Complexity Results for Blocks-World Planning[*]

Naresh Gupta[†]
Computer Science Department
University of Maryland
College Park, MD 20742
naresh@cs.umd.edu

Dana S. Nau[‡]
Computer Science Department
and Systems Research Center
University of Maryland
College Park, MD 20742
nau@cs.umd.edu

Abstract

Although blocks-world planning is well-known, its complexity has not previously been analyzed, and different planning researchers have expressed conflicting opinions about its difficulty. In this paper, we present the following results:

1. Finding optimal plans in a well-known formulation of the blocks-world planning domain is NP-hard, even if the goal state is completely specified.
2. Classical examples of deleted-condition interactions such as Sussman's anomaly and creative destruction are not difficult to handle in this domain, provided that the right planning algorithm is used. Instead, the NP-hardness of the problem results from difficulties in determining which of several different actions will best help to achieve multiple goals.

Introduction

Various versions of blocks-world planning have been widely investigated, primarily because they appear to capture several of the relevant difficulties posed to planning systems. The following version, which we call the Elementary Blocks World (EBW), is especially well-known (our description is based on those in (Kluzniak & Szapowicz, 1990) and (Nilsson, 1980)).

> The objects in the problem domain include a finite number of blocks, and a table large enough to hold all of them. Each block is on a single other object (either another block or the table). For each block x, either x is clear or else there is a unique block y sitting on x. There is one kind of action: move a single clear block, either from another block onto the table, or from an object onto another clear block. As a result of moving x from y onto z, x is sitting on z instead of y, y is clear (unless it is the table), and z is not clear (unless it is the table). A problem in this domain consists of a collection of ON and CLEAR predicates that completely specify the initial state, and another collection of ON and CLEAR predicates that provide necessary and sufficient conditions for a state to be a goal state. A solution to this problem is a plan (i.e., a sequence of "move" actions) capable of transforming the initial state into a state that satisfies the goal formula.

This problem domain, which we will call the Elementary Blocks World (EBW), has been particularly useful in investigations of goal and subgoal interactions in planning. The primary interactions that have been investigated have been deleted-condition interactions such as creative destruction and Sussman's anomaly (Charniak & McDermott, 1985; Kluzniak & Szapowicz, 1990; Nilsson, 1980; Sacerdoti, 1975; Sussman, 1975; Waldinger, 1977), in which a side-effect of establishing one goal or subgoal is to deny another goal or subgoal.

Despite the wide attention that has been given to planning problems in EBW, the complexity of planning in this domain has not been analyzed until now—and in addition, different people appear to have differing notions of what the complexity is. For example, in informal conversations with several prominent planning researchers, we posed the problem of how to find shortest-length plans in the special case where the goal state is completely specified. Some thought it obvious that the problem was easy, and others thought it obvious that the problem was difficult.

In this paper, we present the following results:

1. Given an instance of EBW and a positive integer L, the problem of answering whether there is a plan of length less than L is NP-complete, so the problem of finding a shortest-length plan is NP-hard. This is true even in the special case where the goal state is completely specified.
2. Surprisingly, the details of our proof of NP-completeness indicate that the complexity of the problem is not due to deleted-condition interactions, but instead results from the existence of what we call

[*]This work was supported in part by an NSF Presidential Young Investigator Award to Dana Nau, NSF Grant NSFD CDR-85-00108 to the University of Maryland Systems Research Center, and NSF Grant IRI-8907890.

[†]Also with LNK Corporation, College Park, MD.

[‡]Also affiliated with the University of Maryland Institute for Advanced Computer Studies.

"deadlock" situations, in which some critical step is needed to facilitate the achievement of several different goals. Some steps are capable of resolving more than one deadlock at once—and what is difficult is to find a set of critical steps that resolves all deadlocks and is as small as possible. All remaining steps in the plan can be determined easily, regardless of whether or not they would correspond to deleted-condition interactions in the traditional sense.

Definitions and Notation

In this paper, we use the usual definitions of the ON and CLEAR predicates and the MOVE operator. As usual, a *plan* is a totally ordered set of actions, each of which can be applied to the state that results from the actions that precede it. If P is a plan, then length(P) is the number of actions in P.

The EBW optimization problem is as follows:

Given an EBW problem B, find a plan for B of shortest possible length.

