Can AI Planners Solve Practical Problems?

by
David E. Wilkins
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, California 94025
415-859-2057

May 21,1990

for publication in the Computational Intelligence Journal

Abstract

While there has been recent interest in research on planning and reasoning about actions, nearly all research results have been theoretical. We know of no previous examples of a planning system that has made a significant impact on a problem of practical importance. One of the primary goals during the development of the SIPE-2 planning system has been the balancing of efficiency with expressiveness and flexibility. With a major new extension, SIPE-2 has begun to address practical problems. This paper describes this new extension and the new applications of the planner. One of these applications is the problem of producing products from raw materials on process lines under production and resource constraints. This is a problem of commercial importance and SIPE-2's application to it is described in some detail.[1]

Key words: planning, reasoning about actions, knowledge representation, manufacturing

[1]The writing of this paper and the research described within was supported by the Australian Department of Industry, Technology and Commerce under Grant Agreement 16007, the Australian Artificial Intelligence Institute, and SRI International. Research performed at the Department of Civil Engineering, Stanford University, is also mentioned.

1 The Current State of Planning

There has been great interest in research on planning and reasoning about actions in recent years. In this paper, *planning* is defined as generating sequences of actions that will achieve given goals in a domain complex enough that the appropriateness and consequences of the actions depend upon the world states in which they are to be executed. This is a creative, synthetic process, not merely a matter of filling in certain slots with correct values. In particular, the planning system must keep track of and reason about differing world states at different points in time. This feature distinguishes the planning problem from similar problems such as scheduling, and makes planning inherently difficult as it involves the solution of several exponential problems [Wil88]. Faced with this overwhelming complexity, practical planning systems must balance epistemological and heuristic adequacy, retaining as much expressive power as is practical, yet making enough restricting assumptions so that a viable, efficient implementation can still be realized.

Many scheduling problems, for example, require constraints to be satisfied so that schedules can be correctly met, but do not require that the system reason about how the world changes as scheduled events occur. If the set of actions that must be performed, i.e., the process network, is known beforehand then it is a scheduling problem to assign resources and times to these actions. On the other hand, if the set of actions must be generated based on the current situation and the current goals, then the problem solution also involves planning. Other features of a problem can change it from a scheduling problem to a planning problem. For example, if some subset of the constraints in a scheduling problem will change depending on how another subset is satisfied, e.g., if the constraints on the afternoon's schedule will change depending on how the constraints on the morning's schedule were satisfied, then such a scheduling problem becomes a planning problem.

Scheduling problems are generally exponential and are certainly important, but they are simpler than problems that also require reasoning about the effects of actions on the world state. Most non-AI software used for "planning," such as CPM and PERT tools, cannot reason about the effects of actions and is suitable for a much simpler problem than the planning problem; often, such software merely offers bookkeeping facilities for a plan that has been generated by humans. Linear programming and operations research techniques often cannot express certain important constraints, cannot revise plans as they do not represent the causal relationships between activities, and require several hours of computation (see Section 7). Of course, these techniques can guarantee optimality while AI planners generally do not. Finally, ISIS [FS84] is an example of an AI system that can solve practical constraint-satisfaction problems but cannot reason about the different world states that occur after actions are executed.

As problem domains become increasingly complex, the ability to plan becomes more important. This is shown by the aforementioned example of the constraints on the afternoon's schedule changing. Similarly, if a system is to modify an existing plan or schedule in response to unexpected occurrences during execution, then reasoning about the effects of actions and the causal relationships between actions becomes necessary to determine which subplans remain valid or are amenable to modification. In most real-world situations, unexpected occurrences are the norm during operations, and it is important to modify plans quickly in response to these occurrences.

Despite recent AI work in planning, nearly all research results have been theoretical. Many researchers are solving problems involved in using logical formalisms to reason about actions [All83, McC80, McD82, Sho87], but almost none of this work has resulted in an implemented system. Most implemented AI planning systems have been either too inefficient to tackle real problems, e.g., FORBIN [DFM88] and TWEAK [Cha87], or not expressive enough, e.g., NONLIN [Tat76] and NOAH [Sac77]. GEMPLAN [Lan88] is another implemented planner that has not yet been applied to a practical problem; however, its method of localized reasoning may prove useful on certain types of problems. We know of no previous examples of a planning system that has made a significant impact on a problem of commercial importance.

This paper discusses an application of the SIPE-2 (System for Interactive Planning and Execution) planning system to a practical problem, and mentions results of applications in other domains. SIPE-2 is based on SRI International's original SIPE system, and includes all its features [Wil88]. SIPE-2 includes numerous improvements on SIPE, but the primary addition is the ability to reason about arbitrary orderings of actions. This has resulted in redesigning and reimplementing nearly every important algorithm in the system.

This paper summarizes SIPE-2 and describes differences between SIPE and SIPE-2 that are crucial in the construction and manufacturing domains. This is followed by a detailed description of a manufacturing problem and SIPE-2's application to this problem. Finally, there is an analysis of system performance on this and other problems, and a comparison of this approach to other approaches, both AI and non-AI (e.g., operations research).

2 Application of SIPE-2 to Practical Problems

Faced with the overwhelming complexity of planning, SIPE-2 has attempted to balance epistemological and heuristic adequacy. It retains enough expressive power to be useful, yet makes enough restricting assumptions to produce a viable, efficient implementation. It is implemented in Symbolics CommonLisp and runs on machines supporting Symbolics Gen-

era software.[2] Unlike most AI planning research, the design of SIPE-2 has taken heuristic adequacy as one of its primary goals.

SIPE-2 provides a domain-independent formalism for describing operators (the planner's representation of actions), and utilizes the knowledge encoded in these operators, together with heuristics for handling the combinatorics of the problem, to plan means to achieve given goals in diverse problem domains. The plans include causal information so that the system can modify these plans in response to unanticipated events during plan execution. Unlike expert systems, AI planners are capable of generating a novel sequence of actions that responds precisely to the situation at hand.

SIPE-2 is more advanced than most implemented AI planners, primarily because it can reason about resources, can post and use constraints, and can employ a deductive causal theory to represent and reason about different world states. It retains much of the efficiency of the STRIPS assumption [FN71] while avoiding some of its disadvantages through the use of the above mechanisms. Automatically, or under interactive control, the system generates possibly nonlinear plans containing conditionals that will achieve the given goals when executed in the given initial situation. It can intermingle planning and execution, and can accept arbitrary descriptions, in the language used to describe the domain, of unexpected occurrences during execution and modify its plan to take these into account.

