan that addressed the problem. This top-level plan would include subtasks; the planner would search the library for suitable plans for each of the subtasks, and then their subtasks, and so on. At each level the plans would be instantiated with information specific to the problem at hand. This information was sometimes used, under special conditions, to enforce an ordering between subtasks. But whenever possible the subtasks remained unordered.

Hierarchical planning provides significant guidance in the search for a feasible plan. The partial ordering of the plan steps (through least-commitment) allows a high-level plan to be flexible enough to accept the constraints of the low-level details. Unfortunately, planning of this sort still has several major problems. First off, while the search is greatly reduced, compared to situation-based planning, it is still intractable [Chapman87]. Second, planners based on the NOAH and NONLIN systems still rely on fopl as their representation. It has been shown that these representations can lead to incorrect conclusions when handling temporal and resource relationships [Hanks86]. Finally, the least-commitment strategy can mislead the planner into thinking it is solving a problem, when in fact the problem is either insolvable, or the planner has already made a fatal decision.

A simple example of this last problem occurs if there are several unordered tasks, each to be carried out at its own workstation, and having a deadline over the entire group of tasks. Least-commitment makes it impossible to estimate the amount of travel time needed to get to the individual workstations because there is no ordering between the tasks. If the distance between workstations is significant, the route taken between them can decide the success or failure of meeting the overall deadline. A least-commitment planner would develop the plans for each of the tasks and then search through the factorial number of possible ordering trying to find one that meets the deadlines.

One way to get around the problems mentioned above is to perform planning via task scheduling. In the most general sense, task scheduling involves using some mechanism to generate the low-level tasks that must be performed and then use some other algorithm to assign them the proper order — generating any transition tasks (such as moving from one workstation to another) that come about as a result of that order.

The DEVISER planner [Vere83] did a very simple version of task scheduling. It used the NONLIN planner to generate the tasks, and also did some bookkeeping on the time windows in which the tasks had to be performed. It then did a total ordering based on the propagation of the time windows, and added the transition tasks. Because DEVISER did all of its planning before any scheduling was done, the plans that were developed often could not be successfully scheduled if the tasks were tightly constrained. DEVISER's performance was therefore exponential because of the backtracking.

The remainder of this paper describes a different paradigm for doing task scheduling: simulation-based planning. In this paradigm task scheduling is fully integrated with the generation of subtasks and transition tasks. Simulations are performed on the partially developed plan, and that information is used to further guide the successive refinement and scheduling of the remainder of the plan.

## 2. AN EXAMPLE DOMAIN

A typical domain that requires simulation-based planning is one where resources, independent processes, and temporal constraints

exist. These factors occur in most real world domains such as planning production runs for a manufacturer, travel planning, and the day to day running of one's life.

The domain that will be used for illustration throughout the remainder of this paper consists of the semi-automated factory shown in Figure ??. This factory has two employees, three production machines, storage facilities, and a loading area. The factory can produce two different products: widgets and gizmos, that require particular production procedures and raw materials. The production machines consist of a lathe, a high speed lathe, and a recycler. Widgets are made out of plastic, while gizmos are metal. The lathes produce lots of shavings, which normally fall into the lathe's shavings bin. The recycler takes shavings as input and produces new blanks. Either lathe may be used for creating either product; the product depends on the raw material used.

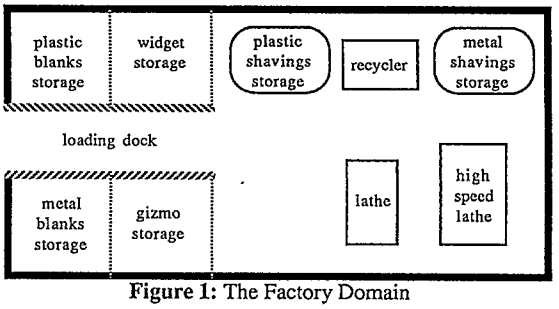


**Figure 1:** The Factory Domain

It takes time to move from one place in the factory to another. If a lathe is being used for gizmos, then it is producing metal shavings. The shavings bin for that lathe must be emptied at least every ten gizmos or it will fill up and the lathe will shut down. The shavings bin must be emptied every time the raw material is switched otherwise it will fill with a mixture of metal and plastic shavings which cannot be recycled. If widgets are being produced, then the bin must be emptied at least every 15 widgets. It takes the shavings from 20 gizmos or widgets to make a new blank of metal or plastic. The recycler can handle the shavings from upto 60 gizmos or 100 widgets, at one time.