Note that in B, every state s consists of a set of stacks of blocks. We say that a block b in s is *in its final position* if there is a state g satisfying the goal formula such that all blocks below b in s are also below b in g, in exactly the same order.

We say that the set of blocks $\{b_1, b_2, \ldots, b_p\}$ in s is *deadlocked* if there is a set of blocks $\{d_1, d_2, \ldots, d_p\}$ such that the following two conditions hold:

1. in the state s, b_1 is above d_1, b_2 is above d_2, ..., and b_p is above d_p;
2. in the goal state, b_1 is above d_2, b_2 is above d_3, ..., b_p is above d_1.

For example, in Fig. 1, the initial state contains two deadlocked sets of blocks:

1. a is above c and d is above e in the initial state, and a is above e and d is above c in the goal state, so $\{a, d\}$ is deadlocked in the initial state;
2. a is above b in both the initial state and the goal state, so $\{a\}$ is deadlocked in the initial state.

Suppose S is a deadlocked set and A is an applicable action. Then we say that A *resolves* the deadlock if S is no longer deadlocked after A is performed. If one of the blocks in S is clear, then moving it to the table will always resolve the deadlock—and it may resolve more than one deadlock simultaneously. For example, in Fig. 1, moving a to the table will resolve both the deadlocked sets $\{a, d\}$ and $\{a\}$.

Recall that in EBW, the initial state must be completely specified but the goal state need not be completely specified. Instead, it is possible for more than one state to satisfy the goal formula. We define the Primitive Blocks World (PBW) to be the special case of EBW in which the goal state is completely specified.

For use in proving NP-completeness results about PBW, we follow the standard procedure for converting optimization problems into yes/no decision problems. The *Yes/No PBW* (YPBW) decision problem is as follows:

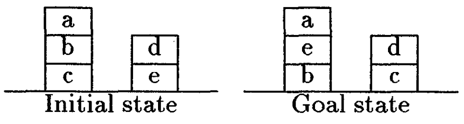

Figure 1: A blocks-world problem with two deadlocked sets of blocks: $\{a, d\}$ and $\{a\}$.

Given a PBW problem B and an integer $L > 0$, is there a plan for B of length less than L?

Algorithms

Let B be a PBW problem. We make the following observations:

1. Whether or not each block b is in its final position can be computed in low-order polynomial time, by examining the stack of blocks beneath b to see if the adjacent blocks in this stack correspond to ON predicates in the goal formula.
2. Any block already in its final position need not be moved.
3. Any block not already in its final position must be moved at least once. However, it does not need to be moved more than twice: we can move it to the table until its final position is clear, and then move it to its final position.
4. If b is moved more than once in some plan P for B, then there is a plan P' for B in which b is moved twice (once from its initial position to the table, and later from the table to its final position), and all other actions are the same as in P.
5. From the above observations, it follows that if there are n blocks in B, then the length of a shortest-length plan for B is between 0 and $2n$.

The above observations make it clear that for any PBW problem B, it is easy to generate a plan P whose length is no more than twice the shortest possible length. Simply move to the table all blocks that are not in their final positions, and then move these blocks one by one into their final positions. This plan can be generated in low-order polynomial time.

Finding a shortest-length plan is more difficult—but if we are allowed to use nondeterminism, it can be done in low-order polynomial time by the nondeterministic algorithm Solve-PBW shown below. In this algorithm, RESOLVE is a nondeterministic command that resolves a deadlock among the top blocks of stacks in the current state, by moving one of the blocks from top of a stack to the table.

```
Algorithm Solve-PBW
Input: a PBW problem $B$.
```

```
Output: a plan $P$ for $B$.
Step 1: Current state $\leftarrow$ initial state.
Step 2: If the current state is the same as the goal
    state, then EXIT.
Step 3: If there is a block $b$ that can be moved to
    its final position, move it and go to Step 2.
Step 4: If we reach this step, then the set of all
    blocks that are at the tops of their stacks and
    are not in their final positions form one or more
    deadlocked sets. Call RESOLVE to resolve one
    of these deadlocks, and go to Step 2.
```

Theorem 1 *Let B be any PBW problem with p blocks, of which q blocks in the initial state are already in their final positions. Then the length of a shortest-length plan for B is $p-q+r$, where r is the minimum number of times RESOLVE is called in any of the execution traces of nondeterministic procedure Solve-PBW(B).*

