SPAN: Integrating Problem Solving Tactics

Daniel L.S. Berlin
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

This paper describes SPAN, a system designed to integrate a variety of problem solving tactics in a coherent package. The paper discusses some of the tactics that had been used in previous systems to overcome the combinatorial explosion that is inherent in any planning problem. It then continues with a description of SPAN architecture. Two case studies are presented. The first, from the blocks world, is already implemented, and the second, from the domain of bridge playing, is in the coding stage. SPAN'S limitations are discussed and directions for further research are considered.

I INTRODUCTION

Much of the activity in artificial intelligence can be thought of as problem solving, so it is not surprising that, over the years, a lot of effort has gone into developing automatic problem-solvers. These efforts at producing domain-independent techniques usually have concentrated on solving simple problems.

In a 1980 revision of an article first published in 1979, Earl Sacerdoti presented an overview of problem-solving tactics [Sacerdoti 79] [Sacerdoti 80]. In that paper he states:

"To date, there has been no successful attempt known to this author to integrate a significant number of the tactics we have described into a single system."

This failure to integrate these tactics into a coherent package has been one of the main reasons that general domain-independent planning has met with limited success in solving non-trivial problems.

SPAN is a system for integrating a large variety of tactics into a cohesive package. The general architecture, along with those domain-independent parts necessary for planning in the relatively simple domain of the blocks world have been already implemented in the LOOPS programming language on the 1100 series XEROX personal work stations. Other domain-independent parts necessary for planning in more complex domains are currently being added.

SPAN'S architecture allows us to combine the insights developed in Sacerdoti's NOAH [Sacerdoti 77], Sussman's HACKER [Sussman 75], and Waldinger's goal regression system (independently developed by Warren [Waldinger 77] [Warren 74]); all of which were for ordering conjunctive sub-problems. The same framework is used to integrate these methods with tactics for choosing between alternatives. These techniques include splicing alternatives together (when neither is guaranteed to succeed), in addition to domain-specific comparisons of probability. A generalization of Berliner's B* algorithm [Berliner 78] can also be integrated into the system, to facilitate planning in a competitive domain.

This work was funded in part by NASA Grant #NAG 5-261, DARPA Contract #N00039-83-O0136. and a fellowship from NL-Industries. Computing resources were supplied by XEROX-PARC and the SUMEX computing facility at Stanford (NTH Grant #RR-0785-ll).

II HANDLING THE COMBINATORIAL EXPLOSION

Typically, the world is modeled by a series of propositions which describe what is true in the world (i.e. its state) at a particular time. An action is usually modeled as a transformation from propositions that hold in one world model to propositions that hold in the new world model after the action has been executed. These transformations are referred to as operators.

It may be the case that an operator requires certain conditions to be true in order for its action to be executable. These conditions are referred to as preconditions. An operator can be viewed as a solved problem, with the initial state specified by the preconditions, and the goal specified by the initial state and the transformation of the propositions.

In the simplest planning case, each operator corresponds to a single action in the real world. These operators are known as primitive operators. Typically, in any one state many different operators can be applied because their preconditions are true. Furthermore, it is not always clear which operator gets one closer to the goal state. For this reason, simple search techniques face a combinatorial explosion, and thus are unable to solve anything other than very elementary problems.

Much of the history of planning has been concerned with getting around this combinatorial explosion, and thus increasing the applicability of automatic problem solvers. Three ideas, in particular, have been useful in dampening the explosion. The first of these is the concept of abstraction. The second is the notion that some sub-problems (particularly the difficult ones) can constrain the number of solutions to other sub-problems, and so should be tackled first. The third is the idea that, if possible, one should always apply the best operator for getting from the initial state to the goal.

The advantage of abstraction is quite simple. If one can break down a large problem into a series of smaller problems, then solving all of the smaller problems will be easier than solving the original problem. The reason for this is that the breakdown into smaller problems is linear, whereas the effort to solve any particular problem is exponential.