To produce a product, an employee takes an appropriate blank (up to 20 blanks of any combination may be carried at a time) inserts the blank into the lathe (the high speed lathe can take fifteen at a time, the normal lathe five) and start the lathe running. The lathe will then grind out the products until it runs out of blanks, fills its shavings bin, or fills its output hopper (which can hold twice as many finished products as the input hopper may hold blanks). The finished products are then moved to the appropriate storage area. Shavings to be recycled are moved to the appropriate storage bin. The bins are periodically emptied into the recycler, which after a short time melts the shavings into new blanks. An employee then moves those blanks either to a lathe or to the appropriate storage area.

Trucks come into the loading dock to either pick up a finished order, or to drop off raw materials. An employee is needed to load and unload the trucks.

Given a particular set of orders, initial raw materials, and a delivery/resupply schedule, there is a huge number of different plans that may be attempted to meet the delivery schedule. In a real factory, the plan should include a large amount of slop time and material (as much as practical) so that unexpected situations, such as equipment failures or an emergency order, may be handled. The resulting plan must do more than construct the desired number of gizmos and widgets, and the plan must do more than meet the delivery and resource constraints. The plan should be one of the most efficient plans possible (to allow maximum overruns of time and materials), yet it must not take an exponential amount of time to construct the plan.

## 3. SEARCH AND SIMULATION

When faced with a constraint optimization problem, the normal response (at least since the advent of high speed computers) is to perform a search. In the problem domains that we are concerned about, the search area is at best exponential. In the domain presented above, it is, for all practical purposes, infinite. This is due to being able to put in arbitrarily sized delays (some of which are vital to the success of the plan) in between the different actions that the factory employees are to carry out. Since an exhaustive search of the problem space is impractical for all but the most trivial of problems, some metric (or set of metrics) must be constructed to be used as a basis for constraining the search.

Typical forms of best-first search look at the cost that has been incurred by the partial solution, and uses that as the metric for deciding whether or how that particular partial soultion should be extended. The straightforward implementation of this for automated planning falls short of what is needed in two major ways.

First, if the least-commitment paradigm of planning is used, then there will be no transition costs figured into the cost of the partial solution (because the order of the steps in that partial solution is not yet defined). The situation-based planning paradigm is simply not efficient enough for problems of the complexity posed above.

Second, the cost of a partial plan is often the inverse of the cost of the remainder of the plan. For example, if there are five steps to be done, three of which take a minute each and two of which take an hour each, then some partial plans that contain two steps will have a high cost, yet the total cost will not be much higher, while other partial plans will have a very misleading low cost so far. In many cases these differences will far outshadow any inherent efficiencies or inefficiencies developed in the partial plan.

The solution to the first problem is to perform a simulation of selected total orderings for the plan that's been created so far. The simulation will allow a detailed accounting of the transition costs, and pinpoint areas where the transitions are infeasible. The total orderings can be selected on the basis of scheduling heuristics. A wide variety of scheduling heuristics has been developed and are detailed in the OR literature [Graham77]. Additional heuristics have been developed for particular planning problems [Fox83], [Miller87]. It is not necessary at this point in the search process to locate the best ordering, only to explore in sufficient detail the probable quality of the best ordering for a particular set of steps.