Proof. Let P be any plan for B. From Observation 4 it follows that there is a plan P' with length$(P') \leq$ length(P) such that for every block b that is moved more than once, b is moved exactly twice: once from its initial position to the table and once from the table to its final position. Let s be the state just before b is moved for the first time. For every block c in s that can be moved directly to its final state, moving c to its final position before moving b to the table cannot possibly interfere with any subsequent moves. Thus, there is a plan P'' with length$(P'') \leq$ length(P) having the property that whenever a block is moved to the table, the current state contains no block that can be moved directly to its final state. Solve-PBW generates every plan having this property, so therefore it generates a plan of shortest possible length. But the shortest-length plan generated by Solve-PBW has length $p-q+r$, so this must be the shortest possible length of any plan for B. ■

Complexity Results

In this section, we show that YPBW is NP-complete—and that therefore, the PBW and EBW optimization problems are NP-hard.

Lemma 1 *YPBW belongs to NP.*

Proof. We give a nondeterministic polynomial time algorithm to solve YPBW:

```
Algorithm Solve-YPBW
Input: a YPBW problem $(B, L)$.
Output: True if there is a plan for $B$ of length
    $< L$, and False otherwise.
Step 1: If Solve-PBW($B$) finds a plan $P$ such that
    length$(P) < L$, return True. Else return False.
```

Solve-YPBW returns *True* if and only if (B, C) belongs to YPBW. Since Solve-PBW takes polynomial time, Solve-YPBW also takes polynomial time. ■

To show that YPBW is NP-hard, we need to show that an NP-complete problem reduces to YPBW. For this purpose we use the *Feedback Arc Set* (FAS) decision problem, which can be stated as follows:

Given a digraph $G = (V, E)$ and a positive integer k, is there a set of edges S of cardinality less than k, such that the digraph $G' = (V, E-S)$ is acyclic?

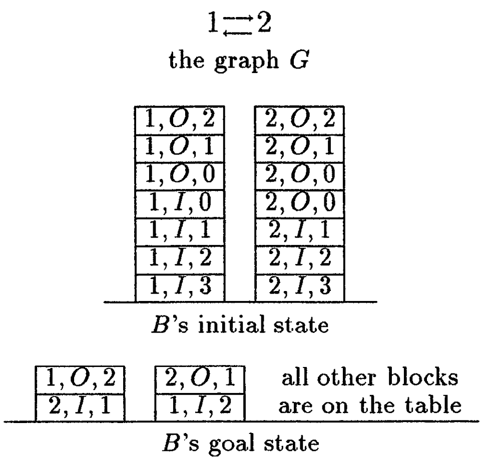

Figure 2: A graph G, and the PBW problem B returned by $M(G, k)$.

This problem is known to be NP-complete ((Garey & Johnson, 1979), p. 192).

If $G = (V, E)$ is a digraph, then without loss of generality we may assume that V is the set of integers $\{1, 2, \ldots, n\}$ for some n. If (G, k) is a FAS problem, then we define $M(G, k)$ to be the YPBW problem (B, L), where $L = (2n+2)n + k$, and B is the PBW problem defined below:

- For each $v \in V$, B's initial state contains a stack of $2n+3$ blocks, named (from the bottom of the stack to the top) $[v, I, n+1]$, $[v, I, n]$, ..., $[v, I, 0]$, $[v, O, 0]$, ..., $[v, O, n]$ (see Fig. 2). Thus, B's initial state consists of n stacks of $2n+3$ blocks each, for a total of $2n^2 + 3n$ blocks.
- B's goal state has $[x, O, y]$ stacked on $[y, I, x]$ for every edge (x, y) in E, and all other blocks sitting on the table. Thus, B's goal state contains $|E|$ stacks of 2 blocks each, and $2n^2 + 3n - |E|$ blocks sitting on the table by themselves.

$M(G, k)$ can easily be computed in polynomial time.

For the rest of this section, we let (G, k) be any FAS problem, and $(B, L) = M(G, k)$.