To achieve heuristic adequacy, SIPE-2 incorporates special techniques for solving a number of problems; these techniques are described in detail elsewhere [Wil88]. Some of the more important problems for which specialized algorithms have been developed are these:

- Determination of the truth of a formula at a particular point in a plan
- Deduction of context-dependent effects
- Unification of two variables once they have constraints on them
- Handling parallel interactions
- Detecting and resolving resource conflicts
- Searching efficiently through the space of possible plans

Previous applications of SIPE include the blocks world, tower of hanoi, travel planning, movement of aircraft on a carrier deck, and control of a mobile robot [Wil88]. Of those domains, planning the actions of the mobile robot was the only problem that was reasonably complex and realistic (see Section 6). SIPE-2 has recently been applied to two problem areas

[2] Currently, this includes all Symbolics machines and, with appropriate Symbolics boards, Apple MacIntosh IIs and Sun workstations.

of practical importance. One is the generation of project plans for construction,[3] and the other is the scheduling of process lines in a manufacturing environment.[4]

A description of the construction problem can be found elsewhere [Kar89]; briefly, the plan generated by SIPE-2 for a three-story office building is among the most complex construction plans so far generated automatically. Although resources and time were ignored, the number of jobs and their interrelated constraints are on a realistic scale. In Section 5, the application of SIPE-2 in the manufacturing domain is described. This shows by example that problems of practical importance are beginning to be addressed by this system. In Section 6, system performance in all these problem domains is presented along with a description of the size and complexity of each domain.

Before describing the manufacturing application, we describe the recent extensions and enhancements to SIPE, not described previously in the literature, that enable SIPE-2 to be effective in the construction and manufacturing domains. In these domains, the planner had to generate thousands of nodes in its plans and reason with world models that contained thousands of predicate instances. Because of this, several planning algorithms were made more efficient. Other enhancements include extended temporal reasoning, improved plan critics, extended resource reasoning, and a vastly improved graphical interface. The most important new component is the ability to produce arbitrary orderings of unordered actions. Section 3 describes problems posed by unordered actions and SIPE's approach to these problems, and section 4 describes SIPE-2's approach. Section 4 also describes the specification of goals as external conditions — a new capability that is needed to take advantage of the ability to arbitrarily order actions. The new ability to specify parallel reusable-resources is described in Section 5.

3 Reasoning about Partially Ordered Actions

SIPE-2 has removed the restrictions SIPE previously made on possible ways to order actions. First we summarize the problems involved in reasoning about partially ordered actions; a planner that supports partially ordered actions is defined as a *nonlinear* planner. A plan is partially ordered if it contains actions that are unordered with respect to each other, i.e., actions for which the planner has not yet determined an order and which may possibly be executed in parallel. If a planner has the ability to represent and reason about partially-ordered

[3]This application was done by Nabil Kartam, Department of Civil Engineering, Stanford University.

[4]This application was done by the author, with the able help of David Morley, and supported by The Australian Artificial Intelligence Institute and the Australian Department of Industry, Technology and Commerce under Grant Agreement 16007.

plans, it can avoid committing to a particular ordering of actions until it has accumulated information that permits determination of the correct ordering. This can avoid an exponential search of all possible plan orderings as is typical of a linear planning system. For example, 52 actions unordered with respect to each other have as many possible orderings as the number of ways a deck of cards can be shuffled. This is an enormous number, and a nonlinear planner can often avoid searching such prohibitively large search spaces.

Another advantage of nonlinear planners is that actions can remain unordered in the final plan, thus permitting parallel plans to be generated; this is a necessity in many domains. The planner must ensure that no harmful interactions occur among unordered actions. A nonlinear planner must order actions during the planning process, but actions can often remain unordered until such time as the planner discovers the order it wishes to impose. For this reason, nonlinear planners are often referred to as employing the least-commitment approach.

A planner's *truth criterion* is its algorithm for determining whether a formula is true in a particular world state. From an efficiency standpoint, the truth criterion is the heart of the planning system as it is a primitive operation used in every planning algorithm. As Chapman has shown [Cha87], nonlinearity makes the truth criterion NP-complete, given a reasonably powerful representation. To ensure soundness when the truth criterion requires that a query predicate is true, the system needs to determine all possible orderings that would make the query true, then constrain the system to only allow these orderings. This process is exponential, and is the underlying reason why nonlinear planners that make only sound deductions at each step are not practically useful. Of course, the soundness of the final plan should be guaranteed. As discussed below, heuristic algorithms for a nonlinear truth criterion offer a method for avoiding some of the combinatorics in planning.

3.1 SIPE-2's Handling of Partially Ordered Actions

SIPE-2 incorporates mechanisms and heuristics for circumventing this NP-complete problem. The high-level description of these mechanisms in this subsection applies to SIPE as well as SIPE-2, although different implementations are needed to achieve these mechanisms in the two systems (see Section 4). The most powerful heuristic for avoiding combinatorics is incorporated in SIPE-2's truth criterion (TC) which does not always enforce the ordering constraints that would ensure soundness. Thus, invalid plans may be temporarily produced. The system relies on *plan critics* that check for and correct problems in these plans. These critics are applied after each *planning level*, i.e., an expansion of the whole plan to a greater level of detail, though the user can control the frequency of critic application as is appropriate for the problem domain. One could view this as doing the work of enforcing validity every so often instead of at every primitive step.

An important technique for making plan critics efficient is distinguishing between the main effects and side effects of an action. While plan critics guarantee the truth of main effects at every point where the effects are needed in the plan, they ignore interactions between side effects that are not used by the plan.

The TC merely proves that there is one possible ordering of the partially ordered actions that makes the query predicate true, without enforcing that order. Proving that there is one possible ordering is efficient, as the ordering itself need not even be calculated, and there is certainly no need to calculate all such orderings. Roughly, this is because the system only needs to find an action that (1) achieves the query predicate, and (2) does not have an action that negates the query predicate that necessarily follows it. This algorithm can produce temporarily invalid plans, since different calls to the truth criterion may assume different implicit orderings. Thus, the idea of defining conditions that make planning operators sound, as Lifschitz attempts to do for STRIPS [Lif87], is not directly applicable — nor, in my opinion, can it lead to an heuristically adequate, nonlinear planning system.

There are other reasons for temporarily generating invalid plans. A nonlinear planner must reason about how actions that may take place concurrently interact with each other. Often, two actions interfere with each other if they are executed at the same time, and the planner must recognize and correct these situations in order to generate correct plans. Nonlinear planning poses other problems as well. A planner must consider the possibility of achieving a goal by adding ordering constraints instead of by planning some action. Given a goal G, and some action A that achieves G and is unordered with respect to it, then a planner can achieve G by ordering it after A. This may or may not be the correct choice for producing a valid plan; in general, the correctness of such a choice cannot be predicted without completely investigating all its consequences, which entails a combinatorial search.

If problems are detected by the critics, *solvers* are applied to modify the plan, possibly adding ordering constraints to the actions. Solvers in SIPE-2 are more powerful than those in previous classical planners, as they use the replanning actions of the system to modify plans [Wil88], possibly removing subplans in order to make better plans. The invalid plans temporarily produced have not been a problem in practice [Wil88], primarily because appropriate heuristics and algorithms have been developed over time. The TC has proved to be a useful compromise that provides the user with a powerful tool to produce useful plans efficiently.

Now that SIPE-2's approach to partially ordered actions has been summarized, the next subsection describes the need to generate arbitrary orderings of unordered actions, and Section 4 describes recent extensions that allow the system to do just that.