The second problem also involves simulation in the solution. A good ordering (derived as described above) is combined with a high-level description of the remainder of the task. A simulation is carried out on the partial plan and its overall role in carrying out the task. The simulation uses the high-level descriptions of the remainder of the task to project the time and resources required for completing the task. The simulation can also be used to estimate what percentage of the solution (along a variety of metrics) has already been planned out, and how much there is to go. These calculations can be factored into the rating for that partial plan.

If the rating function is perfect, a best-first search will yield the optimal solution in time proportional to the length of the solution. The rating scheme above is not necessarily perfect, but if the simulations are relatively accurate, and the search is further constrained by performing some sort of beam search [Lowerre76], then near-optimal solutions may usually be found in linear-time.

## 4. SIMULATION-BASED PLANNERS

Two simulation-based planners have been developed using the basic scheme outlined in the previous section. The BUMPERS planner [Miller85b] creates sensor plans for a mobile robot system. The simulations the planner uses allow the system to quickly focus in on the plans that do not overuse a particular sensor. The simulations are also used to project the effects of using different motor speeds on the robot's effectors — to see whether the increased speed and overall plan effciency leads to unacceptable

coordination problems.

The FORBIN planner [Miller85a], [Dean87] works in a semi-automated factory domain similar to that described in Section 2. FORBIN does a type of hierarchical least-commitment planning. It uses simulations and task scheduling to rate various plans and as a filter for partial plans. Unlike NOAH or NONLIN, FORBIN uses simulations of the partial plans it is working with to check that there is at least one feasible total ordering. The plan remains partially ordered until completed — for future flexibility. But the planner is assured that it will eventually be able to create a totally ordered working plan, because it has checked the existance of such a plan by doing the total order simulations each time it makes a refinement to some high-level portion of the plan.

Simulation-based planners use a more detailed task language than has been typical of previous planners [Miller86]. The language used in FORBIN consists of two parts the *task expansion library* and the *causal theory*. The library of task expansions contains the available methods for doing any particular task (at a certain level in the hierarchy). There might be several methods of accomplishing a particular subtask. The task scheduler in the FORBIN planner runs simulations on the different methods to predict which would fit in best with the overall plan that is being constructed.

The causal theory provides the high level descriptions of subtasks that can be used in the simulations — before particular methods for those subtasks are chosen. The causal theory contains procedural information for running the simulations (e.g., the warm-up function for the factory recycler). The high-level subtask descriptors take the form of time and resource estimates for accomplishing the subtasks.

## 5. RESULTS AND CONCLUSIONS

Simulation-based planners have had success in domains where more traditional styles of planning have not been able to function. A limited domain simulation-based planner such as BUMPERS is able to handle virtually all problems that fall within its domain. By limiting the domain, the BUMPERS planner can use canned simulation routines which model the robot and its world in great enough detail to act as an accurate predictor for the planner and task scheduler.

The FORBIN system was designed as a more general purpose planner. All the domain specific knowledge is kept in the plan library and causal model. This has resulted in the knowledge structures for these libraries to be complex, and the libraries themselves somewhat difficult to design, maintain, and expand. To ensure that it is in fact possible to create these libraries, for a wide variety of domains, the level of detail with which they model those domains has been limited, hence limiting the predictive power of the simulation.

The result of these limitations sometimes causes FORBIN to fail to find a plan where a solution exists. The FORBIN planner can easily be modified to backtrack when it comes across a failure, but that would be courting the exponential performance that has plagued previous systems. We believe that FORBIN's power could be increased by incorporating more detailed domain knowledge for its simulation engines. To do this, without making the domain knowledge impossible to program, we are considering making a library of program modules, each specific to a particular class of domains. Each module would encode the general simulation information for a particular class of problems, leaving only the fine details to be encoded in the plan library and causal model. FORBIN would no longer be a domain independent planner, but rather a collection of planners, each capable of handling a small class of domains.

It has been shown by several researchers that complete domain-independent planning is NP-hard. To make planning tractable, completeness must therefore be sacrificed. In order for a planner to come up with a good plan most of the time, it must have some basis for choosing one partially developed plan over another for further development. The detailed knowledge provided by simulating a partial plan, and simulating some select subset of its possible completions, has proven a promising method for giving planners the needed clues to efficiently search their exponential problem space.

