Artificial Intelligence, 56(2-3):223–254, August 1992.

On the Complexity of 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

In this paper, we show that in the best-known version of the blocks world (and several related versions), planning is difficult, in the sense that finding an optimal plan is NP-hard. However, the NP-hardness is not due to deleted-condition interactions, but instead due to a situation which we call a deadlock. For problems that do not contain deadlocks, there is a simple hill-climbing strategy that can easily find an optimal plan, regardless of whether or not the problem contains any deleted-condition interactions.

The above result is rather surprising, since one of the primary roles 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. However, we can explain why deadlocks are hard to handle in terms of a domain-independent goal interaction which we call an enabling-condition interaction, in which an action invoked to achieve one goal has a side-effect of making it easier to achieve other goals. If different actions have different useful side-effects, then it can be difficult to determine which set of actions will produce the best plan.

Please address all correspondence to the second author.

*This work was supported in part by NSF Grants NSFD CDR-88-03012 and IRI-8907890.

†Also with LNK Corporation, College Park, MD.
‡Also affiliated with the University of Maryland Institute for Advanced Computer Studies.

Initial state: $I = \{\text{clear}(a),$ $\text{on}(a, b), \text{on}(b, \mathcal{T}), \text{clear}(c), \text{on}(c, \mathcal{T})\}$

Goal formula: $G = \{\text{on}(b, c)\}$

Figure 1: A simple EBW problem.

1 Introduction

Blocks-world planning has been widely investigated by planning researchers, primarily because it appears to capture several of the relevant difficulties posed to planning systems. It has been especially useful in investigations of goal and subgoal interactions in planning—particularly deleted-condition interactions such as creative destruction and Sussman's anomaly [4, 15, 16, 18, 19, 20, 21], in which a side-effect of establishing one goal or subgoal is to deny another goal or subgoal.

The following version of the blocks world, which we call the Elementary Blocks World (EBW), is especially well known. Our description is based on those in [15] and [21].

> The objects in the problem domain include a finite number of cubical 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 b, either b is clear or else there is a unique block a sitting on b. 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 b from c onto d, b is sitting on d instead of c, c is clear (unless it is the table), and d is not clear (unless it is the table).
>
> A problem in this domain is specified by giving two sets of ground atoms,[1] one specifying an initial state of the world, and the other specifying necessary and sufficient conditions for a state to be a goal state (for example, see Figure 1). A solution to this problem is a plan (i.e., a sequence of "move" actions) capable of transforming the initial state into a state satisfying the goal conditions.

In this paper, we present the following results about EBW and related problem domains:

1. *Planning in EBW.* In EBW, finding a non-optimal plan is quite easy, and finding an optimal plan is NP-hard (but no worse). Surprisingly, the NP-hardness is not due to deleted-condition interactions, but to a different kind of goal interaction which we call a "deadlock". For EBW problems that do not contain deadlocks, there is a simple hill-climbing strategy that is guaranteed to find an optimal plan in time $O(n^3)$ where n is the problem size, regardless of whether or not the problem contains deleted-condition interactions. Classical examples of deleted-condition interactions, such as Sussman's

[1]Since EBW contains no function symbols, for our purposes a ground atom is a predicate whose arguments are all constants denoting blocks or the table.

anomaly and creative destruction, do not contain deadlocks—and thus they are easily handled by this planner.

2. *Completely specified goal states.* Planning in EBW has been thought to be simpler in the special case where the goal state is completely specified, but there has been disagreement on how much simpler. For example, in informal conversations with several prominent planning researchers, we posed the problem of how to find shortest-length plans in this special case. Some thought it obvious that the problem was easy, and others thought it obvious that the problem was difficult.

   It turns out that this special case is basically equivalent to the general case. There is an algorithm which, given any EBW problem, will produce in time $O(n^3)$ a completely specified goal state such that any optimal plan for reaching this goal state is also an optimal plan for the original problem. Thus, the results we stated above for EBW still hold even if the goal state is completely specified.

3. *Other versions of the blocks world.* Other versions of the blocks world have also appeared in the AI literature. For example, Winograd's original version of the blocks world contained blocks of different sizes and colors, and also contained pyramids [23].

   If we generalize EBW to contain objects of varying sizes, including blocks, pyramids, and frustums (and prohibit objects from being placed on smaller objects), then all of the above results still hold. If we limit the total number of blocks that may sit on the table, then different planning algorithms are required, but finding a non-optimal plan is still easy, and finding an optimal plan is still NP-hard but no worse. If in addition to limiting the table size we allow different blocks to have different sizes, then it is no longer possible to find optimal plans nondeterministically in polynomial time, because there are some planning problems in which the shortest plan has exponential length.

4. *Enabling-condition interactions.* The difficulty of handling deadlocks can be described in terms of a domain-independent goal interaction which we call an "enabling-condition interaction," in which an action invoked to achieve one goal has a side-effect of making it easier to achieve other goals. Enabling-condition interactions can also be used to explain some of the difficulties that occur in planning problems investigated by other researchers [5, 13]. In general, if different actions have different useful side-effects, then it can be difficult to determine which set of actions will produce the best plan.

This paper is organized as follows. Section 2 contains basic definitions. Section 3 describes some planning algorithms for EBW, and shows that planning in EBW is NP-hard. Section 4 explains why the NP-hardness is due to deadlocks rather than deleted-condition interactions. Section 5 describes what happens if we generalize EBW to allow limited table size and/or objects of varying sizes. Section 6 summarizes our results, and Section 7 discusses related work. Section 8 discusses the significance of our results, describes enabling-condition interactions, and suggests topics for future research. The proofs are contained in the appendices.

Table 1: Initial block positions in the EBW problem of Figure 1.

Block	Block's position in I	Position is a maximal stack	Position is consistent with the goal $G = \{\text{on}(b, c)\}$
a	$\{\text{on}(a, b), \text{on}(b, \mathcal{T})\}$	Yes	No: $\text{on}(b, \mathcal{T})$ contradicts $\text{on}(b, c)$
b	$\{\text{on}(b, \mathcal{T})\}$	No	No: $\text{on}(b, \mathcal{T})$ contradicts $\text{on}(b, c)$
c	$\{\text{on}(c, \mathcal{T})\}$	Yes	Yes: $\{\text{on}(b, c), \text{on}(c, \mathcal{T})\}$ is consistent

2 Basic Definitions

2.1 Formulas, States, Stacks, and Positions

An *atom* is a term of the form "on(x, y)" (meaning that x is on y) or "clear(x)" (meaning that x is clear), where x and y are either constants (i.e., specific blocks or the table) or variables. If x and y are constants, then the atom is a *ground atom*. The constant $\mathcal{T}$ denotes the table.

A *formula* is any set F of ground atoms. A formula F is *consistent* if there is at least one configuration of blocks that satisfies the meanings of the atoms in F. A formula F is *consistent with* a formula G if $F \cup G$ is consistent.

A formula F is a *state* if it specifies the exact configuration of some set of blocks, (i.e., what blocks are clear, what blocks are on the table, and what blocks are on what other blocks). An immediate consequence of this definition is that every state is consistent.

An *EBW problem* is an ordered pair $B = (I, G)$, where I is a state called the *initial state*, and G is a formula called the *goal formula*. B is *solvable* if there exists at least one plan for B.

Let F be any consistent formula. A *stack* in F is a formula

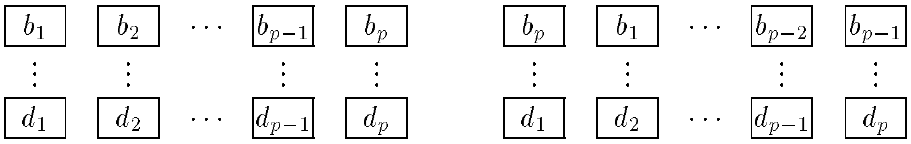
$$E = \{\text{on}(b_1, b_2), \text{on}(b_2, b_3), \ldots, \text{on}(b_{p-1}, b_p)\} \subseteq F$$

such that each b_i is a block except for b_p, which may be either a block or the table. Informally, we will write E as "b_1 on b_2 on b_3 on ... on b_p". The *top* and *bottom* of E are b_1 and b_p, respectively. E is a *maximal stack* if it is not a subset of any other stack in F. If F is a state and b is a block in F, then b's *position* in F is the largest stack in F whose top is b, i.e., the stack in F whose top is b and whose bottom is the table. b's position is a maximal stack if and only if b is clear.

From the above definitions, it follows that in any EBW problem, the position of a block a is consistent with the goal formula G only if the positions of all blocks underneath a are also consistent with G. For example, consider the EBW problem shown in Figure 1. As shown in Table 1, since b's position in I is inconsistent with G, a's position in I is also inconsistent with G.