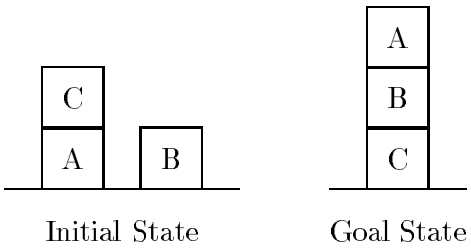

Figure 1: The Sussman Anomaly

Figure 2: SIPE-2 Plan for Two-Conjunct Sussman Anomaly

3.2 Considering All Possible Orderings

SIPE had a nonshuffling restriction [Wil88] that made the critics and solvers simple and efficient. Given a set of subplans that are unordered with respect to each other, SIPE would only order them by putting one whole subplan before or after the others. This greatly reduced the number of possible orderings, but it could not produce all possible shuffles of the primitive actions in the subplans. While this assumption is reasonable when planning the actions of a single robot in an environment without other agents, it has proven too restrictive in problems with multiple agents or conjunctive goals. As will be explained, effects of this restriction appeared in the blocks world, but use of SIPE in a construction domain clearly showed that this restriction was not acceptable.

SIPE correctly solved all the standard block-world problems involving three blocks. However, with the nonshuffling restriction, it obtained a nonoptimal solution for the Sussman Anomaly (shown in Figure 1) when the initial goal is given as the two unordered conjuncts (On A B) and (On B C). (The optimal solution was produced for the three-conjunct problem where (On C Table) is given as a third goal conjunct.) In the two-conjunct case, SIPE

produced a plan in which B is first moved onto C and then moved back to the table. This shortcoming is a direct consequence of the assumption that unordered subplans will not be shuffled together since finding the optimal solution requires separating the goal of clearing A in the (On A B) subplan from the rest of the subplan, as can be seen in the SIPE-2 plan shown in Figure 2. With the extension which removes the nonshuffling restriction (described in the next section), SIPE-2 produces optimal plans for all 3-block block-world problems.

In addition to the evidence from the Sussman Anomaly, planning the construction of buildings and structures shows the inadequacy of this nonshuffling restriction. Consider the problem of building a deck that is supported by four beams which are in turn supported by four columns sunk in concrete footings. The natural hierarchical decomposition of this problem is to plan the construction of the four beams in parallel and then lay the deck on the beams. Constructing a beam consists of constructing its two supporting columns. Ordering constraints must be added since two different beams require the construction of the same column. With the nonshuffling restriction, a planner can only produce a plan that constructs two footings, then constructs two columns, then lays a beam across those two columns, and then proceeds to the next footing and column. This is not acceptable as the workers who lay the concrete have to wait between the second and third footings for the carpenters to construct columns and beams. A plan that had all footings in parallel followed by all columns in parallel would be much preferable.

With the capability described in Section 4, SIPE-2 produces a plan with only a minimal number of ordering constraints for this problem, as shown in Figure 3, taken from SIPE-2's graphical interface. Plans can be considered to be a set of nodes linked to each other by ordering links (arrows). Nodes represent actions, control information, goals, and other information required by the planner. The diamond-shaped nodes depict control information – the S and J nodes represent split and join nodes respectively which enclose a set of parallel branches that are initially unordered with respect to each other. The hexagonal nodes represent processes which are actions to be executed, and the square nodes represent phantoms which will not be executed, but represent necessary information about what conditions must be true at particular points in time.

SIPE-2 produces this plan in 3 seconds, and produces plans which involve four such decks in one structure in less than 23 seconds. One possible ordering implicit in this plan, and produced by a regrouping algorithm in SIPE-2, is the one shown in Figure 4 with all the footings, columns, and beams in parallel. The regrouping algorithm groups first all nodes that could be executed first, then all nodes that could be executed first after that group, and so forth. The above regrouping algorithm happens to work well in this simple construction domain, but the user may wish to add ordering links in a more complex fashion. How to allow the user to specify desirable further orderings of the least-committed plan is an interesting

Figure 3: SIPE-2 Plan for Constructing a Deck

Figure 4: Regrouped Plan for Deck Construction

research issue.

4 Parallel Links Extension to SIPE

Because of the unacceptability of the nonshuffling restriction, SIPE-2 was created to allow all possible orderings of unordered actions. Many planning systems already have this capability, such as GEMPLAN and TWEAK, but no such systems provide heuristic adequacy at present. Extending SIPE required a useful, efficient, implementation that would retain its efficient truth criterion while taking into account arbitrary ordering links that may exist. In this section, we briefly outline this implementation and mention some of the problems that had to be solved to obtain heuristic adequacy.

The system now permits *parallel links* which are defined as successor and predecessor links that can be placed between any two actions in a plan that are unordered with respect to each other. Such links will be referred to as p-predecessor and p-successor links to distinguish them from successor and predecessor links that have always been part of the plan. This is done because the plan-traversal algorithms must follow these links differently. These algorithms follow links from node to node and recurse upon themselves when split-join nodes are encountered. However, parallel links may link nodes inside a split-join to nodes outside that split-join, thus invalidating the recursive algorithms used in SIPE.

Accommodating parallel links required the development of several complex algorithms to add the necessary capabilities to the TC and the plan critics, as well as new routines for displaying the plans produced. The intention of this section is not to communicate how SIPE-2's critics or TC work in detail (the SIPE versions of these algorithms are described elsewhere [Wil88]), but to communicate the complexity of the problem and show the types of algorithms that must be developed to address it.

The representation of parallel links is complicated by the use of hierarchical planning levels and SIPE-2's ability to represent alternative plans in different contexts. There may be a number of parallel links at any one node, e.g., there may be multiple p-predecessors. However, some links may not apply to the current plan and context but rather apply to an alternative expansion of the plan. There are also complications in copying these links down to more detailed expansions of the plan, since some nodes may be expanded to the more detailed level while other nodes may not be. Thus the TC and other algorithms have to be able to calculate the lowest-level expansion of the node to which they are linked. For efficiency reasons, the system fills in links at lower levels when they can be calculated, rather than recomputing links by following higher-level links down to the lowest existing level.

4.1 Implementing Parallel Links

The changes to the critics and solvers were fairly straightforward. The solvers now add ordering constraints by adding p-successor and p-predecessor links, such as the link from the put C on Table process to the Clear C phantom in Figure 2. The critics need to determine if two nodes are still unordered with respect to each other before checking for resource conflicts or harmful interactions. Determining such ordering relationships becomes a basic operation in many parts of the planner, but is a nontrivial algorithm which must check if either node is ordered before the other. To see if $node_1$ is ordered before $node_2$, the system must find the common split-join that includes both nodes, then attempt to follow p-predecessor and predecessor links from one set of nodes (the nodes from $node_1$ to the common join) into a second set of nodes (the nodes from $node_2$ to the common split).

Parallel links make the TC much more complex, although heuristic adequacy is retained by incorporating the same heuristic of looking for one possible ordering that makes a predicate true. This can still be done efficiently in SIPE-2. The TC traverses the nodes in a plan from the end of the plan to the beginning, taking into account the effects of each node. The algorithm naturally recurses on itself when split-join nodes are encountered. The TC must wait to process the effects of nodes with p-successor links until nodes that necessarily come after these nodes in other branches of some enclosing split-join are processed. Since split-joins may be nested arbitrarily deeply, this may involve popping up through several levels of recursion for additional processing and then returning to the computation that was suspended. Furthermore, the suspended computation cannot simply wait for its p-successor to be processed because the TC may determine an answer on that branch before it ever processes the p-successor node.