Many versions of abstraction exist, including ignoring some details in the state description, ignoring some operators, or combining some operators to form a macro-operator. In the end, all of these techniques are equivalent to creating abstract operators. An abstract operator is like a primitve operator in that it has preconditions and transformation rules. However, unlike primitive operators, it cannot be translated immediately into actions that can be carried out in the real world. Instead, it must be refined into a sequence of primitive operators. These primitive operators may add further preconditions of their own, and may specify further transformations. Thus an abstract operator has abstracted out some of the detail involved in solving the problem it addresses. This notion of abstraction is at the heart of criticality lists in ABSTRIPS [Sacerdoti 74] and the generalisation of the logic operators in GPS [Ernst 69].

The second point, that some problems constrain others, is used to determine which sub-problem to attack first. This is closely related to abstraction, since one of the criteria for defining an abstract operator is that it restrict the number of options available to solve other problems. Indeed, one can view a partial ordering of which sub-problems to attack first as a mapping of the initial problem into an abstract domain that ignores the details of the other sub-problems. This point forms the basis of Stefik's constraint satisfaction system [Stefik 80]. It also forms the basis of an even more extreme form of problem solving: scripts or skeletal plans [Friedland 79]. With scripts, once the abstract operator (ie. script) has been chosen, all that remains to be done is to instantiate the variables of primitive operations that compose it. The selection of primitive operations to compose the abstract operator is automatic.

The third point is that it is not always possible to know which operators to choose and in what order to choose them, but that intelligent choice of operators can substantially improve the problem solving. This has led to a number of strategies, including the "don't choose until you have to" strategy of NOAH, the "make a random choice and fix it later" strategy of HACKER, and the "choose an arbitrary order but change the order of the goals as necessary" strategy of Waldinger's goal regression system.

All of these insights and tactics have some merit, but is it possible to develop a system that unites these approaches, making use of each of them when appropriate? SPAN is such a system.

HI SPAN ARCHITECTURE

As mentioned earlier, a problem can be represented by an initial state and a goal state. Solving the problem consists of finding a sequence of primitive operators, whose consecutive application will result in a final state for which the goal conditions are true.

At the top level of the SPAN planning system there is a scheduler with an agenda of tasks. These tasks are domain - independent, or, more accurately, their domain of expertise is planning. A typical example of a task is *ordering conjunctive sub-problems.* These tasks are represented as objects with slots to store relevant information. For example, a task to order *conjunctive sub-problems* will have pointers to the sub-problems. In addition, each task has an attached procedure that performs the task. This procedure is executed when the task is selected from the agenda.

This mechanism is similar to the meta-planning of MOLGEN [Stefik 80] [Stefik 81]. The agenda here corresponds to its design layer. The problems associated with the tasks correspond to the planning layer, and each system has a simple interpreter. However, in SPAN we have dispensed with a strategy level. In MOLGEN, the only strategy choices available were least-commitment and guessing. Not all the design operators are affected by differences in strategies, and local knowledge about the current state of the plan is likely to determine which strategy is the better choice, so this strategy knowledge is dispersed to the procedures that execute the design tasks. As in MOLGEN, preference is given to least-commitment, and arbitrary choices are only made as a last resort.

The key problem with any agenda-based system is deciding which task to do next. Selecting the wrong task can result in a lot of wasted planning effort. Several schemes exist for solving this problem. One technique is to provide priorities for each of the types of tasks. Unfortunately, global differences in the types of tasks do not always provide sufficient information to get an optimal ordering. Another is to poll each active task on the agenda (whose preconditions are satisfied) to see if it should be performed and to choose the one that is most confident the conditions are right for its execution. This involves a large overhead each time a task is to be selected. SPAN actually uses a variant on this second technique: It assumes that each active task is confident that it can be performed, and so selects one at random. However, the procedure that performs the task has the ability to suspend its execution. Thus, if a task is selected and decides it should not be done at this time, it has the ability to tell the system to choose another task. This produces a close approximation to the polling system with much less overhead.