## BIBLIOGRAPHY


[Chapman87] Chapman, D., Planning for Conjunctive Goals, *Artificial Intelligence*, vol 32 #3, (1987), pp. 333-378.

[Dean87] Dean, T., Firby, R.J., Miller, D.P., *The FORBIN Paper*, Yale University Department of Computer Science Technical Report RR#550, July, 1987.

[Ernst 69] Ernst, George W. and Newell, Allen, *GPS: a Case Study in Generality and Problem Solving*, Academic Press, 1969.

[Fikes 71] Fikes, Richard and Nilsson, Nils J., STRIPS: A new approach to the applications of theorem proving to problem solving, *Artificial Intelligence*, 2 (1971), pp. 189–208.

[Fox 83] Fox, Mark S., *Constraint–Directed Search: A Case Study of Job–Shop Scheduling*, Technical Report CMU–RI–TR–83–22, CMU Robotics Institute, December 1983.

[Graham 77] Graham, R.L., Lawler, E.L., Lenstra, J.K., J.K., Kan, A.H.G., Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Proc. of Disctrete Optimization 1977*, Vancouver, B.C., Canada, August 1977.

[Hanks86] Hanks, S., McDermott, D., Default Reasoning, Nonmonotonic Logics, and the Frame Problem, in *Proceeding of the Fifth National Conference on Artificial Intelligence*, AAAI, Philadelphia, PA, 1986.

[Joslin86] Joslin, D.E., Roach, J.W., *An Analysis of Conjunctive Goal Planning*, Technical report TR-86-34, Virginia Tech Computer Science, 1986.

[Korf85] Korf, R., *Learning to Solve Problems by Searching for Macro-operators*, Pittman Publishing, 1985.

[Lowerre 76] Lowerre, B., *The HARPY Speech Recognition System*, Ph.D. Thesis, Carnegie–Mellon University, 1976.

[Miller 85a] Miller, D., Firby, R.J., Dean, T., Deadlines, Travel Time, and Robot Problem Solving, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, IJCAI, AAAI, Los Angeles, CA, August 1985, pp. 1052–1054.

[Miller 85b] Miller, D.P., Planning by Search Through Simulations, Yale University, Department of Computer Science, Research Report #423, October 1985.

[Miller86] Miller, D.P., A Plan Language for Dealing with the Physical World, Proceedings of the Third Annual Computer Science Symposium on Knowledge Based Systems, Columbia, SC, March 1986.

[Miller87] Miller, D. P., A Task and Resource Scheduling System for Automated Planning, in *The Annals of Operations Research: Approaches to Intelligent Decision Support*, Jeroslow, R.G., editor, 1987.

[Sacerdoti 77] Sacerdoti, Earl, *A Structure for Plans and Behavior*, American Elsevier Publishing Company, Inc., 1977.

[Tate 77] Tate, Austin, Generating Project Networks, *Proc. of the 5th Int. Joint Conf. on Artificial Intelligence*, IJCAI, Cambridge Ma, U.S.A, August 1977, pp. 888–893.

[Vere 83] Vere, Steven A. *Planning in Time: Windows and Durartions for Activities and Goals*, IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI–5/3 May (1983), pp. 246–267.

## AUTHOR'S BIOGRAPHY

David P. Miller is an assistant professor of computer science at Virginia Tech. He received a B.A. in astronomy from Wesleyan University in 1981, and a Ph.D. degree in computer science at Yale University in 1985. His research interests include automated planning, shop scheduling, robotics, and sensor simulation and interpretation. He has been a consultant to the Jet Propulsion Laboratory and the NSWC. He is the founder of the Artificial Intelligence Society of the Mid-Atlantic States and is a member of the ACM, IEEE, and AAAI.

David P. Miller
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
(703) 961-5605
miller@vtopus.cs.vt.edu