Another complication in the TC occurs when it is trying to determine the truth of a predicate at a node (called the current node) inside a split-join. The TC marks all nodes that must precede the current node by virtue of p-predecessor links, then starts traversing the plan from the outermost join that includes the current node. This is the same TC algorithm except that it ignores unmarked nodes.

A necessary subtask of implementing parallel links was an enhancement of the output routines. As plans get more complex, the issue of how to present the plan so that a human can understand it becomes important and nontrivial. For example, the least-constrained plan in Figure 5 for finishing a room in the construction domain is hard to express without graphics (and this is one of the simpler construction plans). SIPE-2's graphical interface has become essential for determining the structure of the plans now being produced. SIPE's interface had to be extended in order to debug the other algorithms that were developed. It was necessary to include the capability to regroup unordered actions as shown in Figure 4, since plans

Figure 5: SIPE-2 Plan for Finishing a Room

with parallel links can be difficult to interpret. Figure 6 shows SIPE-2's domain-independent graphical interface displaying part of one of the schedules produced in the manufacturing domain; a domain-specific interface was also implemented for this domain.

4.2 External-Condition Goals

With the new capability provided by parallel links, the user needs new ways to specify domain-specific knowledge about orderings. This was accomplished in SIPE-2 by allowing certain goals in operators to be specified as external-condition goals. Such a specification indicates that the goal is not to be achieved by planning a sequence of actions, but rather will be achieved by actions external to this operator and "phantomized" by inserting an ordering link. Such goals correspond to "unsupervised conditions" in the NONLIN planner [Tat76]. External-condition goals are necessary in the construction domain: e.g., if finishing the walls is proceeding in parallel with electrical wiring, there must be some way to specify that the portion of the wiring inside the walls must be finished before the walls are completely enclosed. This can be accomplished by having a external-condition goal of finishing the internal wiring placed before the goal of closing the walls in the plot of an operator which specifies the actions for finishing the wall.

Figure 6: SIPE-2's Graphical Interface

5 Planning in Manufacturing Domains

SIPE-2 has now reached a point where it can be useful in practical problems. This is shown by its recent application in the manufacturing domain.

In competitive markets, manufacturers must be able to reconfigure factories or reschedule activities in response to shifts in customer demands for product and contractual obligations covering deliveries. In addition, events that affect operations are continually taking place, e.g., equipment breaking, workers not showing up. Thus a planning system that can quickly respond to such changing situations can result in greater productivity and performance.

Mathematical modeling is appropriate for making long-range predictions about the behavior of a system and can help in designing a factory for maximum throughput at minimum cost. However, operations-research techniques are not well suited to shorter term tactical planning where it is important to have a clear understanding of the interactions between processes and the consequences of any given course of action in qualitative terms. These techniques cannot support replanning after unexpected occurrences as they do not represent the causal relationships between activities in a plan/schedule. They often cannot express

all the constraints that actually exist in the factory. Furthermore, the combinatorics of the problem space results in such large execution times that computer programs based on such techniques must be run off-line. Thus these approaches are of little help after an unexpected occurrence invalidates the plan that has been produced by an overnight computer run. Such unexpected occurrences often happen within the first hour of factory operation and generally result in the plan being discarded.

We have used the scheduling of packaging lines at one of the world's largest breweries as a problem domain. There are several reasons for this: it is a real problem in terms of practical importance and numerous, complex constraints; it is a fairly typical problem in that most manufacturing operations will have similar problems; our techniques are fairly well suited to this problem; and, importantly, the brewery was willing to cooperate and provide access to the necessary domain information.

The problem is typical because it is an example of producing products from raw materials under constraints. Many manufacturers have a number of production lines that can produce a variety of products. The production lines often have overlapping capabilities, but each line also has its specializations. Production is often interrupted by events such as the unavailability of some raw material, equipment malfunction, or resource shortages, and the production planner must modify its plan in response.

The brewery packaging plant exemplifies this domain. The domain actually implemented in SIPE-2 is described here. This problem is a simplification of the actual situation at the brewery, but it contains the important aspects of the situation. There are over 300 products which are assembled from the raw materials of beer, cans or bottles, tops, labels, wrapping, and cartons.[5] Manpower availability is quantized into a number of work shifts which have been planned by human long-range planners. Beer and bottles can sometimes be diverted from one product to another, but cans, labels, and cartons cannot.

This section first describes the manufacturing problem that was addressed, then explains the representation of this problem in SIPE-2, then outlines the system's generation and modification of plans, and, finally, presents performance data.

5.1 Planning Production Lines

SIPE-2 was applied to one of the most critical parts of the production scheduling problem: daily operations planning. Two of the most important advantages of using AI planning technology are its ability to deal with many complex constraints, and its ability to formulate new plans in response to changing circumstances. Both these properties are crucial to daily

[5]The number of products is large because different states and countries require different information on the labels. Thus many products have the same beer but different labels or cans.

operations planning, and both are weaknesses of the operations-research approaches that are often used on similar problems.

While weekly or monthly long-range planning is also suitable for AI planning technology, this will more easily and correctly be achieved after the basic, underlying physical constraints have been incorporated. Thus, an automated daily-operations planner will be able to provide input and information that can be used as the basis for higher level decisions. In summary, the reasons for first addressing daily operations planning are the following:

- Daily planning involves most of the complex, physical constraints
- Fast reaction to unplanned events is important
- Automating the foundation will provide a basis for automation of higher-level planning

In addressing the daily scheduling problem, the number of shifts that management has decided to run is accepted as an input to the system, along with the current state of orders, inventories, equipment, and manpower. Some of the many physical constraints that have been incorporated into the implementation are mentioned below.

The domain consists of six production lines fed from several beer tanks through beer lines. Each beer line runs to one particular cellar and can only connect to the beer tanks in that cellar. Production lines can therefore contend for the use of both beer tanks and beer lines. The beer in the beer tanks has been produced more or less in accordance with a planned fermentation cycle computed from the order backlog as it stood two weeks ago. However, the planner must react to the current demand for products, as well as the current state of manpower, equipment, and available beer and materials. Other considerations include the many physical constraints on the combinations of beer, package type, and production line. There is a cost involved in switching the type of beer that flows through a beer line as the line must be flushed, resulting in the loss of a significant amount of beer. For most products, there is a backlog of orders which is subject to change. The problem is to generate plans that meet as many of the orders as possible, while meeting all physical constraints and minimizing waste from flushing of lines. The system must also provide help in modifying plans when unanticipated problems occur during their execution.

An important point to consider when judging the usefulness of SIPE-2 for this type of planning is that the effort described below involved only six man-months of work. Of this, about three man-months were spent extending the system and three man-months were spent encoding and debugging the domain knowledge. While the implementation has certain limitations, the scope is nevertheless broad for the amount of effort involved.