As a result of the decision to randomly choose an active task, the agenda actually consists of three separate lists: a list of active tasks, a list of suspended tasks, and a history list of completed tasks. At each cycle the scheduler randomly selects an active task and performs it. If there are no active tasks, then the suspended ones are reactivated.

We will now examine the various types of planning tasks in SPAN.

IV PLANNING TASKS

Solving a problem (when only the initial stale and goal are known)

The system first checks to make sure that the initial state is completely defined. This means that there is a direct link from the intial state of the original problem to the initial state of this sub-problem, and that this link does not go through unsolved sub-problems. The reason for this is to guarantee that what could be known about the initial state is known. This is not to say that everything about the initial state is known; there might be information that is hidden from the planner, but it does guarantee the system is detecting real differences between the initial state and the goal. If this is not true then the task suspends itself, until such time as it becomes true.

Assuming the initial state is defined, this general problem is approached using means-ends analysis. Means-ends analysis is used because it is more flexible than forward or backward chaining. The goal is compared to the initial state and differences are determined. A sub-problem is proposed for each difference detected. These sub-problems are treated as an unordered set of conjunctive goals, whose collective solution constitutes a solution to the original problem. For each of these sub-problems a task is posted to find a sequence of operators that eliminates its difference. This is also known as reducing the difference detected. In addition, if more than one difference was detected, then a task is proposed to order the sub-problems. Obviously, if only one such difference was detected, then there is only one sub-problem to be ordered.

Reducing a detected difference

When a difference is detected, a task is posted to reduce that difference. This task indexes a procedure that uses domain-specific knowledge about how to reduce the detected difference. The procedure may do many things. One possibility is to delay its execution (by suspending the task) until further information becomes available from other parts of the plan. If it does execute, it may use a special purpose algorithm (eg. a routing algorithm in a robot planner). Alternatively, the procedure may propose an operator or a set of alternative operators.

Proposed operators may be abstract or primitive, and may be related to each other in any one of a number of ways. They may be an unordered set, or in a strict sequence as in a script.

Refining an abstract operator

When an abstract operator is proposed it needs to be refined. This refinement process invokes a domain-dependent procedure. The options available at this time are very similar to the options available when reducing a detected difference. Further tests may be performed, or a sequence of sub-operators may be proposed. These sub-operators may or may not be ordered, and there may or may not be alternative choices available. Whenever an unordered sequence is proposed, a task to order the sequence is posted. Similarly, whenever a collection of alternatives is proposed, a task to select an alternative is posted.

Ordering sub-problems

This task corresponds to the problem most often considered in early planning research. The body of its procedure consists of a number of rules for ordering sub-problems. It will suspend itself if none of the rules is applicable, in the hope that eventually information will become available that will aid in this decision. If, since there are no other tasks left to perfom, it is forced to make a choice when no other rules apply, it will do so in an arbitrary manner.

This task has much the same flavour as the techniques used by Sacerdoti in NOAH, but it also incorporates other techniques (see Case Study 1 below).

Choosing from amongst alternatives

Pretty much ignored by mainstream planning, this task becomes crucial when some domain information is inherently uncertain, and thus no plan can be guaranteed to work. In these cases, one must consider strategies involving the gathering of further information, and issues of recoverability from failed plans, rather than just taking the one most likely to succeed.

V CASE STUDIES

We will now present two case studies. The first of these is an English-language trace of one of the blocks world examples that is already running on the system. The second is an example from the domain of declarer play in bridge and whose implementation has not been completed. This second example is more difficult than the blocks world examples, and demonstrates some of the additional issues that a domain-independent planner must address in more complex domains.