2.2 Actions, Plans, and Deadlocks

We use move(x, y, z) to denote the action of moving x from y to z. A *plan* is a finite sequence of such actions. If P is a plan, then $|P|$ is the number of actions in P, and $P(S)$ is the state



The current state S The goal G

Figure 2: An illustration of the definition of deadlock, in the case where $p > 1$.

produced by starting at S and applying the actions in P one at a time (if not all of them are applicable, then $P(S)$ is not defined). A plan *for* an EBW problem $B = (I, G)$ is a plan P such that $P(I)$ is consistent with G. It is an *optimal* plan for B if every plan Q for B has $|Q| \geq |P|$.

The set of blocks $\{b_1, b_2, \ldots, b_p\}$ is *deadlocked* in the state S if there is a set of blocks $\{d_1, d_2, \ldots, d_p\}$ such that the following three conditions hold (see Figure 2):

1. In S, b_i is above d_i for $i = 1, 2, \ldots, p$.

2. In G, b_i is above d_{i+1} for $i = 1, 2, \ldots, p-1$, and b_p is above d_1.

3. In S, none of $b_1, b_2, \ldots, b_p$ are in their final positions (if $p > 1$, then the other two conditions entail this condition).

For example, in Figure 3, in the initial state I there are two deadlocked sets of blocks:

1. In I, a is above c and d is above e. In G, a is above e and d is above c. Thus $\{a, d\}$ is deadlocked in I.

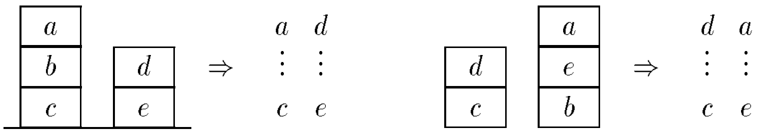
2. a is above b in both I and G, and a is not in its final position in I. Thus $\{a\}$ is deadlocked in I.

Suppose some set of blocks D is deadlocked in the state S. If A is an action applicable to S, then A *resolves* D if D is not deadlocked in the state produced by applying A to S. If any of the blocks in D is clear in S, then moving it to the table will always resolve the deadlock—and it may resolve more than one deadlock simultaneously. For example, in Figure 3, the action move$(a, b, \mathcal{T})$ will resolve both the deadlocked sets $\{a, d\}$ and $\{a\}$.

3 Planning in EBW

3.1 Planning Algorithms

From the meanings of the "on" and "clear" atoms in EBW, it follows that a formula F is consistent if and only if the following conditions hold for every block b mentioned in F: b is not above itself, the table is not on b, b is on at most one object, at most one object is on b, and nothing is on b if b is clear. One can verify whether or not these conditions are satisfied in time $O(n)$ as follows, where n is the number of atoms in F. Consider the



a is in d's way and d is in a's way, so $\{a, d\}$ is deadlocked.

a is in its own way, so $\{a\}$ is deadlocked.

Initial state: $I = \{\text{clear}(a),$
$\text{on}(a, b), \text{on}(b, c), \text{on}(c, \mathcal{T}),$
$\text{clear}(d), \text{on}(d, e), \text{on}(e, \mathcal{T})\}$

Goal formula:
$G = \{\text{on}(a, e),$
$\text{on}(e, b), \text{on}(d, c)\}$

Figure 3: A problem in which two sets of blocks are deadlocked: $\{a, d\}$ and $\{a\}$.

graph $H(F)$ whose nodes are the blocks, and having an arc from block b to block c if and only if $\text{on}(b, c) \in F$. Such a graph is called a Hasse diagram (e.g., see [17]), and it can be constructed in time $O(n)$ using a modification of a topological sorting algorithm such as the ones given in [6, 14]. F is consistent if and only if F contains no atoms of the form "$\text{on}(\mathcal{T}, x)$", and $H(F)$ consists of one or more disjoint acyclic paths, with each clear block at the beginning of a different path.

Let $B = (I, G)$ be any EBW problem. Let m and n, respectively, be the number of blocks and atoms in $I \cup G$. Since every atom contains at least one block, $m < n$. Since I is a state it is consistent, so it contains at most two atoms for each block. If G is consistent, then G will also contain at most two atoms for each block, so $n \leq 4m$ and thus $O(m) = O(n)$.

One can check whether or not B is solvable in time $O(n \log n)$, by checking whether or not G is consistent, and whether or not G mentions any blocks not mentioned in I. If G is inconsistent or mentions a block not mentioned in I, then clearly B is not solvable. But if G is consistent and only mentions blocks mentioned in I, then each of the paths in $H(G)$ represents one of the maximal stacks in G. One can produce a plan for B by moving all blocks to the table, and then using $H(G)$ to guide us in building these maximal stacks from the bottom up. The length of this plan is at most $2m$, and it takes time $O(n)$ to produce it.

In time $O(n^3)$ one can find a plan of length no more than twice the length of the optimal plan. To see this, consider the algorithm Solve-EBW shown below. This is basically a simple hill-climbing algorithm: any time a block can be moved directly to a position consistent with the goal condition, it does so.

Algorithm Solve-EBW(I, G):
Step 1: If G contains any blocks not in I, then (I, G) is not solvable, so exit

with failure.

Step 2: Construct the graph $H(G)$, and use it to check whether or not G is consistent. If G is inconsistent, then (I, G) is not solvable, so exit with failure.

Step 3: $S \leftarrow I$.

Step 4: If S is consistent with G, then exit with success.

Step 5: If S contains clear blocks b and c such that b's position is not consistent with G, G contains on(b, c), and c's position is consistent with G, then move b to c and go to Step 4.

Step 6: If S contains a clear block b such that b's position is not consistent with G and there is no c such that G contains on(b, c), then the position on$(b, \mathcal{T})$ is consistent with G, so move b to the table and go to Step 4.

Step 7: At this point, every clear block whose position is not consistent with G is in a deadlocked set. Arbitrarily move one of these blocks to the table, and then go to Step 4.

In Appendix A we prove that the plan Q produced by Solve-EBW satisfies the property $|Q| \leq 2(m-q)$, where m is the total number of blocks in B, and q is the number of blocks whose positions in I are consistent with G. Since every plan for B must have length at least $m-q$, this means that the length of Q is no more than twice the optimal length. Each of the steps in Solve-EBW takes time at most $O(n^2)$ to execute. Since Solve-EBW exits after $O(m) = O(n)$ steps, this means it runs in time $O(n^3)$.

All of the moves made by Solve-EBW preserve plan optimality except for the moves made in Step 7. The algorithm Solve-EBW-Optimally shown below is identical to Solve-EBW, except that Step 7 is modified to make all possible choices nondeterministically. Thus, as proved in Appendix A, Solve-EBW-Optimally is guaranteed to find an optimal plan—and the length of this plan is $m - q + r$, where r is the minimum number of times that Step 7 is executed in any of the execution traces of Solve-EBW-Optimally.

Algorithm Solve-EBW-Optimally(I, G):

Step 1: If G contains any blocks not in I, then (I, G) is not solvable, so exit with failure.

Step 2: Construct the graph $H(G)$, and use it to check whether or not G is consistent. If G is inconsistent, then (I, G) is not solvable, so exit with failure.

Step 3: $S \leftarrow I$.

Step 4: If S is consistent with G, then exit with success.

Step 5: If S contains clear blocks b and c such that b's position is not consistent with G, G contains on(b, c), and c's position is consistent with G, then move b to c and go to Step 4.

Step 6: If S contains a clear block b such that b's position is not consistent with G and there is no c such that G contains on(b, c), then the position on$(b, \mathcal{T})$ is consistent with G, so move b to the table and go to Step 4.

Step 7: At this point, every clear block whose position is not consistent with G is in a deadlocked set. Nondeterministically move one of these blocks to the table, and then go to Step 4.

3.2 NP-Completeness

For use in proving NP-completeness results about EBW, we follow the standard procedure for converting optimization problems into yes/no decision problems. We define *EBW PLAN* to be the following decision problem:

> Given an EBW problem (I, G) and an integer $L > 0$, is there a plan for this problem of length L or less?

To show that EBW PLAN is NP-hard, we need to show that an NP-complete problem reduces to EBW PLAN. For this purpose we use the *FEEDBACK ARC SET* problem, which can be stated as follows:

> Given a digraph (V, E) and a positive integer k, is there a set of edges F such that $|F| \leq k$ and the digraph $(V, E - F)$ is acyclic?

This problem is known to be NP-complete ([10], p. 192).

