ELSEVIER

Artificial Intelligence 67 (1994) 71–112

Artificial Intelligence

Partial-order planning: evaluating possible efficiency gains

Anthony Barrett, Daniel S. Weld*

Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195, USA

Received July 1992; revised February 1993

Abstract

Although most people believe that planners that delay step-ordering decisions as long as possible are more efficient than those that manipulate totally ordered sequences of actions, this intuition has received little formal justification or empirical validation. In this paper we do both, characterizing the types of domains that offer performance differentiation and the features that distinguish the relative overhead of three planning algorithms. As expected, the partial-order (nonlinear) planner often has an advantage when confronted with problems in which the specific order of the plan steps is critical. We argue that the observed performance differences are best understood with an extension of Korf's taxonomy of subgoal collections. Each planner quickly solved problems whose subgoals were independent or trivially serializable, but problems with laboriously serializable or nonserializable subgoals were intractable for all planners. Since different plan representations induce distinct search spaces, the subgoals for a given problem may be trivially serializable for one planner, laboriously serializable for another, and nonserializable for a third. We contend that the partial-order representation yields superior performance because it more frequently results in trivial serializability.

1. Introduction

Since the early work on NOAH [29], the common wisdom of the planning community has been that nonlinear planners are more efficient than linear algorithms, but this intuition has never been convincingly demonstrated. Furthermore, the very term "linear planner" is often confusingly given two different meanings:

(1) a planner that represents plans as totally ordered sets of actions (i.e., manipulates linear lists of actions);

* Corresponding author. E-mail: weld@nooksack.cs.washington.edu.

SSDI 0004-3702(93)E0023-F

(2) a planner that focuses problem solving attention on one subgoal, shifting to another only after the first has been completely addressed. [1]

We argue that the factors of plan representation and subgoal selection can and should be considered independently. In this paper we focus on the former and hold the latter fixed; we evaluate the relative efficiency of total-order and partial-order representations in planners that focus on a single subgoal before shifting to the next goal. [2] To alleviate confusion, we follow the advice of Drummond and Currie [8] and avoid the adjective "linear" in the rest of this paper. By holding the subgoal elaboration strategy fixed, we present an objective evaluation of early ordering commitment on planning efficiency.

We found no problem domains in which a total-order planner performed significantly better than an equivalent partial-order planner, but several domains in which the partial-order algorithm was exponentially faster than the total-order planners. The contribution of this paper is a careful characterization of the types of domains in which a partial-order planner beats total-order approaches (the requisite features are rather subtle) and a description of domains in which *both* approaches encounter intractable branching. We argue that the observed performance differences are best understood with an extension of Korf's taxonomy of subgoal collections [17]. Each planner performed well when dealing with problems whose subgoals were independent or trivially serializable, but problems with laboriously serializable or nonserializable subgoals were intractable. Since the different plan representations induce different search spaces, the subgoals for a given problem may be trivially serializable for one planner, laboriously serializable for another, and nonserializable for a third. We believe that the partial-order representation yields superior performance because it more frequently produces trivial serializability.

1.1. Algorithms and methodology

We performed this evaluation by implementing three planners that share key subroutines but differ in important ways. All planners operate on action schemata that conform to the STRIPS representation [11].

The planner that turned out to perform the best is a lifted version of McAllester and Rosenblitt's [19] propositional planner; since it represents plans with a partial order and uses tagged pointers, called "causal links", to

[1] In fact, Sussman's original definition of the "linear assumption" is satisfied only by a planner that assumes that subgoals can be solved independently and in any order [32, p. 58], but few consider this a viable strategy.

[2] Note that it is crucial to distinguish between a planner's plan-time and execution-time commitments. A planner's decision to plan for subgoal U before subgoal V is not necessarily related to the decision to execute the actions corresponding to U strictly before, strictly after or interleaved with those generated for V. In fact, in a partial-order planner, these decisions are *necessarily* distinct while some total-order planners link the decisions and some do not. As long as a planner does not link these decisions, subgoal ordering is independent of completeness.

mark protections, we call it *POCL*.[3] The second planner represents plans as totally ordered sequences of steps; since this planner also uses causal links to determine appropriate locations for new steps, it is called the "total-order, causal-link planner", or *TOCL*. The third, and simplest, planner is called the "total-order, prior-insertion planner" (*TOPI*) since it dispenses with causal links and only adds steps prior to the existing steps of an incomplete plan. To assure a fair comparison, the three planners share data structures and utility routines to the maximum extent possible.