Case Study 1: Blocks World

Consider the following problem from the blocks world: The initial state consists of the block configuration where C is on top of A and block B is sitting separately on the table. This is represented by the conjunction: (On C A) (Clear B) (Clear C). The goal is a state in which the conjunction (On A B) (On B C) is true. Given the operators Stack and Unstack the solution for this problem is the sequence of actions: (Unstack C A) (Stack B C) (Stack A B).

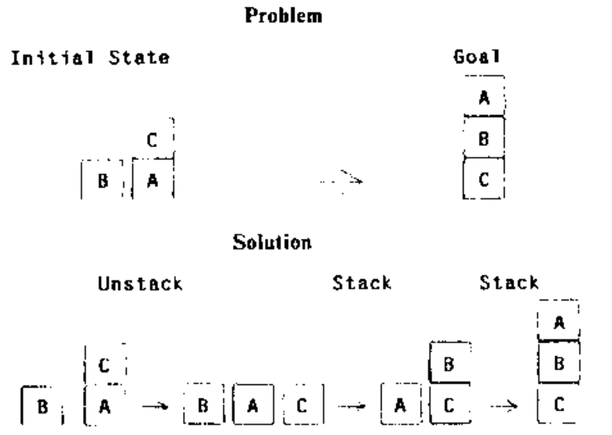

Figure 1. The problem, and SPAN's solution, discussed in Case Study 1.

We now present an English-language trace of SPAN solving this problem. This trace assumes that tasks will be chosen from the agenda in the worst possible order, thus demonstrating the system's ability to delay the execution of tasks until an appropriate time.

Initially, the only task on the agenda is to solve this problem. This task is selected and the body of the procedure associated with this task performs means-ends analysis on the problem. The two detected differences are that both (On A B) and (On B C) are initially false and need to be true in the goal state. Thus, the procedure creates a sub-problem for each difference detected, and adds to the agenda a corresponding task for each sub-problem to reduce the detected difference. Since there is more than one sub-problem, an additional task is posted to order the sub-problems. It is the execution of this third task (or its delay in execution) that determines the strategy taken in ordering conjunctive sub-goals. Both Sacerdoti's and Sussman's approach can be implemented this way. merely by changing the body of the procedure associated with this task. As implemented, this procedure in SPAN emphasizes Sacerdoti's approach.

Assume the task of ordering the sub-problems is now selected. It will examine the preconditions and goals of the two sub-problems and decide it has no basis for choosing which to do first. It thus defers this decision and is placed on the suspended list. SPAN will try again to order subproblems when there are no other tasks to perform.

Assume that the next task selected is the one associated with the sub-problem of making (On A B) true. The associated procedure uses the On function as an index to any operators that possibly could make (On A B) true. The only one suggested is the Stack operator, which has preconditions that A and B both be clear initially. This produces two sub-problems: first to get from the initial state to one in which (Clear A) (Clear B) is true and second to get from the state where (Clear A) (Clear B) is true to the goal where (On A B) is true. The second problem is solved by the primitive operator Stack. The first results in the posting of a task to solve the problem.

Assume this new task is the next one selected. The initial state of the sub-problem is unknown, since we do not know what actions will be taken from the initial state of the original problem before this sub-problem is tackled. This is a result of leaving unordered the sub-problems of making (On A B) and (On B C) true. The strategy adopted is to delay making a decision by suspending the task.

The procedure invoked to make (On B C) true opperates in the same way. It suggests the Stack operator with preconditions that B and C both be clear; which results in two sub-problems (one solved) and one task. The task (to make the preconditions true) is suspended because the initial state is not completely defined.

At this point, there are no more active tasks so the three suspended tasks are reexamined. If one of the tasks involving the preconditions is selected, it gets suspended, so we can assume that the ordering sub-problem is the next task selected and actually performed. The associated procedure examines the preconditions of making (On B C) true and notes that this conflicts with the goal of making (On A B) true. Thus it decides to order the (On B C) sub-problem before the (On A B) problem.