In Appendix B, we show that EBW PLAN is NP-complete, by showing that FEEDBACK ARC SET reduces to EBW PLAN and that EBW PLAN can be solved nondeterministically in polynomial time using Solve-EBW-Optimally. From this, it follows that finding optimal plans in EBW is NP-hard. The fact that Solve-EBW-Optimally will find optimal plans nondeterministically in polynomial time suggests that finding optimal plans in EBW is no worse than NP-hard—and in Appendix B we prove that this is true.

3.3 Completely Specified Goal States

Primitive Blocks World (PBW) is the special case of EBW in which the goal formula specifies a single state. *PBW PLAN* is the following decision problem:

> Given a PBW problem (I, G) and an integer $L > 0$, is there a plan for this problem of length L or less?

Although PBW has been thought to be simpler problem domain than EBW, it turns out that planning in PBW is basically equivalent to planning in EBW. In particular, given any solvable EBW problem, one can easily add additional conditions to the goal formula G to produce a completely specified goal state G', in such a way that any optimal plan for the modified problem is also an optimal plan for the original problem.

Before describing how to do this in the general case, we first illustrate the idea using Sussman's anomaly,[2] which is shown in Figure 4. The desired goal state G' must contain $\text{on}(a, b)$ and $\text{on}(b, c)$, and it cannot mention any other blocks since no other blocks are mentioned in I. In G', c is below the other two blocks, so c must be on the table. Furthermore, a is above the other two blocks, so a must be clear. Thus, G' must be the state $\{\text{on}(a, b), \text{on}(b, c), \text{on}(c, \mathcal{T}), \text{clear}(a)\}$.

For the general case, let $B = (I, G)$ be any solvable EBW problem, and let F be the formula consisting of the following "on" atoms:

- every "on" atom in G;

[2]This EBW problem was proposed by Allen Brown [20, p. 127], and popularized by Sussman [19].

Initial state: $I = \{\text{clear}(c), \text{on}(c, a), \text{on}(a, \mathcal{T}), \text{clear}(b), \text{on}(b, \mathcal{T})\}$

Goal formula: $G = \{\text{on}(a, b), \text{on}(b, c)\}$

Figure 4: Sussman's anomaly.

- every atom $\text{on}(b, c)$ in I such that b's position in I is consistent with G;
- an atom $\text{on}(b, \mathcal{T})$ for every block b such that nothing is below b in G and b's position in I is inconsistent with G.

Then G' is the state consisting of F plus an atom $\text{clear}(b)$ for every block b at the top of a maximal stack in F.

Since $G \subseteq G'$, every plan for $B' = (I, G')$ is also a plan for B. As shown in Appendix C, G' is the final state produced by Solve-EBW(I, G) and by every execution trace of Solve-EBW-Optimally(I, G). From this it follows that every optimal plan for B' is also an optimal plan for B.

What this means is that planning in EBW and planning in PBW are basically equivalent. If you have a planner that will find optimal plans for PBW problems, and if you want to find an optimal plan for an EBW problem B, then you can do this by computing B' as described above, and using your planner on B'. Thus, all of our results about EBW apply equally well to PBW: finding non-optimal plans is easy, finding optimal plans is NP-hard, resolving deleted-condition interactions is easy, resolving deadlocks is difficult, etc. In fact, the theorems and proofs in Appendices A and B apply to PBW with no modifications, except for replacing "EBW" by "PBW".

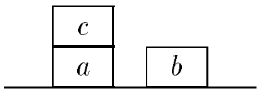
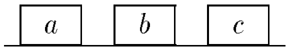
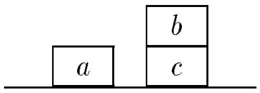
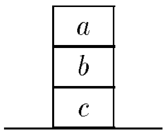
4 Why EBW Planning is Hard

In this section, we show that the difficulty of planning in EBW is not due to deleted-condition interactions, but instead due to the difficulty of determining the best way to resolve multiple deadlocks. We also discuss the difference between deadlocks and deleted-condition interactions.

4.1 The Difficulty of Resolving Multiple Deadlocks

To see that deadlocks make planning difficult in EBW, consider our proof that EBW PLAN is NP-hard. This proof (see Appendix B) involves reducing FEEDBACK ARC SET to EBW PLAN. For each digraph (V, E), our reduction produces an EBW problem B, such that finding a set of k blocks that resolves all deadlocks in B corresponds to finding a feedback arc set of size k in (V, E). But the difficulty of finding a small feedback arc set

Table 2: Successive states generated by Solve-EBW on Sussman's anomaly.

State	Block	Position	Consistent with G
	a	a on $\mathcal{T}$	no
	b	b on $\mathcal{T}$	no
	c	c on a on $\mathcal{T}$	no
	a	a on $\mathcal{T}$	no
	b	b on $\mathcal{T}$	no
	c	c on $\mathcal{T}$	yes
	a	a on $\mathcal{T}$	no
	b	b on c on $\mathcal{T}$	yes
	c	c on $\mathcal{T}$	yes
	a	a on b on c on $\mathcal{T}$	yes
	b	b on c on $\mathcal{T}$	yes
	c	c on $\mathcal{T}$	yes

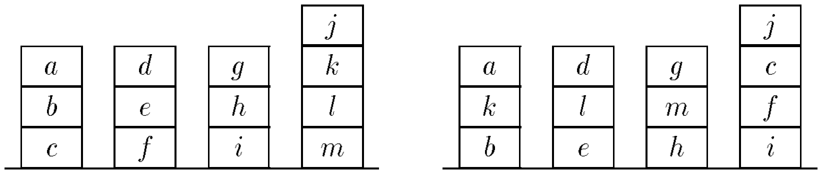
makes the FEEDBACK ARC SET problem NP-hard. Thus, the difficulty of finding a small set of blocks that resolves all deadlocks makes EBW PLAN NP-hard.

To see that deadlocks are the *only* thing that makes planning difficult in EBW, note that in Solve-EBW-Optimally, the only time nondeterminism is required is to resolve a deadlock. For EBW problems that contains no deadlocks, Solve-EBW-Optimally will never enter Step 7, which is the only step where nondeterminism occurs. Thus, for such problems, the deterministic algorithm Solve-EBW will always find an optimal plan in time $O(n^3)$.

To illustrate this, below we consider two EBW problems: one without deadlocks, and one with deadlocks.

Example 1. Consider Sussman's anomaly (shown in Figure 4). a and b are not in deadlocked sets, because there are no blocks below them in I; and c is not in a deadlocked set, because there is no block below it in G. Thus Sussman's anomaly contains no deadlocks, so Solve-EBW can solve it easily, as we now show.

Table 2 shows the successive states and positions generated by Solve-EBW on Sussman's anomaly. Initially, none of the blocks are in positions consistent with G, and neither a nor b can be moved to positions consistent with G. c can be moved to a position consistent with G by moving it to the table, so Solve-EBW does this in Step 6. Once this is done, the positions of a and b are still inconsistent with G, but b's position can be made consistent with G by moving it to c, and Solve-EBW does this in Step 5. At this point, a's position is inconsistent with G but can be made consistent with G by moving it to b, and Solve-EBW does this in Step 5. Now the current state is consistent with G, so Solve-EBW exits with success in Step 4.



The initial state I The goal state G

Figure 5: In this problem, different ways of resolving the deadlocks produce plans of different lengths.

Example 2. Consider the EBW problem shown in Figure 5. This problem contains six deadlocked sets: $\{a\}$, $\{d\}$, $\{g\}$, $\{a, j\}$, $\{d, j\}$, and $\{g, j\}$. In the initial state, every clear block is in one of these deadlocked sets. Moving a, d, or g to the table resolves two deadlocks, and moving j to the table resolves three deadlocks. Thus, moving j to the table might appear to be the most attractive choice—but it will not result in an optimal plan.

Every plan for this problem that involves moving block j to the table contains at least 16 actions. However, there are plans for this problem that do not move j to the table and contain only 15 actions. For example, below are two plans produced by two of the nondeterministic execution traces of Solve-EBW-Optimally: one in which j is moved to the table and one in which it is not.

1. move$(j, k, \mathcal{T})$, move$(a, b, \mathcal{T})$, move$(b, c, \mathcal{T})$, move(k, l, b), move$(a, \mathcal{T}, k)$, move$(d, e, \mathcal{T})$, move$(e, f, \mathcal{T})$, move(l, m, e), move$(d, \mathcal{T}, l)$, move$(g, h, \mathcal{T})$, move$(h, i, \mathcal{T})$, move$(m, \mathcal{T}, h)$, move$(g, \mathcal{T}, m)$, move$(f, \mathcal{T}, i)$, move$(c, \mathcal{T}, f)$, move$(j, \mathcal{T}, c)$.