5.2 Ontological Issues

Two natural but conflicting ontologies for the representation of the problem domain presented themselves. One is to have the planner solve goals of filling orders. This has several disadvantages in the SIPE-2 framework: not all goals can be met when scheduling a particular set of shifts, e.g., when producing a schedule for the next day; knowledge of what is best to do next must be incorporated in the choice of goals to expand next rather than in the choice of operators used to expand a goal; and, it is not clear how to manage the temporal aspects of the domain.

The alternative ontology, which has been used in the implementation, is to have the system plan to solve goals of scheduling production lines. This works very well in the SIPE-2 framework. The system is still driven by the orders that exist, as operators used to solve scheduling goals find the best orders to fill. In this ontology, the knowledge about which order is best to fill can be incorporated in the preconditions of the operators. Another advantage is that time flows forward in the plan, and a running schedule for each line corresponds naturally to the way the brewery's human schedulers produce plans.

This ontology required the implementation of an extended notion of reusable resource in SIPE-2. Previously, SIPE would order actions that had resource conflicts [Wil88]. For example, when trying to drive two nails, the planner would do one before the other if only one hammer were available. In this production-line ontology however, a resource conflict indicates an invalid plan as the lines *must* be run in parallel to get maximum throughput. SIPE-2 now provides an additional notion of reusable resource, namely, one that must be used in parallel. This was a straightforward extension of the system. Reusable resources are implemented by posting Optional-Not-Same constraints between two variables that are resources in actions that are unordered with respect to each other. To implement parallel reusable resources, the system simply posts Not-Same constraints instead, to prevent conflicts from occurring [Wil88].

5.3 Representation in SIPE-2

The implementation of the brewery domain uses predicates and SIPE-2's sort hierarchy to describe all products, the current orders, the current stock levels, the materials requirements of each product, the locations of each beer line and beer tank, the actual manpower turnout, the actual machine availability, the actual materials availability, and other information about the domain. The initial world state is described by 2063 predicate instances: to the best of our knowledge, this is considerably larger than any problem previously solved by an AI planning system. Figure 7 gives the number of predicate instances in the initial world for each

```
Size of input domain in number of predicate instances:
 Instances of WAITING-ON-STOCKS: 5
 Instances of STOCKS-UNAVAILABLE: 5
 Instances of SEVERE-SHORTFALL: 1
 Instances of SLIGHT-SHORTFALL: 1
 Instances of EXCESS: 1
 Instances of UNAVAILABLE: 1
 Instances of CONTAINS: 10
 Instances of BOTH-LINE: 3
 Instances of OPERATIVE: 7
 Instances of LEVEL: 35

 Instances of APPROPRIATE-LINE: 263
 Instances of IS-STOCK: 34
 Instances of REQUIRES-BEER: 199
 Instances of REQUIRES-HARD-STOCK: 797
 Instances of IS-ORDER: 166
 Instances of STOP-TIME: 21
 Instances of START-TIME: 21
 Instances of BEER-LINE: 8
 Instances of BEER-TANK: 21
 Instances of STOCK-PRODUCT: 42
 Instances of PRODUCT: 182
 Instances of CARTON-PACKING-RATE: 7
 Instances of CARTON-VOLUME: 11
 Instances of USES-PACKAGE-TYPE: 213
 Instances of SHIFTS: 7
 Instances of SPECIALTY-LINE: 2

69 dynamic predicates, 1994 static predicates.
2063 predicate instances in initial world state.
```

Figure 7: Predicate Instances in Manufacturing Domain

predicate name. The predicates will not be described in detail, but the names are mnemonic so figure 7 further helps to describe the size and structure of the domain.

Operators describe the actions available to the planner and how to achieve particular goals. They include the resources required and the conditions under which the operation can proceed. Operators encode knowledge of what orders are preferred for scheduling next, how to connect packaging lines to beer tanks, how to start and end shifts, how to avoid flushing beer lines when possible, and other such process oriented knowledge. Physical constraints on transporting raw materials from stores inventory to the packaging line have been adequately modeled in operators; this involves satisfying constraints on using beer lines to feed several production lines from beer tanks, as well as ensuring that all necessary raw materials are

Operator: Connect-bt-pl-noflush
Arguments: beer-tank1, packaging-line1, beer-line1, cellar1, beer1;
Purpose: (connected beer-tank1 packaging-line1);
Precondition:
(beer-tank beer-tank1 cellar1),
(beer-line beer-line1 cellar1),
-(connected-bt beer-tank1 beer-line1),
(contains beer-tank1 beer1),
(or (clean beer-line1) (contains beer-line1 beer1));
Plot:
Parallel
Goals: (connected-bt beer-tank1 beer-line1);
Goals: (connected-pl packaging-line1 beer-line1);
End Parallel
Process
Action: check-connection;
Arguments: packaging-line1,beer-line1;
Resources: beer-line1;
Effects: (connected beer-tank1 packaging-line1);

Figure 8: SIPE-2 Operator for Connecting a Beer Line

available and meet the constraints of the packaging line being used.

Figure 8 shows a SIPE-2 operator, Connect-bt-pl-noflush, for connecting a beer tank to a packaging line when the line does not need to be flushed. This case is separated from the flushing case so that the system will try this operator first, because of the given preference order of operators, and thus prefer actions that do not flush a beer line. This operator will be described in detail to indicate how SIPE-2 is used to represent actions in this domain.

An operator is applied to achieve goals that match its *purpose*, in this example, connecting a packaging line to a beer tank. The *arguments* slot of this operator specifies three additional variables used in the operator that are constrained to be in the following classes by virtue of their variable name: cellar, beer line, and beer. The precondition of an operator must be true in a given world state before this operator can be applied. Preconditions can be used to encode many types of knowledge, e.g., physical constraints on the domain, and preferences for certain actions. The first predicate of the precondition in Connect-bt-pl-noflush constrains the cellar1 variable to be the cellar containing beer-tank1. The second predicate constrains beer-line1 to run to that same cellar. The third predicate constrains beer-line1 to be a beer line that is not already connected to this beer tank; in this case, some other operator can accomplish the goal more efficiently. The fourth predicate constrains beer1 to be the beer in beer-tank1. Finally, the last conjunct of the precondition checks that beer-line1 does not

need to be flushed; this is the case when the line is clean or it already contains the same beer.

The *plot* of an operator specifies a template for further elaboration of a plan. A plot is a network of actions and goals that will accomplish the purpose of the operator in any situation satisfying the precondition. The plot of Connect-bt-pl-noflush specifies that the two ends of the beer line can be connected in parallel. This is followed by an action which tests that the connection has been established and declares beer-line1 as a resource.

SIPE-2 uses its new abilities to post parallel links and reason about parallel, reusable resources to correctly allocate both beer tanks and beer lines to each packaging line. Allocation of such resources is trivial: the user simply declares the beer line and beer tank variables in his operators to be resources, and the system will ensure there is no conflict. For example, simply declaring beer-line1 as a resource in Connect-bt-pl-noflush will cause the system to avoid conflicts in the use of beer lines. Parallel links are used to order early shifts/runs on one line before later shifts/runs on another line, thus enabling such shifts to use the same resources.