Lemma 2 *Each cycle in G corresponds to a unique deadlocked set in B.*

Proof. Suppose G contains a cycle $(v_1, v_2, \ldots, v_p, v_1)$. Then the edges (v_1, v_2), (v_2, v_3), ..., (v_p, v_1) are in E, so in B's goal state, we have $[v_1, O, v_2]$ on $[v_2, I, v_1]$, $[v_2, O, v_3]$ on $[v_3, I, v_2]$, ..., and $[v_p, O, v_1]$ on $[v_1, I, v_p]$. But in B's initial state, we have

$[v_1, O, v_2]$ above $[v_1, I, v_p]$, $[v_2, O, v_3]$ above $[v_2, I, v_1]$, ..., and $[v_p, O, v_1]$ above $[v_p, I, v_{p-1}]$. Thus the set $\{[v_1, O, v_2], [v_2, O, v_3], \ldots, [v_p, O, v_1]\}$ is deadlocked.

Conversely, suppose a set of blocks D is deadlocked. Then each block $b \in D$ must be on some other block c in the goal state. But from the definition of B, this means there are v, w such that $b = [v, O, w]$ and $c = [w, O, v]$. Thus, there are $z_1, z_2, \ldots, z_p$ such that $D = \{[z_1, O, z_2], [z_2, O, z_3], \ldots, [z_p, O, z_1]\}$ and B's goal state contains the following stacks: $[z_1, O, z_2]$ on $[z_2, I, z_1]$, $[z_2, O, z_3]$ on $[z_3, I, z_2]$, ..., $[z_p, O, z_1]$ on $[z_1, I, z_p]$. From the definition of B, this means that $(z_1, z_2, \ldots, z_p, z_1)$ is a cycle in G.

Thus each cycle in G corresponds to a unique deadlocked set of blocks in B. ■

An example of Lemma 2 appears in Fig. 2, in which the cycle $(1, 2, 1)$ in G corresponds to the deadlocked set of blocks $\{[1, O, 2], [2, O, 1]\}$.

Lemma 3 *B has a plan of length less than L iff G has a feedback arc set of size less than k.*

Proof. $(\rightarrow)$: Suppose B has a plan P for which length$(P) < L$, and let S be the set of all blocks that are moved more than once in P. From Observation 4, we may assume that each block in S is moved twice: once to the table and once to its final position. Since $2n^2 + 2n$ blocks in B's initial state are not in their final positions, they must be moved at least once. Thus $|S| \leq L - 2n^2 + 2n = k - 1$. For each deadlocked set D, P resolves the deadlock by moving some block $b \in D$ to the table. From the definition of deadlock, b's final position must be on top of some other block, so $b \in S$. From the proof of Lemma 2, $b = [v, O, w]$ for some edge (v, w) in G. Thus, S contains blocks $[v_1, O, w_1], \ldots, [v_j, O, w_j]$ such that every deadlocked set D contains at least one of these blocks. From the proof of Lemma 2, it follows that every cycle in G contains one of the edges (v_1, w_1), ..., (v_j, w_j), so G has a feedback arc set of size $j \leq |S| \leq k - 1$.

$(\leftarrow)$: Suppose G has a feedback arc set $S = \{(v_1, w_1), \ldots, (v_p, w_p)\}$ such that $p < k$, and suppose we invoke Solve-PBW(B). The initial state contains $2n^2 + 2n$ blocks that are not in their final positions, so Step 3 of Solve-PBW will be executed $2n^2 + 2n$ times. Each time Solve-PBW enters Step 4, the set of all blocks b that are at the top of their stacks and are not in their final positions form one or more deadlocked sets. From Lemma 2, each such deadlocked set D corresponds to a cycle in G, so at least one block $[v_i, O, w_i] \in D$ corresponds to an edge $(v_i, w_i) \in S$. But moving $[v_i, O, w_i]$ to the table will resolve the deadlock. Thus, there is an execution trace for Solve-PBW(B) in which all deadlocks are resolved by moving to the table blocks in the set $\{[v_1, O, w_1], \ldots, [v_p, O, w_p]\}$, whence Step 4 is executed at most p times. Thus, one of the execution traces for Solve-PBW finds a plan P of length length$(P) \leq 2n^2 + 2n + p < 2n^2 + 2n + k = L$. ■

Theorem 2 *YPBW is NP-complete.*

Proof. Lemma 3 shows that M reduces FAS to YPBW. Since M runs in polynomial time, this means that YPBW is NP-hard. But Lemma 1 shows that YPBW is in NP. Thus, YPBW is NP-complete. ■

Theorem 3 *The PBW and EBW optimization problems are NP-hard.*

Proof. If we can find a shortest-length plan, then we can immediately tell whether there is a plan of length less than L. Thus from Theorem 2, the YPBW optimization problem is NP-hard. Since PBW is a special case of EBW, the EBW optimization problem is also NP-hard. ■

Discussion

The nondeterministic algorithm Solve-PBW finds shortest-length plans, and it needs to make a nondeterministic choice only when a deadlock occurs. If there were no deadlocks, then Solve-PBW would be a deterministic algorithm operating in low-order polynomial time. Thus, it is the deadlocks that are responsible for the NP-completeness of YPBW—so we need to examine them more carefully.