2. move$(a, b, \mathcal{T})$, move$(b, c, \mathcal{T})$, move$(d, e, \mathcal{T})$, move$(e, f, \mathcal{T})$, move$(g, h, \mathcal{T})$, move$(h, i, \mathcal{T})$, move$(f, \mathcal{T}, i)$, move$(c, \mathcal{T}, f)$, move(j, k, c), move(k, l, b), move$(a, \mathcal{T}, k)$, move(l, m, e), move$(d, \mathcal{T}, l)$, move$(m, \mathcal{T}, h)$, move$(g, \mathcal{T}, m)$.

The reason why moving j to the table is not part of any optimal plan is that although moving j to the table resolves three deadlocks ($\{a, j\}$, $\{d, j\}$, and $\{g, j\}$), it leaves three other deadlocks unresolved ($\{a\}$, $\{d\}$, and $\{g\}$), and the only possible way to resolve these deadlocks is to move a, d, and g to the table. But moving a, d, and g to the table resolves all of the deadlocks involving j, leaving no need to move j to the table.

4.2 Deadlocks Versus Deleted Conditions

It is important to understand that deadlocks are different from deleted-condition interactions. In a deleted-condition interaction, the side-effect of achieving one condition is to delete some other condition that will be needed later. In contrast, in a deadlock situation there are several goal conditions left to be achieved, none of which can be directly achieved. Of the actions available to achieve subgoals for these goals, some will achieve several subgoals at once, and the question is which of these actions to use.

Initial state: $I = \{\text{clear}(a), \text{on}(a, c), \text{on}(c, \mathcal{T}), \text{clear}(b), \text{on}(b, d), \text{on}(d, \mathcal{T})\}$

Goal formula: $G = \{\text{on}(b, c), \text{on}(a, d)\}$

Figure 6: A problem that contains a deadlock but no deleted-condition interactions.

Below, we illustrate the difference between deadlocks and deleted-condition interactions, by describing two planning problems: one that contains deleted-condition interactions but no deadlocks, and one that contains a deadlock but no deleted-condition interactions.

Example 3. Sussman's anomaly (see Figure 4) is well-known as an example of a planning problem in which deleted-condition interactions occur regardless of the order in which one tries to achieve the goals:

1. Suppose one tries to achieve on(a, b) first and on(b, c) second. The way to achieve on(a, b) is to move c to the table and a to b. But once this has been done, one must undo on(a, b) in order to achieve on(b, c).

2. Suppose one tries to achieve on(b, c) first and on(a, b) second. The way to achieve on(b, c) is to move b to c. But once this has been done, one must undo on(b, c) in order to achieve on(a, b).

However, as shown in Example 1, Sussman's anomaly contains no deadlocks.

Example 4. Consider the planning problem shown in Figure 6. In the initial state, a is above c and b is above d, and in the goal state, a is above d and b is above c, so $\{a, b\}$ is a deadlocked set. However, in this problem, neither goal ordering produces deleted-condition interactions:

1. Suppose one tries to achieve on(a, d) first and on(b, c) second. The way to achieve on(a, d) is to move b to the table and then move a to d. Once this has been done, the way to achieve on(b, c) is to move b to c, and this does not delete on(a, d).

2. Suppose one tries to achieve on(b, c) first and on(a, d) second. The way to achieve on(b, c) is to move a to the table and then move b to c. Once this has been done, the way to achieve on(a, d) is to move a to d, and this does not delete on(b, c).

5 Generalizations of EBW

Although EBW is the best-known version of the blocks world, it is not the only one. For example, Winograd's original version of the blocks world [23] included complications such as pyramids and blocks of different sizes. Below, we consider three such generalizations of EBW:

- VBW, in which there can be blocks, pyramids, and frustums of pyramids, all of which may vary in size; and no object a can sit on an object whose top face is smaller than a's bottom face.
- LBW, in which the table can hold only a limited number of blocks.
- VLBW, which has the features of both VBW and LBW.

Our results for these planning problems can be summarized as follows:

1. Planning in VBW is so similar to planning in EBW that all of our results about planning in EBW apply equally well to VBW.
2. LBW requires different planning algorithms than the ones we presented earlier for EBW, but its time complexity is not very different from EBW's. Finding a non-optimal plan is easy, finding an optimal plan is NP-hard but no worse, and optimal plans can be found nondeterministically in polynomial time.
3. Planning in VLBW is more difficult. In particular, there are VLBW problems for which the shortest plan has exponential length.

The details appear below.

5.1 Blocks World with Varying Block Sizes

Varying Block-Size Blocks World (VBW) is like EBW, except that for each block b there is a positive integer k_b denoting the size of b's bottom face, and a nonnegative integer $h_b \leq k_b$ denoting the size of b's top face (b is a pyramid if $h_b = 0$, and it is a frustum of a pyramid if $h_b < k_b$).[3] Just as in EBW, each block b either is clear or else has a unique block a sitting on it—but a can sit on b only if b's top face is at least as large as a's bottom face.[4] Thus, move(b, c, d) has the same preconditions as in EBW, plus the requirement that either $d = \mathcal{T}$ or else $k_b \leq h_d$.

VBW PLAN is the following decision problem:

> Given a VBW problem (I, G, K) (where K is a list giving the sizes of each block's top and bottom faces) and an integer $L > 0$, is there a plan for this problem of length L or less?

In VBW, a formula F is consistent if and only if the following conditions are satisfied:

1. F contains no atoms of the form "on$(\mathcal{T}, x)$";

[3] Since b's top and bottom faces are both square, it is immaterial whether h_b and k_b denote area, perimeter, or the length of an edge bounding the face.

[4] Another possible generalization would be to allow more than one block to sit on b simultaneously. In that case, whether other blocks could be placed on b would depend on where a is located on b, making the problem much more complicated.

2. The Hasse diagram $H(F)$ consists of one or more disjoint acyclic paths, with each clear block at the beginning of a different path;

3. $k_a \leq h_b$ for all blocks a and b such that F contains on(a, b).

The first two conditions are identical to to those required in EBW, and it is easy to check the third condition. Thus, just as in EBW, one can check the consistency of a VBW formula in time $O(n)$.

Using the above definition of consistency, all of the results we stated earlier for EBW hold for VBW as well, with only minor modifications needed in the proofs (we leave these modifications as exercises to the reader). A list of these results appears later, in Section 6.

It is easy to see that the same results also apply to other generalizations of EBW that are not as general as VBW. For example, for each b we could require $h_b = k_b$, in which case the blocks may vary in size but pyramids and frustums are not allowed. Or for each b we could require $k_b = 1$ and $h_b \in \{0, 1\}$, in which pyramids are allowed, frustums are not allowed, and all blocks must be the same size. In such cases, the same results still hold.

5.2 Blocks World with Limited Table Capacity

Limited Table-Capacity Blocks World (LBW) is like EBW, except that there can be at most h number of blocks sitting directly on the table. Thus, move(b, c, d) has the same preconditions as in EBW, plus the requirement that if $d = \mathcal{T}$ then the current state S must contain less than h atoms of the form "on$(x, \mathcal{T})$".

Given an LBW problem $B = (I, G, h)$ (where h is the size of the table), checking whether or not B is solvable is somewhat more complicated than for EBW and VBW problems, but it can still be done in low-order polynomial time. Below, we explain how to test whether B is solvable, and how to find a (not necessarily optimal) plan for B if it is solvable:

1. If $h = 2$, then B is solvable if and only if I contains at most two maximal stacks, G mentions no block not mentioned in I, and I can be transformed to a state consistent with G by moving blocks from one of these stacks to the other. If this can be done, then it yields a plan of length at most m, and this is an optimal plan.

2. If $h \geq 3$, then B is solvable if and only if (I, G) is a solvable EBW problem and neither I nor G contains more than h atoms of the form "on$(x, \mathcal{T})$". If these conditions are satisfied, then by examining the Hasse diagram $H(G)$ it is easy to find a state G' consistent with G such that G' contains at most h maximal stacks. Any plan for (I, G') is also a plan for (I, G). Let $M_1, M_2, \ldots, M_{h'}$ be the maximal stacks in G', where $h' \leq h$. We can produce a plan for (I, G') in the following manner:

   - Move all blocks to two temporary stacks T_1 and T_2. This will take less than m moves.
   - Construct the stacks $M_1, M_2, \ldots, M_{h'-2}$, by moving blocks back and forth between T_1 and T_2 to expose blocks that can be moved directly to their final positions. The number of moves this will require is less than $m+(m-1)+\ldots+(k+1) = m(m+1) - k(k+1)$, where k is the total number of blocks in $M_{h'-1}$ and $M_{h'}$.