The current implementation represents all products and accepts the number of shifts, current manpower, current stock outages, and state of current equipment as input. It attempts to fill arbitrary orders that are also given as input. The planner establishes connections between production lines, beer lines and beer tanks, taking advantage of connections that already exist, and schedules as many production runs as are needed to fill each shift. The generated plan includes actions to assure that all needed materials are present for each run and satisfies all physical constraints, such as constraints on the use of different containers on different lines.

The system uses consumption rates to calculate the changing levels of orders, the changing availability of raw materials, and the start time, end time, duration, and level of production of each run and shift as a function of the size of the order and the packaging rate of the particular production line. The numerical values of time, levels of orders, and levels of beer tanks are all represented using SIPE-2's numerical variables and distinguished *level*, *produce*, and *consume* predicates [Wil88]. Figure 9 shows the plot of an operator that solves the goal of scheduling a packaging line at a certain time and thus updates these numerical values. Several predicates in the precondition of this operator choose an appropriate product to bottle based on orders and other factors. This is done by placing constraints on the *product*1 variable. All variables obtain their constraints initially either by matching of the operator's precondition or by specification in the operator's arguments slot.

The first goal in the plot is to connect an appropriate beer tank to the packaging line. By solving the *connected* goal, the planner will assure that the appropriate beer is available for the bottling run. The plot specifies that this condition should be maintained by the execution monitor until the bottling run is finished; this will occur when the appropriate number of

Plot:
Goal: (connected beer-tank1 pack-line1);
  Protect-until: (consume product1 num-cartons2);
Process
Action: load-stock;
  Arguments: product1;
  Effects: (loaded pack-line1 product1);
  Protect-until: (consume product1 num-cartons2);
Process
Action: bottle;
  Arguments: pack-line1, product1, num-cartons2,
    beer-tank1,start-time1, duration1, beer-volume1;
  Resources: beer-tank1;
  Effects: (consume product1 num-cartons2),
    (consume beer-tank1 beer-volume1);
  Duration: duration1;
Goal: (schedule pack-line1 shift1 current-time2 stop-time1);

Figure 9: SIPE-2 Plot for Scheduling a Bottling Run

cartons have been consumed. The default is to protect conditions through the last action in a plot. However, because of the scheduling-goal ontology, the plot of this operator ends by specifying another scheduling goal which causes the system to recurse and continue to schedule actions until the stop-time is reached. The connection of the beer tank should only be maintained through the bottling action and not through this recursive scheduling goal.

The arguments slot of the operator is used to define the values of variables such as *num-cartons2* and *duration1*. The variable *num-cartons2* is declared in the *arguments* slot of the operator as *num-cartons2 is (cartons duration1 pack-rate1)* which posts a function constraint [Wil88] that will cause the variable's value to be computed by calling a Lisp function (*cartons*) that takes the duration and the bottling rate of the line as arguments. Similarly, the *duration1* variable has a function constraint which computes the length of a bottling run as a Lisp function of several arguments.

Process nodes in plots can be used to specify either a primitive, executable action or another operator that will further elaborate the plan. The first process node in the plot in Figure 9 specifies an action of loading the necessary raw materials (other than beer) on the packaging line. The next process specifies the bottling run, and has several arguments including *duration1* and *beer-volume1* which have function constraints that will cause their value to be computed. This process declares the beer tank as a resource just as Connect-bt-pl-noflush declares beer lines as resources. At this level of abstraction, beer lines have not been introduced into the plan; that will happen when the *connected* goal is solved. The

effects of this process are to reduce the level of beer in the beer tank and to reduce the level of outstanding orders for the product. These two effects are encoded by *consume* predicates which cause SIPE-2 to appropriately update the levels of these two numerical quantities over time. Similarly, the *duration* slot on this process will update the level that represents time. Finally, the *schedule* goal will cause the system to schedule another bottling run from the now current time (after *duration1*) until *stop-time1* is reached.

5.4 Generating and Modifying Plans

By applying operators in the preference order given, the system will take advantage of any existing connections to beer lines and avoid unnecessary flushings. Similarly, these preferences first try to fill high priority orders. The resulting schedule minimizes flushings for the orders it has scheduled and schedules as many of the highest priority orders as possible during the given shifts. Because we have chosen the ontology of goals for scheduling production lines, there is always a solution that can be found without relaxing constraints; this matches basic assumptions built into the system.

SIPE-2 currently quits as soon as it finds a valid plan, i.e., has scheduled all given shifts completely, though it would be straightforward to implement a best-first search using the context mechanism if a good measure for the utility of a plan is provided [Wil88]. Good plans are produced by the system because domain-specific knowledge about search control and the utility of plans is encoded in the operators. For example, operators encode preferences on what orders to fill and what beer lines to use. While all constraints described above have been incorporated, there are aspects of daily operation that have not been addressed. For example, this implementation does not form groups of individuals with certain skills into one crew (as must be done when workers fail to report to work).

Producing a plan that schedules dozens of orders on six production lines with around twenty separate product runs, with their corresponding needs for different connections and materials, requires less than four minutes on a Symbolics 3645. To produce one such plan with no backtracking requires the generation of 1100 action and goal nodes (at all planning levels). Another 1000 control nodes are produced in the plan. How the solution time varies as the problem is changed is discussed in Section 6.

Figure 10 depicts the plan produced in a Gantt chart. This is a simple translation of the actual plan which is a PERT-type structure with many hundreds of nodes. Ignoring all nodes except bottling actions, the translation uses the product being bottled, the start time, and the end time to draw and label the Gantt chart, and inserts vertical dotted lines to denote different shifts. This same plan is depicted graphically in Figure 6 which shows the first few action nodes for the last three production lines; these include actions for connecting and

Figure 10: Gantt Chart of Plan for 11 Shifts on 6 Lines

flushing beer lines. Information about the assumptions underlying various subplans is also included in each plan to help with replanning; these are not shown in either figure.

The information on the right of Figure 10 provides more details of the second run of Golf-Lager-Ottumwa on line PL-12, in response to a mouse click on the Gantt chart. This information includes the scheduled start and stop times of the run and an inventory of the materials needed. It was necessary to bottle Golf-Lager-Ottumwa for 9 hours to fill an outstanding order, so the system decided to make a 1 hour run at the start of the second shift to finish filling the order.

An important motivation for using AI planning technology is the ability to modify plans quickly during execution. SIPE-2's execution monitor accepts arbitrary descriptions of unexpected events in the language used to describe the domain, and is able to determine how they affect the plan being executed [Wil88]. It uses a set of replanning actions to modify plans, possibly removing subplans in order to make better plans. In many cases, it is able to retain most of the original plan by making changes in that plan to avoid problems caused by these unexpected events. It is also capable of shortening the original plan when serendipitous events occur.