However, the procedure on ordering sub-problems does more. Since stacking B on C is a primitive operation, and thus its exact actions are known, and since the preconditions of (On A B) don't conflict with the goal of (On B C), these preconditions may be regressed back through the body of (Stack B on C). The heuristic used is to apply this technique of goal regression whenever possible. The reason is that it allows the earlier steps to reason with a more accurate picture of the complete goal. The result of this goal regression is to change the precondition for making (On B C) true to the conjunct: (Clear B) (Clear C) (Clear A). Now that an ordering for the sub-problems has been established, (On B C) is the first sub-problem and its initial state is known to be the initial state of the original problem. Thus when the task to solve the problem of making the conjunct (Clear B) (Clear C) (Clear A) true is selected, the initial state is properly defined. Means-ends analysis is applied and the difference detected is that (Clear A) is false initially, since (On C A) is true. A task is posted to make (Clear A) true.

Clearly this task must make (On C A) false in order to make (Clear A) true. (On C A), which is implemented as an object, knows the Unstack operator is a way of making the clause false. The task indexes this operator via the On clause. The Unstack operator is applied without anv hitches and leads to the final plan: (Unstack C A) (Stack B C) (Stack A B).

The process of finding this solution applied some of the insights developed by Sacerdoti along with the goal regression techniques developed by Waldinger (also independently by Warren). The reason this system was able to combine these methods was a flexible control structure which allowed procedural information specific to the type of task being performed to be invoked at the correct moment. The efficiency of using meta-level reasoning about planning depends on the tradeoff between the <u>benefits</u> of executing the best task and the <u>costs</u> of (a) reasoning at this level plus (b). partially executing tasks that are later suspended plus (c) deciding to suspend tasks. Although this example in the blocks world is too simple to justify the overhead, we believe that, in general, the tradeoff will favor meta-level planning.

Case Study 2: Bridge

The proposed architecture handles the previous example very smoothly, but it has been solved by other systems with much less overhead. The justification for the system proposed is the ease with which it can handle more complex problems. Consider the following declarer play problem from the game of bridge:

```
        North

S       9    5    2
H       8    7    2
D       A    K    5    3
C       K    Q    2

        South

S       A    Q    4    3
H       A    5
D       Q    6    2
C       A    7    3
```

The contract is 3 no trump by South. The opening lead by West is the king of hearts.

The declarer, South, needs to win at least nine of the thirteen available tricks to make his contract. He has eight immediate winners; three top spades, three top diamonds, the ace of hearts and the ace of clubs. There are two possible sources for the ninth trick. If the missing diamonds are split 3-3, then the first three rounds of diamonds will force them all to be played and the fourth diamond in the dummy (North) will be high. If the diamonds are not split evenly this plan fails.

Alternately, if East is holding the king of clubs, then South can lead a club from the dummy, winning the trick with the queen if East plays low and capturing the king with the ace (thus setting up the queen) if East plays the king. If West has the king then this plan fails.

Using techniques similar to those in the previous example (only the operators change), the planning process produces these two alternative plans. In addition, a task is posted to select between the alternatives. The associated procedure with this task first tries to order the alternatives so that if one branch fails the other can still be tried. In this particular case, the system determines that if the diamonds are tried first and don't split, then the club finesse can still be attempted. However, if the club finesse is tried first and fails the declarer may never get a chance to test the diamonds, since, if the hearts are not split 4-4, the defenders will win at least four hearts in addition to the king of clubs. Thus a hybrid plan, to test the diamonds and, if they don't split 3-3, to take the club finesse, is produced.

If this attempted hybridization had failed, the sytem would have looked at the relative probabilities of success for each alternative. This comparison could have been made using whatever method of uncertainty measurement was relevant for the particular domain (probably table look-up in this case).

VI <u>LIMITATIONS</u>

The system developed does have some inherent limitations. It assumes the world can be modeled as a series of discrete states and that operations on the world can be modeled as a transformation between these states. This discrete time assumption is basic to the model.