- Move all remaining blocks in T_1 to the top of M_1, and all remaining blocks in T_2 to the top of M_2, creating temporary stacks T_1' and T_2' on top of M_1 and M_2, respectively. This will take less than k moves.
- Construct $M_{h'-1}$ and $M_{h'}$, by moving blocks back and forth between T_1' and T_2' to expose blocks that can be moved directly to their final positions. The number of moves this will require is less than $k + (k-1) + \ldots + 1 = k(k+1)$.

The length of the above plan is less than $m(m+1) + 2m$.

The above technique can be modified to produce a plan of length $O(m \log m) = O(n \log n)$, by sorting the blocks in T_1 and T_2 into an appropriate order before starting to construct the stacks M_i. The details of this modification are left to the reader.

LBW PLAN is the following decision problem:

> Given an LBW problem (I, G, h) and an integer $L > 0$, is there a plan for this problem of length L or less?

Since EBW is a special case of LBW, it is clear that LBW PLAN is NP-hard. But optimal plans for LBW problems can be found nondeterministically in polynomial time. Thus, LBW PLAN is NP-complete, and it can be shown that LBW is no worse than NP-hard. These results are proved in Appendix D.

Since EBW is a special case of LBW, it is clear that deadlocks are difficult to solve in LBW. However, we did not investigate whether or not deleted-condition interactions are hard to solve in LBW. This is still an open question.

5.3 Blocks World with Varying Block Sizes and Limited Table Capacity

Varying Block-Size, Limited Table-Capacity Blocks World (VLBW) is like EBW, except that it incorporates the features of both VBW and LBW: for each block b, the top and bottom faces have sizes h_b and k_b respectively, and the table has a capacity $h_{\mathcal{T}}$. Thus, move(b, c, d) has the same preconditions as in EBW, plus the requirement that either $k_b \leq h_d$, or else $d = \mathcal{T}$ and the current state S contains less than $h_{\mathcal{T}}$ atoms of the form "on$(x, \mathcal{T})$".

VLBW PLAN is the following decision problem:

> Given a VLBW problem (I, G, K) (where K is a list giving the table capacity and the sizes of each block's top and bottom faces) and an integer $L > 0$, is there a plan for this problem of length L or less?

VLBW PLAN includes VBW PLAN as the special case in which $h_{\mathcal{T}} \geq$ the total number of blocks. Thus since VBW PLAN is NP-hard, so is VLBW PLAN.

In EBW, VBW and LBW, the problem of finding an optimal plan is NP-hard, but it can be solved nondeterministically in polynomial time. In contrast, there are some VBW problems for which nondeterminism will not enable us to produce a plan in polynomial time, because the shortest plan has exponential length. We prove this in Appendix E, by showing how to reduce the Towers of Hanoi problem to VLBW in polynomial time, in a manner that preserves plan length.

The above consideration suggests that VLBW PLAN is not in NP, but does not demonstrate it conclusively. Even though one cannot produce an optimal plan nondeterministically in polynomial time, it still might be possible to determine the length of that plan nondeterministically in polynomial time. This can be done in certain special cases, such as the Towers of Hanoi Problem [1] and certain generalizations of it [11], but we have not explored whether or not it can be done in general. Thus, we do not know whether or not VLBW PLAN is in NP.

6 Summary of Results

In the previous sections, we have shown the following:

1. Given an Elementary Blocks World (EBW) problem, one can tell in time $O(n \log n)$ whether or not it is solvable. If it is solvable, then one can produce in time $O(n)$ a plan that moves no block more than twice, and in time $O(n^3)$ a plan whose length is no more than twice optimal.

2. Given an EBW problem and a positive integer L, the problem of answering whether there is a plan of length L or less is NP-complete. Thus, the problem of finding an optimal plan is NP-hard. However, it is no worse than NP-hard, and there is a nondeterministic algorithm that can solve it in time $O(n^3)$.

3. If an EBW problem contains no deadlocked sets, then one can find an optimal plan deterministically in time $O(n^3)$. Thus, the NP-hardness of finding an optimal plan is due to deadlocks.

4. Deadlocks are different from deleted-condition interactions. In particular, there are some problems that contain deadlocks but no deleted-condition interactions, and other problems that contain deleted-condition interactions but no deadlocks.

5. Given an EBW problem, in time $O(n^3)$ one can formulate additional conditions to add to the goal formula, to produce a completely specified goal state such that any optimal plan for achieving this state is also an optimal plan for the original problem. Thus, all of the above results also apply to PBW (the special case of EBW in which the goal state is completely specified).

6. All of the above results also apply to VBW (a generalization of EBW in which there can be pyramids, frustums of pyramids, and blocks of different sizes). Furthermore, they also apply to other versions of the blocks world intermediate between EBW and VBW (for example, if we restrict VBW to disallow frustums, or to allow pyramids but require all blocks to have the same size).

7. LBW (a generalization of EBW in which the table capacity is limited) requires different planning algorithms than the ones we developed for EBW, but its time complexity is not very different. In low-order polynomial time, one can tell whether or not an LBW problem is solvable, and produce a plan of length $O(n \log n)$ if the problem is solvable. Given an LBW problem and a positive integer L, the problem of answering whether there is a plan of length L or less is NP-complete, so the problem of finding an optimal plan is NP-hard. However, the problem is no worse than NP-hard, and there is a nondeterministic algorithm that can solve it in polynomial time.

8. In VLBW (a generalization of EBW which incorporates both the features of both VBW and LBW), planning is more difficult. Given a VLBW problem and a positive integer L, the problem of answering whether there is a plan of length L or less is NP-hard. There is no nondeterministic polynomial-time algorithm to find optimal plans for VLBW problems, because there are some VLBW problems in which the shortest plan has exponential length.

7 Related Work

The first results on the computational complexity of blocks-world planning appeared at AAAI-91. These included our NP-completeness proof for PBW PLAN [12], and Chenoweth's NP-hardness proof for a problem we will call MPBW PLAN [5]. Since PBW is a special case of MPBW, our result subsumed Chenoweth's—but his proof and examples were different from ours, and they are worth discussing here because they provide additional insight into the nature of blocks-world planning.

Multiple-Copy Primitive Blocks World (MPBW) is like PBW except that more than one block can have the same name. *MPBW PLAN* is the following decision problem:

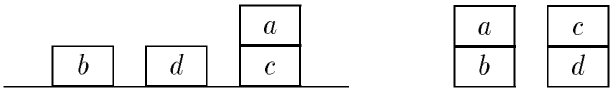
> Given an MPBW problem (I, G) and an integer $L > 0$, is there a plan for this problem of length L or less?

One of Chenoweth's examples [5, Fig. 2] is an example of a particular kind of deleted-condition interaction which is sometimes called "creative destruction" [4]. In this example, some of the goal conditions are satisfied in the initial state, and one can produce a non-optimal plan that preserves these conditions, but in order to produce the optimal plan one must undo them.

This example is interesting because it suggests that in MPBW, unlike PBW, deleted-condition interactions might be hard to solve. However, this is still an open question, because Chenoweth's proof that MPBW PLAN is NP-hard does not depend on deleted-condition interactions. Instead, it depends on a problem somewhat similar to the problem of resolving multiple deadlocks.

Chenoweth's proof of NP-hardness is by reduction from 3SAT. Given a 3SAT problem with m clauses and n variables, he generates an MPBW problem in which $L = 3n + 5m + 1$. For each i ($i = 1, \ldots, n$), there are two blocks named u_i, at the tops of two large stacks. For each i, one of the two u_i's must be moved to the top of a block named v_i, and the question is which u_i to move. If we move the wrong one, then later in the plan, we will have to move one or more blocks temporarily to the table rather than moving them directly to their final positions, whence the resulting plan will be longer than L.

The above problem is similar to the problem of resolving multiple deadlocks in PBW. In both problems, if we make the wrong choice, then too many blocks must be moved temporarily to the table rather than directly to their final positions. However, the two problems are not identical. If no two blocks have the same name, then for a wrong choice to force us to move extra blocks to the table, we must have blocks which mutually block each others' progress—and this led to our definition of deadlock. But if more than one block can have the same name, then one can find other ways for a wrong choice to force us to move extra blocks to the table—and that is what Chenoweth did.



Initial state: $I = \{\text{clear}(b), \text{on}(b, \mathcal{T}), \text{clear}(d), \text{on}(d, \mathcal{T}), \text{clear}(a), \text{on}(a, c), \text{on}(c, \mathcal{T})\}$

Goal formula: $G = \{\text{on}(a, b), \text{on}(c, d)\}$

Figure 7: In this problem, moving a to b enables us to move c to d.