Figure 11: Modified Plan after Lid-A is Unavailable

Let us consider an unexpected occurrence during the execution of the plan in Figure 10. Suppose that a scheduled shipment of Lid-A did not arrive by the end of the first shift. The run of Golf-Lager-Ottumwa that begins the second shift can be seen to require 55,440 of Lid-A. SIPE-2 modifies the plan by removing all subplans that require Lid-A. In this example, the planner removes the last two bottling actions on line PL-12, together with all their associated actions of making connections and loading stock. The deleted subplans are replaced with goals of scheduling the production lines for the open time period, and this new plan, with unsolved goals, is given to the planner as a problem to solve. The solution is shown in Figure 11. Line PL-12 has been rescheduled to bottle Pella-Ale, one of the few products with outstanding orders that can be bottled on this line and that does not require Lid-A; see the right side of Figure 11. This modified plan is produced in one minute, a response time that makes it reasonable to consider using the planner on the factory floor to schedule daily operations and respond to new events.

6 Performance of SIPE-2

Our claims regarding efficiency rest on system performance in solving actual problems. It is not possible to prove lower bounds on computation for any input because the representation provided is powerful enough to express large and combinatorial constraints. Furthermore, some algorithms inside the planner address exponential problems. In practice, efficient performance depends upon axiomatizing one's domain so as to take advantage of system heuristics and algorithms.

In this section, we review SIPE-2's performance on all the domains mentioned in this paper. Since acceptable computational bounds cannot be proven, actual performance data are important for judging the usefulness of a planning system. While choice of hardware or software may vary execution times by an order of magnitude, such times are still important because they can vary several orders of magnitude across AI planners. The units in which the times are expressed are significant, and the numbers simply provide a datapoint. Solution times expressed in seconds and minutes for planning problems like those described here mean the combinatorics of the problem have been addressed — an exponential algorithm could have solution times expressed in centuries or millenia.

Unfortunately, research in AI planning and knowledge representation, unlike most other areas of computer science, has generally failed to address issues of system performance: it has neither developed reasonable measures for comparing task difficulty, nor has it standardly reported performance data for the tasks undertaken. Although measures such as CPU-time expended are not entirely sufficient as a measure of a planning system's performance, we nonetheless believe that the field will be advanced by the disclosure of such data. These numbers will change in the face of hardware and software advances, but progress can be gauged by changes in their order of magnitude. Thus, for example, if a new planner that solves particular problems is described, it is useful to know whether the solutions took seconds, minutes, hours, or days. The fact that the answer is sometime hours, or even days, is no reason to withhold this information from the community.

Typical data for SIPE-2 plan generation are shown in Table 1. Various sizes are given in this table to help judge the size of the problem. The *all nodes* column refers to the number of goal/process nodes created when generating, with no backtracking, one solution to a typical problem in the domain. This count includes nodes at all hierarchical planning levels, but does not include control nodes, e.g., split and join nodes, which often make the actual number of nodes in a plan nearly double the number shown. The *primitive nodes* column gives two numbers; the first is the number of process nodes in the final plan, only at the lowest planning level; the second is the number of phantom nodes in the final plan at the lowest planning level, these are goal nodes that are already true at the point where they occur. Phantom

Domain	All nodes	Primitive nodes	World predicates	Oprs	Time (secs)
block world	20	3-6	20	5	1
tower of hanoi 3	220	7	35	14	10
tower of hanoi 4	800	15	35	14	80
travel	30	5-1	300	35	3.5
mobile robot	175	10-16	250	50	20
house construction	160	50-10	1	25	5
office construction	1450	115-110	112	10	370
beer production	1100	250-20	2000	40	210

Table 1: Performance of SIPE-2

nodes are used by the execution monitor and add as much complexity to planning algorithms as do the process nodes; thus these algorithms must traverse a plan that is the sum of these two numbers. The *world predicates* column is the number of predicate instances that describe the initial world state. Although not shown in the table, the amount of information in the sort hierarchy would also be relevant to determining the size of the problem domain. The *oprs* column refers to the number of operators used to describe the domain, and the *time* column gives execution times in seconds on a Symbolics 3645 processor for a typical problem in each domain. Similar times are obtained on a Macintosh II with a MacIvory board.[6]

Problems in an extended block-world (i.e., one that permits more than one block to be on top of another and poses problems such as "get some red block on top of some blue block") are solved in 1 second. The tower-of-hanoi problem with 3 discs is solved in 10 seconds, while the 4-disc problem requires 80 seconds. The latter involved refining the plan over 26 levels of detail and generating 800 process/goal nodes. The travel-planning domain has 35 operators and over 300 predicate instances describing the world, and it requires solutions containing about 30 goal and process nodes. Such solutions are generated in 3 to 4 seconds.

The simple indoor mobile-robot world consists of five rooms connected by a hallway, the robot itself, and various objects. The rooms were divided into 35 symbolic locations that included multiple paths between locations, which greatly increased the amount of work done by the planner. The initial world is described by 222 predicate instances, about half of which were deduced from deductive operators. The description of possible actions includes 25 action-describing operators and 25 deductive operators. The operators use four levels

[6]Times on both Symbolics and Macintosh machines vary from run to run depending on many factors (e.g., free memory, fullness of the symbol table), so the given times are only approximate.

of abstraction in the planning process. The planner produces primitive plans that provide actual commands for controlling the robot's motors. This low level of abstraction requires the planner to generate hundreds of goal nodes to generate one plan, yet SIPE-2 takes only about 20 seconds to formulate such a plan completely, or 9 seconds for an executable plan if the planner intermingles planning and execution. This is acceptable performance, as the robot requires several seconds to move down the hall.

Results for two problems in the construction domain are given. The first is the house problem that was solved by NONLIN [Tat76]; this is given for comparison purposes. The problem is simple, SIPE-2's input is much more readable and concise than that of NONLIN, and SIPE-2 solves the problem in 5 seconds. NONLIN required a comparable amount of time (20 seconds) to produce the solution. However, NONLIN's cumbersome input requirements meant the system had significantly less work to do to solve the problem because of the additional input information.

The second construction problem is a three-story office building. Since resources and time were ignored, the planner's effort was primarily spent in obtaining the correct orderings among the more than 100 tasks that were, for the most part, initially unordered. This problem makes extensive use of the parallel-links capability and requires about 6 minutes to solve. Most of this time is spent by the plan critics determining which of the enormous number of possible orderings to impose.

As already described, producing a plan in the beer manufacturing domain that schedules twenty separate product runs on six production lines requires less than 4 minutes. To produce one such plan with no backtracking requires the generation of 1100 action and goal nodes while another 1000 control nodes are produced. To modify such a plan when a raw material becomes unexpectedly unavailable can take only a minute.

Interestingly, the three larger domains each stress a different part of the system. In the robot domain, the planner spends around 70% of its computational effort deducing the effects of its planned actions. By comparison, no time is spent deducing effects in the construction domain, and very little in the manufacturing domain. In the construction and manufacturing domains, the effects of actions were mostly known apriori, but in the robot domain the effects depend upon where the robot is and what it is holding when it moves.

In the construction domain, most of the computational effort is spent by the plan critics determining orderings for actions. In the manufacturing domain, most of the effort is spent in determining the truth of the preconditions of operators. This is attributable to a number of features of the domain: operators have large preconditions with as many as a dozen predicates, the plans have hundreds of nodes, and the world state is described by thousands of predicate instances.