It is important to note that deadlocks are not the same as deleted-condition interactions. Rather, an action a that resolves a deadlock is useful because it facilitates the achievement of several different goals at once. Some actions are capable of resolving more than one deadlock—and in finding a shortest-length plan, the critical problem is to find a smallest possible set of actions capable of resolving all existing deadlocks. All actions other than these resolving actions can be determined quite easily by Solve-PBW.

The primary reason why deleted-condition interactions do not cause problems in Solve-PBW is that it does not consider the ON predicates in isolation, but instead considers the partial order induced by them. For example, if we want to achieve ON(x, y), we should make sure that y is in its final position before we try to move x to y. By "final position", we do not mean merely whether y is on the same block it will be on in the goal state, but whether the entire stack of blocks below y is the same as it will be in the goal state.

It is straightforward to generalize Solve-PBW to solve problems in EBW, and we intend to include a proof of this in (Gupta & Nau, 1991). In EBW there is not necessarily a unique goal state—but by doing a topological sort on the ON predicates in the goal formula, we can easily deduce whether any block that is currently below y in the current state will need to be elsewhere in the goal state.

Conclusion

In this paper, we have discussed a well-known class of planning problems which we call the Elementary Blocks World (EBW). We have shown that the problem of finding shortest-length plans in EBW is NP-hard, even if the goal state is completely specified. This result is interesting for two reasons. First is that in the case where the goal state is completely specified, different planning researchers have had conflicting intuitions about the difficulty of finding shortest-length plans—and this result answers the question. Second, the nature of the proof says something unexpected about why blocks-world planning is difficult.

One of the primary uses of the blocks world in the planning literature has been to provide examples of deleted-condition interactions such as creative destruction and Sussman's anomaly (Charniak & McDermott, 1985; Kluzniak & Szapowicz, 1990; Nilsson, 1980; Sacerdoti, 1975; Sussman, 1975; Waldinger, 1977), in which the plan for achieving one goal or subgoal interferes with another goal or subgoal. However, as we pointed out in Section , such interactions present no problem if an appropriate planning algorithm is used.

The complexity of planning in EBW is due instead to "deadlock" situations, in which some critical action must be chosen in order to help in achieving the remaining goals. In choosing which action to use to resolve a deadlock, some actions are better than others, because they will resolve more than one deadlock—but if we use the hill-climbing approach of always choosing the action that resolves the most deadlocks, this will not always result in a shortest-length plan. Instead, we must find a minimum-cardinality set of actions that resolves all deadlocks. Apparently this problem is what causes planning in EBW to be NP-hard—for if we can solve this problem, it is easy to plan the other actions.

Acknowledgements

We wish to thank James Hendler for his many helpful comments and discussions.

References

Allen, J.; Hendler, J.; and Tate, A., editors 1990. *Readings in Planning.* Morgan-Kaufmann, San Mateo, CA.

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

Charniak, E. and McDermott, D. 1985. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA.

Garey, Michael R. and Johnson, D. S. 1979. *Computers and Intractability A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company Publishing.

Gupta, N. and Nau, D. S. 1991. On the complexity of blocks world planning. Forthcoming.

Kluzniak, and Szapowicz, 1990. extract from apic studies in data processing no. 24. In Allen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 140–153.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence.* Tioga, Palo Alto.

Sacerdoti, E. D. 1990. The nonlinear nature of plans. In Allen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 206–214. Originally appeared in *Proc. IJCAI-75.*

Sussman, G. J. 1975. *A Computational Model of Skill Acquisition.* American Elsevier, New York.

Waldinger, R. 1990. Achieving several goals simultaneously. In Allen, J.; Hendler, J.; and Tate, A., editors 1990, *Readings in Planning.* Morgan Kaufman. 118. Originally appeared in *Machine Intelligence 8*, 1977.