8 Discussion and Conclusions

In this paper, we have discussed a well-known planning domain which we call the Elementary Blocks World (EBW). We have shown that in EBW and in several other related planning domains, the problem of finding shortest-length plans is NP-hard (but no worse), even if the goal state is completely specified. These results are interesting for two reasons:

1. For the special case of EBW in which the goal state is completely specified, different planning researchers have had conflicting intuitions about the difficulty of finding shortest-length plans—and our results answer the question.
2. Planning in EBW is difficult for an unexpected reason. One of the primary roles of EBW in planning research was in the discovery and investigation of deleted-condition interactions such as creative destruction and Sussman's anomaly [4, 15, 16, 18, 19, 20, 21], in which the plan for achieving one goal or subgoal deletes another goal or subgoal. However, our results show that in EBW and several related planning domains, such interactions can easily be handled by a simple hill-climbing strategy. The complexity of planning in these domains is instead due to the difficulty of resolving multiple deadlocks.

To clarify the significance of deadlocks, we now formulate a domain-independent explanation of them.

8.1 Enabling-Condition Interactions

Let us define an *enabling-condition interaction* to be a situation in which some action invoked to achieve one goal G_1 also makes it easier to achieve another goal G_2. For example, in Figure 7, the action move(a, c, b) achieves the goal on(a, b), but it also has the side-effect of clearing c, making it easier to achieve the goal on(c, d). As another example, consider the following situation (based on [22]):

> John lives two miles from a bakery and two miles from a dairy. The two stores are one mile apart. John has two goals: to buy bread and to buy milk.

If John goes to the bakery to buy bread, then this puts him closer to the dairy, making it easier for him to buy milk. Hayes-Roth and Hayes-Roth's transcript of someone "thinking aloud" while planning a hypothetical day's errands illustrates how people look for such interactions when formulating plans [13, p. 254]:

In section 6, the subject asks, "What is going to be the closest one?" This question indicates a strategic decision to plan to perform the closest errand next in the procedural sequence ...

If a problem contains more than one enabling-condition interaction, then it can be difficult to determine which set of actions will produce the best plan. For example, if actions A and B both achieve goal G_1, and A also aids in achieving goal G_2, then we might prefer action A to action B—but if B also aids in achieving goal G_3, then it may no longer be clear which of A and B we should prefer. The difficulty of resolving such tradeoffs is illustrated in some of the repeated revisions that Hayes-Roth and Hayes-Roth's subject makes to his plan [13, p. 246–247].

In this paper we have seen two cases of multiple enabling-condition interactions, and in both cases, these interactions make it NP-hard to find an optimal plan:

1. *Chenoweth's planning problem* [5], which we discussed in Section 7. This problem occurs in a version of the blocks world in which more than one block can have the same name. In it, there are two different blocks named u_i, and we can achieve the goal on(u_i, v_i) by moving either one of them to v_i. As side-effects, these two possible moves make different sets of goals easier to achieve later on—and thus it is not clear which of the two moves we should prefer. This same kind of difficulty occurs for multiple values of i, making it NP-hard to find an optimal plan.

2. *Resolving multiple deadlocks.* For example, suppose the set of blocks $D_1 = \{a, b\}$ is deadlocked. Then before we can move a and b to their final positions, we must resolve the deadlock by moving either a or b out of the way. Now, suppose that a is also in some other deadlocked set D_2, and b is also in some other deadlocked set D_3. Then moving a out of the way will also resolve D_2, and moving b out of the way will also resolve D_3. Thus, these two possible moves will have side-effects of making different sets of goals easier to achieve later on, so it is unclear which of of the two moves we should prefer. As we discussed in Section 4.1, in the general case of this problem it is NP-hard to find an optimal plan.

8.2 Future Work

Our results suggest several questions for future research—for example, whether or not there are any other important kinds of goal and subgoal interactions, and how easily various kinds of interactions might be handled in various planning domains. Recent studies of the complexity of planning have shown that even with very restricted planning operators, domain-independent planning is an extremely difficult task [3, 7, 8, 9]. But if one could characterize what makes various interactions easy or hard to handle across various classes of planning domains, this might make it possible to produce planners that, although not completely domain-independent, are efficient across significant classes of planning domains. For example, in situations where the only possible interactions are certain restricted kinds of enabling-condition and deleted-condition interactions, the work described in [24, 25] provides some efficient algorithms for merging plans to achieve multiple goals.

Acknowledgement

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

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1976.

[2] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning.* Morgan-Kaufmann, San Mateo, CA, 1990.

[3] Tom Bylander. Complexity results for planning. In *IJCAI-91*, 1991.

[4] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA, 1985.

[5] Stephen V. Chenoweth. On the NP-hardness of blocks world. In *AAAI-91: Proc. Ninth National Conf. Artificial Intelligence*, pages 623–628, July 1991.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw Hill, 1990.

[7] K. Erol, D. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. Submitted for publication. Available as Computer Science Dept. TR-2797, UMIACS TR-91-154, or Systems Research Center TR 91-96, University of Maryland, 1991.

[8] K. Erol, D. Nau, and V. S. Subrahmanian. On the complexity of domain-independent planning. In *Proc. AAAI-92*, to appear, 1992.

[9] K. Erol, D. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proc. First Internat. Conf. AI Planning Systems*, to appear, 1992.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[11] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: a Foundation for Computer Science.* Addison-Wesley, 1989.

[12] Naresh Gupta and Dana S. Nau. Complexity results for blocks-world planning. In *Proc. AAAI-91*, 1991. Honorable mention for the best paper award.

[13] B. Hayes-Roth and F. Hayes-Roth. A cognitive method of planning. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 245–262. Morgan Kaufman, 1990. Originally appeared in *Cognitive Science* 3(4), 1979.

[14] D. E. Knuth. *The Art of Computer Programming-Volume 1.* Addison-Weseley Publ. Co., 1968.

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

[16] Peter Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Morgan Kaufman, 1992.

[17] F. P. Preparata and R. T. Yeh. *Introduction to Discrete Structures.* Addison-Wesley, Reading, MA, 1973.

[18] Earl D. Sacerdoti. The nonlinear nature of plans. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 206–214. Morgan Kaufman, 1990. Originally appeared in *Proc. IJCAI-75.*

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

[20] R. Waldinger. Achieving several goals simultaneously. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 118–139. Morgan Kaufman, 1990. Originally appeared in *Machine Intelligence 8*, 1977.

[21] D. H. D. Warren. Extract from Kluzniak and Szapowicz APIC studies in data processing, no. 24, 1974. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 140–153. Morgan Kaufman, 1990.

[22] R. Wilensky. *Planning and Understanding.* Addison-Wesley, 1983.

[23] Terry Winograd. *Understanding Natural Language.* Academic Press, New York, 1972.

[24] Q. Yang, D. S. Nau, and J. Hendler. Optimization of multiple-goal plans with limited interaction. In *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.

[25] Q. Yang, D. S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, to appear.

A Theorems and Proofs for Section 3.1

Theorem 1 *Let $B = (I, G)$ be any solvable EBW problem, and P be any plan for B. Then there is a plan Q for B such that $|Q| \leq |P|$ and Q has the following properties:*

1. *For every block b whose position in I is consistent with G, b is never moved in Q.*

2. *Q moves no block more than twice.*

3. *For every block b that is moved more than once, the first move is to the table.*

4. *For every block b that is moved to a location $d \neq \mathcal{T}$, $on(b, d) \in G$.*

5. *Let b be any block that is moved more than once. Then in the state immediately preceding the first time b is moved, no block whose position is inconsistent with G can be moved to a position consistent with G.*

Proof. Suppose there is a block b whose position in I is consistent with G such that P moves b. Below, we prove by induction that there is a shorter plan P_1 for B. By applying this argument repeatedly, it follows that there is a plan P_2 that satisfies Property 1. The induction proof is as follows:

Base case: I contains on$(b, \mathcal{T})$. Then by removing from P all actions that move b, we still have a plan for G.

Induction step: Suppose X contains on(b, c) for some block c, and suppose our proof holds for all blocks below b. c's position in S must be consistent with G, so from the induction assumption, there must be a plan P' for G with $|P'| \leq |P|$, such that c is not moved. In P', first remove all actions that move b, and then replace all occurrences of c by occurrences of $\mathcal{T}$. Then the resulting plan P'' is a plan for G.

Suppose that some b is moved more than twice in P_2, and let move(b, u, v) and move(b, x, y), respectively, be the first and last actions in P_2 that move b. If we replace them by move$(b, u, \mathcal{T})$ and move$(b, \mathcal{T}, y)$, respectively, and remove all other actions that move b, then the resulting plan P_3 is a plan satisfying Property 1 such that $|P_3| \leq |P_2|$. By applying this argument repeatedly, we can produce a plan P_4 satisfying Properties 1 and 2.