The solution time in the brewery problem appears to be linear in the number of bottling runs that are scheduled. This is not surprising as each run requires the matching of large operator preconditions to choose an appropriate product to bottle, in addition to planning actions to supply all the raw materials for that run. Adding extra products and orders to the world description has a negligible effect on performance. Adding extra production lines increases planning time only linearly with the number of runs that will be scheduled on the new lines. This is primarily because the interactions between lines are handled by resource reasoning. If the tasks were interconnected in more complex ways, as they are in the construction domain, then the computational resources used by the plan critics could increase significantly as more lines were added.

It is hard to make efficiency comparisons, as it appears that no AI planning systems have been tried on problems of similar complexity, in most cases because such problems cannot be effectively handled. We know of no other AI planning system that can solve these or similar problems with a comparable amount of effort. FORBIN, which makes some concessions in order to gain efficiency, is described as being slow on even simple problems [DFM88]. TWEAK does not have heuristic adequacy as one of its design goals, and would probably be less efficient than FORBIN. Drummond describes his NEWT planner [DC88] as not solving block-world problems because of memory limitations until a "temporal-coherence" heuristic was added to the system. With this heuristic, the system was able to solve simple block-world problems, allowing only one block on top of another, in "a matter of minutes." Furthermore, NEWT was designed with a concern for efficiency. GEMPLAN solves simple block-world problems in less than one minute, and solves the three-disc tower-of-hanoi problem in 17 minutes on a Sun 3, and 7 minutes on a SPARC station.[7] In our experience, more expressive planning systems cannot achieve even this level of performance, e.g., a planner based on formal logic with frame axioms took hours to solve simple block-world problems.[8]

7 Conclusion

We know of no other AI planning systems that could handle a problem as large as the brewery scheduling and still respond in reasonable time. NOAH, NONLIN, and probably TWEAK are not expressive enough, while TWEAK, NEWT and FORBIN would appear not to be heuristically adequate [DFM88]. GEMPLAN has not yet been applied to a practical problem, and its heuristic adequacy will likely rest on how effective its method of localized reasoning will be on a particular problem.

[7] Personal communication.

[8] These results were never published, but obtained through personal communication.

Nonplanning technologies can be applied to this problem, and ISIS is a strong candidate. It would appear that, with a certain amount of creativity, the brewery problem could be translated into the job-shop scheduling constraints provided by ISIS. It is not clear how much effort this would involve or how the resulting system would compare to SIPE-2 in terms of either execution time or quality of answer. SIPE-2 provides a more general replanning capability than ISIS, and also provides interactive capabilities.

Except for systems with operations-research techniques, most commercially available "planning" software does not generate schedules/plans. The primary advantages of an AI planner over operations-research techniques are the following:

- AI representation languages allow representation of some constraints that cannot be expressed by linear equations.
- Resources are allocated with no violation of resource constraints.
- Planners like SIPE-2 can be run interactively, letting a human make crucial, high level decisions while the system ensures that all the details are correctly worked out.
- Because plans are produced in minutes, various "what-if" analyses can be run to produce and compare alternative plans.
- Planners like SIPE-2 can modify their plans in seconds or minutes in response to unexpected occurrences, thus significantly reducing down-time of production lines while permitting a higher level of order fulfillment. Surprises are ubiquitous in the factory and often quickly render useless plans produced by linear programming techniques, since such plans cannot be modified to respond to new situations.

This implementation shows that AI planning techniques are approaching the point where significant problems can be addressed. Producing production plans like those described here can be difficult for humans because there are so many constraints to be considered. For example, a schedule produced by human planners sometimes turns out to be unexecutable because all the right connections cannot be made to the beer tanks, or because some particular lid is unavailable. To schedule a production run, it is necessary to check the level of orders, check if the beer is in a beer tank, check if the beer tank is available, check if a beer line that can connect to the beer tank is available, and check if all labels, lids, bottles, and cartons are available. The problem is well suited to SIPE-2 because there is nearly always some beer-order whose constraints are satisfied; thus, relaxation of constraints is unnecessary. It is primarily a matter of keeping track of all the details. SIPE-2's ability to post constraints allows it to obtain solutions efficiently, often with no backtracking as most of the knowledge for choosing a reasonable product to schedule is incorporated in the preconditions of operators.

It is important to note that the brewery implementation described here involved only six man-months of work. This shows the power of domain-independent planners for representing new domains. There are many limitations: the lack of a search through alternative schedules/plans, the ignoring of certain constraints (e.g., manpower), the inability to relax constraints, and the user interface. However, these issues could not be addressed within the scope of this project. They should be the focus of future research – in particular, use of SIPE-2 in a particular domain would often involve development of a reasonable, probably domain-specific, search algorithm. Of the limitations mentioned above, only relaxation of constraints would be a large effort to incorporate, and our ontology for the brewery domain makes this unnecessary. SIPE-2 already incorporates the foundation for many types of search algorithms as it saves alternative plans and their contexts, and its interactive capabilities should make development of a user interface easier. Future research could use AI planning techniques to provide support for longer range planning, and could integrate the planning system with existing information processing systems.

Two features of the implemented system are expected to make it useful in the brewery: user interaction and replanning. The system can be run interactively, allowing the user to participate in the decision-making process. This allows a human expert to formulate plans based on his knowledge and judgement in a fraction of the time it would take without the AI planning system. The planning system takes care of all details, some of which would otherwise be ignored by the human expert because of the large number of constraints to be checked.

An important problem in this factory, as in most, is that production is often interrupted by unplanned events such as the unavailability of some raw material, equipment malfunction, or resource shortages. The planner has causal information relating the actions in its plans, and can use this information to modify plans during execution without having to replan completely, as shown in the example in Section 5.4. The ability of the system to modify the plan in response to unexpected events has the potential for significantly improving plant performance.[9]

[9]The system described here is a prototype and not in actual use. Such use would require an improved user interface, increased robustness, and probably other extensions. The brewery has funded a project to develop a prototype for actual use.

References

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery*, 26(11):832–843, 1983.

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[DC88] M. Drummond and K. Currie. Exploiting temporal coherence in nonlinear plan construction. *Computational Intelligence*, 4(4):341–348, 1988.

[DFM88] T. Dean, R. J. Firby, and D. Miller. Hierarchical planning involving deadlines, travel time, and resources. *Computational Intelligence*, 4(4):381–398, 1988.

[FN71] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[FS84] M. S. Fox and S. Smith. ISIS - a knowledge-based system for factory scheduling. *Expert Systems*, 1(1):24–49, 1984.

[Kar89] N. Kartam. *Investigating the Utility of Artificial Intelligence Techniques for Automatic Generation of Construction Project Plans*. PhD thesis, Stanford University, Department of Civil Engineering, Stanford CA, 1989.

[Lan88] A. L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4(4):319–340, 1988.

[Lif87] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1987.

[McC80] J. McCarthy. Circumscription – a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McD82] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Sac77] E. D. Sacerdoti. *A Structure for Plans and Behaviour*. Elsevier, North Holland, New York City, NY, 1977.

[Sho87] Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, Cambridge, MA, 1987.

[Tat76] A. Tate. Project planning using a hierarchical nonlinear planner. Department of Artificial Intelligence Report 25, Edinburgh University, 1976.

[Wil88] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1988.