Furthermore, the strategies that have been developed have assumed that planning time is free. There is no consideration of limitations on planning resources. There has also been no consideration of the possibility that the world may change while the planning process is going on. These assumptions are not fundamental to the model, but I have not concentrated my efforts on handling these problems.

VII FUTURE WORK

As mentioned earlier, this system has been developed in the blocks world domain, and is currently being extended to handle the domain of declarer play for the game of bridge. Actual implementation in this domain may suggest better selection criteria for tasks, instead of the random selection now used. It may also confirm (or invalidate) the decision not to bother with an explicit strategy space.

More importantly, more difficult examples within this domain will push on the strategies for choosing amongst alternative plans. These strategies have not yet been implemented because, in the blocks world domain, there has been no need.

There will also be an opportunity to expand the types of tasks to include the collection of data and the execution of a partial plan. This interaction with the world will probably necessitate the establishment of priorities within the agenda, since execution tasks should be delayed until after the planning has been done, and some planning should be delayed until after new information that becomes available during the execution process is checked for relevance.

As a final point, research in the bridge domain will bring the techniques specific to competitive planning into the fold. In particular, the B* algorithm can be adapted to the analysis of interacting hierarchical plans. In some cases it is even possible by graphical analysis of hierarchical plans to determine the critical paths along which two competing plans interact, and thus to avoid most of the combinatorics involved in competitive planning.

Epilog

Danny Berlin died in early 1985. He had already analyzed the applicability of the B* algorithm and the technique of graphical analysis of interacting plans, but he did not get to complete the implementation of the second example. This paper was written by him. The camera-ready copy was prepared by Lucy M. Berlin and his advisor Bruce G. Buchanan.

References

[Berliner 78] Berliner, H.J.
The B* Search Algorithm: A best-first proof procedure.
Technical Report CMU-CS-78-112, Carnegie-Mellon University, 1978.

[Ernst 69] Ernst, G.W. and A. Newell.
GPS: A Case Study in Generality and Problem Solving.
Academic Press, New York, 1969.

[Friedland 79] Friedland, P.
Knowledge-based Hierarchical Planning in Molecular Genetics.
PhD thesis. Computer Science Department, Stanford University, September, 1979.
Report CS-79-760.

[Sacerdoti 74] Sacerdoti, E.D.
Planning in a Hierarchy of Abstraction Spaces.
Artificial Intelligence 5(2):115-135. Summer, 1974.

[Sacerdoti 77] Sacerdoti, E.D.
A Structure for Plans and Behavior.
Elsevier North-Holland, New York, 1977.

[Sacerdoti 79] Sacerdoti, E.D.
Problem Solving Tactics.
In Proc. Sixth International Joint Conference on Artificial Intelligence, pages 1077-1085. IJCAI, Tokyo, Japan, 1979.

[Sacerdoti 80] Sacerdoti, E.D.
Problem Solving Tactics.
AI Magazine 2(1):7-14. Winter, 1980.

[Stefik 80] Stefik, M.J.
Planning with Constraints.
Technical Report STAN-CS-80-784, Computer Science Department, Stanford University, January, 1980.

[Stefik 81] Stefik, M.J.
Planning and Meta -Planning.
Artificial Intelligence 16(2).T41-169, May, 1981.

[Sussman 75] Sussman, G.J.
A Computer Model of Skill Acquisition.
American Elsevier, New York, 1975.

[Waldinger 77] Waldinger, R.
Achieving Several Goals Simultaneously.
In Machine Intelligence 8, E.W. El cock and D. Michie eds.. pages 94-136. Ellis Horwood Limited, Chinchester, England, 1977.

[Warren 74] Warren, D.H.D.
WARPLAN: A system for generating plans.
Technical Report 76, Department of Computational Logic, University of Edinburgh, June, 1974.