Let b be any block that is moved more than once in P_4. Then from Property 2, b is moved exactly twice, so let the two actions that move b be $A_1 =$ move(b, u, v) and $A_2 =$ move(b, v, w). There are two cases:

1. $w = \mathcal{T}$. Then a plan P_5 shorter than P_4 can be produced by removing A_2 and replacing A_1 with move$(b, u, \mathcal{T})$.

2. $w \neq \mathcal{T}$. Then replacing A_1 by move$(b, u, \mathcal{T})$ and A_2 by move$(b, \mathcal{T}, w)$ will produce another plan P_5 for B having length no greater than that of P.

By applying the above argument repeatedly, it follows that there is a plan P_6 satisfying Properties 1, 2, and 3.

In P_6, for every action $A =$ move(b, c, d) such that $d \neq \mathcal{T}$, this is the last time that b is moved in P_6. Therefore, unless on$(b, d) \in G$, replacing this action with move$(b, c, \mathcal{T})$ will produce another plan P_7 for B having length equal to that of P'. By applying this argument repeatedly, it follows that there is a plan P_8 satisfying Properties 1, 2, 3, and 4.

Let b be any block in P_8 that is moved more than once, $A_1 =$ move$(b, c, \mathcal{T})$ be the first action that moves b, and S be the state immediately prior to this action. Suppose that in S, there is a block e whose position is inconsistent with G but which can be moved to position consistent with G. Then later in P_8, there must be an action $A_2 =$ move(e, f, g) that moves e to a position consistent with G. There are two cases:

1. $g = \mathcal{T}$. Since it is possible to move e in S, it is certainly possible to move it to the table, and this cannot possibly interfere with any of the remaining actions in P_8 other than the action A_2 itself. Thus, removing A_2 from P_8 and reinserting it just before A_1 will produce another plan P_9 for B having length equal to that of P_8.

2. $g \neq \mathcal{T}$. Then from Property 4, on$(e, g) \in G$. But if our supposition is true that in S, e can be moved to a position consistent with G, then it must be that g's position

in S is consistent with G. It follows from Property 1 that in the portion of P_8 that comes after S, neither g nor any of the blocks below it is moved. Therefore, moving e from f to g before the action move$(b, c, \mathcal{T})$ cannot possibly interfere with any of the remaining actions in P_8 other than the action A_2 itself. Therefore, removing the action A_2 from P_8 and reinserting it just before A_1 will produce another plan P_9 for B having length equal to that of P_8.

By applying the above argument repeatedly, it follows that there is a plan Q satisfying Properties 1–5. In none of the above steps did we increase the number of actions in the plan, so it follows that $|Q| \leq |P|$. ■

Corollary 1 *Any plan Q satisfying Properties 1–5 also has $m - q \leq |Q| \leq 2(m - q)$, where m is the total number of blocks and q is the number of blocks in I whose positions are consistent with G.*

Proof. Every block whose position in I is inconsistent with G must be moved at least once. There are $m-q$ such blocks, so $|Q| \geq m-q$. But Q moves no block whose position in I is consistent with G, and the other blocks it moves at most twice. Therefore, $|Q| \leq 2(m-q)$. ■

Corollary 2 *B has an optimal plan satisfying Properties 1–5.*

Proof. Let P be any optimal plan for B. From the theorem, $|Q| \leq |P|$, so Q is also optimal. ■

Corollary 3 *All plans produced for B by Solve-EBW and Solve-EBW-Optimally satisfy Properties 1–5.*

Proof. This follows immediately from an examination of the algorithms' steps. ■

Corollary 4 *Solve-EBW-Optimally will find an optimal plan for B.*

Proof. Solve-EBW-Optimally generates every plan for B satisfying Properties 1–5. Thus from Corollary 2, Solve-EBW-Optimally will find an optimal plan. ■

Corollary 5 *The length of an optimal plan for B is $m - q + r$, where r is the minimum number of times that Step 7 is executed in any of the execution traces of Solve-EBW-Optimally.*

Proof. Immediate from Corollary 4. ■

B Theorems and Proofs for Section 3.2

Lemma 1 *EBW PLAN is in NP.*

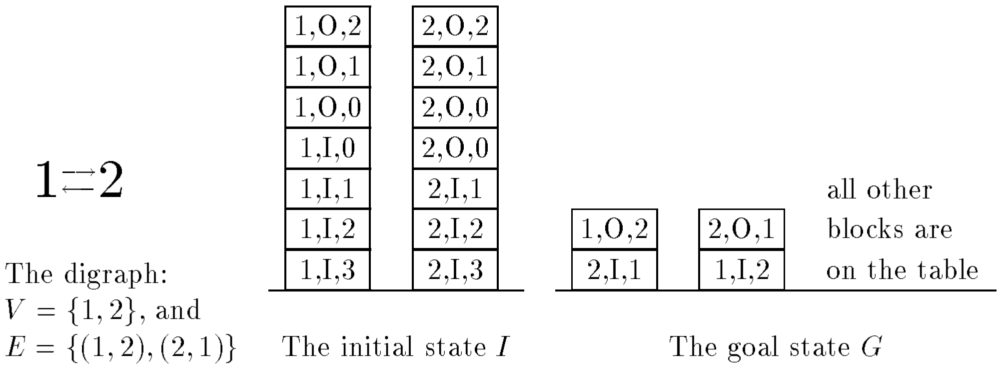


Figure 8: A digraph (V, E), and the EBW problem (I, G) produced by $M(V, E, k)$.

Proof. The following nondeterministic algorithm solves EBW PLAN in polynomial time.

> Algorithm Solve-EBW-PLAN(I, G, L)
> If Solve-EBW-Optimally(I, G) finds a plan P such that $|P| \leq L$, then return *True*. Otherwise return *False*.

■

Definition. If (V, E) is a digraph, then without loss of generality we may assume that V is the set of integers $\{1, 2, \ldots, p\}$ for some p. If (V, E, k) is an instance of FEEDBACK ARC SET, then we define $M(V, E, k)$ to be the following instance (I, G, L) of EBW PLAN, where $L = 2p^2 + 2p + k$, I and G are as defined below:

- For each $v \in V$, I contains stack of $2p + 3$ blocks, whose names (from the top of the stack to the bottom) are $[v, O, p]$, $[v, O, p-1]$, $\ldots$, $[v, O, 0]$, $[v, I, 0]$, $\ldots$, $[v, I, p]$, and $[v, I, p+1]$ (for example, see Figure 8). Thus, I consists of p stacks of $2p + 3$ blocks each, for a total of $2p^2 + 3p$ blocks.
- For every edge (x, y) in E, G contains the atom on$([x, O, y], [y, I, x])$. For every other block b mentioned in I, G contains on$(b, \mathcal{T})$. For every block b mentioned in I such that there is no block c such that on$(b, c) \in G$, G contains clear(b). Thus, G specifies a state consisting of $|E|$ stacks of 2 blocks each, and $2p^2 + 3p - |E|$ blocks sitting on the table by themselves.

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

For the rest of this section, we let (V, E, k) be any instance of FEEDBACK ARC SET, and $(I, G, L) = M(V, E, k)$. Note that in I, the only blocks that are in their final positions are $[1, I, p+1], [2, I, p+1], \ldots, [p, I, p+1]$. Thus, there are $2p^2 + 2p$ blocks that are not in their final positions.

Lemma 2 *For each simple cycle in (V, E) there is a corresponding deadlocked set in (I, G), and vice versa.*

Proof. Suppose (V,E) contains a simple 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 G, 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 I, 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 in G, each block $b \in D$ must be on some other block c. But from the definition of (I, G), this means there are v, w such that $b = [v, O, w]$ and $c = [w, I, v]$. 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 G 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 (I, G), this means that $(z_1, z_2, \ldots, z_p, z_1)$ is a simple cycle in (V, E). ■

As an example of Lemma 2, note that in Figure 8, the simple cycle $(1, 2, 1)$ in (V, E) corresponds to the deadlocked set of blocks $\{[1, O, 2], [2, O, 1]\}$ in (I, G).

Lemma 3 *(I, G) has a plan of length L or less iff (V, E) has a feedback arc set of size k or less.*

Proof. $(\rightarrow)$: Suppose (I, G) has a plan of length L or less. Then from Corollary 2, there is an optimal plan P of length L or less that satisfies the properties of Theorem 1. Let T be the set of all blocks that are moved more than once in P, and U be the set of all blocks that are moved exactly once. Then from Theorem 1, each block in T is moved exactly twice (once to the table and once to its final position), so $|P| = 2|T| + |U|$. But since $2p^2 + 2p$ blocks are not in their final positions, $|T| + |U| = 2p^2 + 2p$. Therefore,

$$|T| = |P| - (2p^2 + 2p) \leq L - (2p^2 + 2p) = k.$$

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 $b \in T$. From the proof of Lemma 2, $b = [v, O, w]$ for some edge $(v, w) \in E$. Thus, T 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 (V, E) contains one of the edges (v_1, w_1), …, (v_j, w_j), so (V, E) has a feedback arc set of size $j \leq |T| \leq k$.

$(\leftarrow)$: Suppose (V, E) has a feedback arc set $F = \{(v_1, w_1), \ldots, (v_q, w_q)\}$ such that $q \leq k$. In the operation of Solve-EBW-Optimally(I, G), Step 6 will never be executed, because G specifies the position of every block. Thus, since I contains $2p^2 + 2p$ blocks that are not in their final positions, Step 5 of Solve-EBW-Optimally will be executed $2p^2 + 2p$ times. Each time Solve-EBW-Optimally enters Step 7, 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 simple cycle in (V, E), so at least one block $[v_i, O, w_i] \in D$ corresponds to an edge $(v_i, w_i) \in F$. But moving $[v_i, O, w_i]$ to the table will resolve the deadlock. Thus, there is an execution trace for Solve-EBW-Optimally(I, G) in which all deadlocks are resolved by moving to the table blocks in the set $\{[v_1, O, w_1], \ldots, [v_q, O, w_q]\}$, whence Step 7 is executed at most q times. Thus, one of the execution traces for Solve-EBW-Optimally finds a plan P of length $|P| \leq 2p^2 + 2p + q \leq 2p^2 + 2p + k = L$. ■

Theorem 2 *EBW PLAN is NP-complete.*

Proof. Lemma 3 shows that M reduces FEEDBACK ARC SET to EBW PLAN. Since M runs in polynomial time, this means that EBW PLAN is NP-hard. But Lemma 1 shows that EBW PLAN is in NP. Thus, EBW PLAN is NP-complete. ■

Theorem 3 *Finding optimal plans for EBW problems is NP-hard, but no worse.*

Proof. If one can find an optimal plan for an EBW problem, then for any L one can immediately tell whether there is a plan of length L or less. Thus from Theorem 2, finding an optimal plan is NP-hard.

To prove that finding optimal plans in EBW is no worse than NP-hard, we show that it is Turing-reducible to EBW PLAN.[5] Suppose we have an oracle which, given an instance (I, G, L) of EBW PLAN, tells whether the answer is yes or no. Then given any EBW problem $B = (I, G)$, we can find the length L of the optimal plan for B by repeatedly guessing a value for L and asking the oracle to solve (I, G, L). Once we know L, we can identify the first action in an optimal plan by repeatedly guessing a first action A and asking the oracle to solve $(I', G, L-1)$, where I' is the state produced by applying A to I. Once we have identified the first move, we can identify the subsequent moves in a similar manner. This will involve at most polynomially many calls to the oracle. ■

C Theorems and Proofs for Section 3.3

Theorem 4 *Let (I, G) be any solvable EBW problem. Then every execution trace of Solve-EBW-Optimally(I, G) produces the same final state $G' = G_1 \cup G_2 \cup G_3 \cup G_4$, where*

$$\begin{aligned}
G_1 &= \{on(b,c) | on(b,c) \in G\};\\
G_2 &= \{on(b,c) \in I | b\text{'s position in } I \text{ is consistent with } G\};\\
G_3 &= \{on(b,\mathcal{T}) | b\text{'s position in } I \text{ is inconsistent with } G \text{ and for all } y,\ on(b,y) \notin G\};\\
G_4 &= \{clear(b) | b \text{ is the top of a maximal stack in } G_1 \cup G_2 \cup G_3\}.
\end{aligned}$$

Proof. Let G' be a final state produced by any of the execution traces of Solve-EBW-Optimally(I, G). For each block b, G' contains exactly one atom of the form "on(b, y)". There are three cases for what this atom is:

1. If on$(b, c) \in G$, then $y = c$, for otherwise G' is inconsistent with G.
2. If b's initial position is consistent with G, then Solve-EBW-Optimally never moves b, so y is the block b was on in I.
3. If b's initial position is inconsistent with G but G contains no atom of the form "on(b, x)", then Solve-EBW-Optimally moves b to the table in Step 5 and never moves b again, so $y = \mathcal{T}$.

[5] For more details on how to do this kind of proof, we refer the reader to pp. 115–117 of [10].

From the above, it follows that the set of all "on" atoms in G' is $G_1 \cup G_2 \cup G_3$. The clear blocks in G' are precisely those blocks that are at the tops of maximal stacks in G', so the set of all "clear" atoms in G' is G_4. Thus, $G' = G_1 \cup G_2 \cup G_3 \cup G_4$. ■

Corollary 6 *Solve-EBW(I, G) produces G' in time $O(n^3)$, and G' is consistent with G.*

Proof. The execution trace of Solve-EBW(I, G) is identical to one of the execution traces of Solve-EBW-Optimally(I, G), and Solve-EBW runs in time $O(n^3)$. Thus Solve-EBW(I, G) produces G' in time $O(n^3)$. Thus G' is consistent with G, because Solve-EBW(I, G) does not exit until its current state is consistent with G. ■

Corollary 7 *PBW PLAN is NP-complete.*

Proof. From Corollary 6, it follows that Solve-EBW can be used to reduce any instance (I, G, L) of EBW PLAN to an instance (I, G', L) of PBW PLAN in polynomial time. Thus PBW PLAN is NP-hard. Since every PBW problem is an EBW problem, it follows from Lemma 1 that PBW PLAN is in NP. ■

Theorem 5 *Finding optimal plans for PBW problems is NP-hard, but no worse.*

Proof. The proof of this theorem is basically the same as the proof of Theorem 3. The details are left to the reader. ■

D Theorems and Proofs for Section 5.2

Theorem 6 *LBW PLAN is NP-complete.*

Proof. Given any EBW problem (I, G), one can easily produce an equivalent LBW problem (I, G, k) by letting k be the total number of blocks in I. Thus since EBW PLAN is NP-hard, so is LBW PLAN.

To see that LBW PLAN is in NP, consider the following nondeterministic algorithm:

Algorithm Solve-LBW-Optimally(I, G, k):
Step 1: $S \leftarrow I$.
Step 2: If S is consistent with G, then exit with success.
Step 3: If Step 4 has been executed more than $m^3 + 3m$ times, then exit with failure.
Step 4: Nondeterministically select any clear block b whose position is not consistent with G, and nondeterministically choose where to move it. Then go to Step 2.

For any LBW problem (I, G, k), Solve-LBW-Optimally will find an optimal plan P nondeterministically in polynomial time. One can tell whether or not there is a plan of length L or less by checking whether or not $|P| \leq L$. ■

Theorem 7 *Finding optimal plans for LBW problems is NP-hard, but no worse.*

Proof. The proof of this theorem is basically the same as the proof of Theorem 3. The details are left to the reader. ■

E Theorems and Proofs for Section 5.3

The Towers of Hanoi problem can be described as follows: there are p disks $d_1, d_2, \ldots, d_p$, and three locations p_1, p_2, p_3. Initially, all the disks are at location p_1, with d_1 on d_2 on $\ldots$ on d_p. The goal is to get all the disks to p_3 by moving them one at a time, with the restriction that one cannot put a disk d_i onto a disk d_j unless $i < j$. It is well known (for example, see [1]) that the shortest plan for solving this problem has length $2^p - 1$.

Theorem 8 *In polynomial time, the Towers of Hanoi problem can be reduced to VLBW in a manner that preserves plan length.*

Proof. Suppose we are given an instance H of the Towers of Hanoi problem in which there are disks $d_1, d_2, \ldots, d_p$. We will map this into the VLBW problem B defined below.

B contains blocks $b_1, b_2, \ldots, b_p$ corresponding to H's disks, and three more blocks c_1, c_2, c_3 to represent the locations p_1, p_2, p_3, respectively. To insure that a block b_i can be put onto a block b_j if and only if $i < j$, we need to make $k_{b_i} < h_{b_j}$ if and only if $i < j$. We satisfy these requirements by setting $h_{b_i} = k_{b_i} = i$ for $i = 1, 2, \ldots, p$, and setting $h_{c_i} = k_{c_i} = p + 1$ for $i = 1, 2, 3$. To insure that there can never be more than three maximal stacks, we let the table capacity be $h_{\mathcal{T}} = 3$. The initial state contains three maximal stacks:

b_1 on b_2 on $\ldots$ on b_{p-1} on b_p on c_1;
c_2;
c_3.

The final state also contains three maximal stacks:

c_1;
c_2;
b_1 on b_2 on $\ldots$ on b_{p-1} on b_p on c_3.

Given H, one can produce B in polynomial time. Clearly, each plan for H corresponds move-for-move to a plan for B, with the optimal plan for H corresponding to the optimal plan for B. ■