We tested the set of planners on large sets of randomly generated problems from both classical (e.g., the blocks world, transportation planning, and a reconstruction of Stefik's [31] MOLGEN molecular biology domain) and artificial domains. In this paper we limit our report almost exclusively to artificial domains. While the "real" domains were a rich source of intuitions, the difficulty of decoupling different causes of combinatorial explosion made them uninformative testbeds for empirical experiments. Specifically, planners make two types of combinatorial choices: deciding how to achieve a goal and deciding when to do so. Most real problems are fraught with both sources of intractability, but neither partial- nor total-order representation provides much guidance in the problem of choosing *how* to achieve a goal. Since we are interested in the utility of partial-order plan representations, we focus this paper on the combinatorics of ordering decisions. Thus most of our artificial domains include only one method for achieving each type of goal, but these methods interact in rich and complex ways. However, since there is interaction between the choice of operator used to achieve a goal and choice of the order in which the resulting steps are executed, Section 3.6 does explore artificial domains with significant operator selection complexity and Section 3.8 briefly discusses experience with a "real" domain.

Another advantage of artificial domains is the ability to quantify the difficulty of problems. In real domains it is extremely difficult to come up with such a measure—the number of objects in the world, the number of subgoals, the length of an optimal solution, and related measures are much too crude to yield any useful generalizations. The regularity of an artificial domain facilitates such a measure (even for randomly generated problems), enabling a precise estimate of the asymptotic complexity growth as problems get harder. Of course, the analysis of artificial domains is not an end in itself, but by decoupling the myriad causes of planning complexity, they provide insight into the difficulties implicit in conventional domains.

We used the following experimental methodology. In each domain and for each difficulty level, we generated a fixed number of random problems which were given to each of the three planners. We display the data by graphing the mean CPU time required by the planners at each difficulty level as well

[3] A COMMON LISP implementation of this algorithm, known as *SNLP*, has become quite popular as a framework for AI research and education. Send mail to bug-snlp@cs.washington.edu for information on acquiring the source code for the three planners and for the domains mentioned in this paper.

as 90% confidence intervals. With probability 0.9 the mean of all possible problems at a particular difficulty level is within the interval. We terminated each planner's performance curve when the difficulty became so great that it could not successfully complete all problems in the random suite within the time bound. Depending on the domain, we varied the number of problems that we generated per difficulty level from five to thirty in an effort to keep the confidence intervals small.

1.2. Contributions

Our paper presents two major results: an extension to Korf's classification of subgoals and a series of experiments comparing the performance of our three planners on eight different domains. We link the contributions by analyzing the experimental results in terms of our augmented taxonomy.

Korf's [17] insightful definition of independent, serializable, and nonserializable collections of subgoals forms the foundation of our work. However, we argue that the definition of subgoal independence is so strong that in practice it rarely applies, and we observe that while nonserializable subgoals are always difficult, many serializable problems are almost as hard. This leads us to refine Korf's class of serializable subgoals with the following new classes:

- A set of subgoals is *trivially serializable* if each subgoal can be solved sequentially in any order without ever violating past progress. As we explain in Section 3, trivial serializability is considerably more general than independence, yet results in comparable performance.
- A set of subgoals is *laboriously serializable* if there exist an inadequate percentage of orders in which the subgoals may be solved without ever violating past progress. When it is difficult or impossible to determine the correct order, laboriously serializable subgoals are just as intractable as nonserializable ones.

Fig. 19 (Section 4) shows the extended hierarchy of subgoal collections that results from our analysis. We also extend Korf's treatment of subgoals from search through states of the world to search through a space of incomplete plans, since this is the representation of choice in modern planners [6]. With this reformulation the computational advantages of the various algorithms becomes clearer: the natural subgoal decomposition of a problem might be trivially serializable for the search space of one planner, laboriously serializable for another, and nonserializable for a third (Table 3, Section 4). In fact, the artificial domain D^1S^2 (Section 3.5) has exactly this property.

In addition to classifying problems in terms of their subgoal structure, we performed a series of experiments to evaluate the relative performance of the three planners. In no domain did either of the total-order planners perform significantly better than the partial-order planner; however, in some cases the partial-order planner *POCL* did exponentially better than either total-order algorithm. Both causal-link planners outperformed *TOPI* on all but the simplest problems and domains.

In all our tests there was a clear correspondence between the classification of a problem's natural subgoals and the speed of the planner. All three algorithms took exponential time to solve problems in which the subgoals were laboriously serializable or nonserializable yet took apparently linear (or low-order polynomial) time on domains with independent or trivially serializable subgoals. We conclude that the major advantage of using a partial-order planning algorithm derives from the fact that it renders many subgoal collections trivially serializable.

1.3. Outline

In the next section we formally define the class of problems that we are trying to solve. We then give pseudocode descriptions for the three planning algorithms, *POCL*, *TOCL*, and *TOPI*, and present a complexity analysis of their operation. The bulk of the paper is our analysis of the performance of the planners on a number of different domains. We start, in Section 3, by extending Korf's [17] characterization of problem domains, and then we perform a sequence of experiments that isolate the domain features that give the partial-order planner a major advantage over total-order approaches. Section 4 summarizes our results and proves several generalizations. Related and future work are discussed in Sections 5 and 6 respectively. Finally, Section 7 closes by stating our contributions.

2. Planners

Before presenting our results, we summarize the algorithms and representations used. Each planner uses what is known as the STRIPS action representation [5] although it is in fact a simplification of that used by STRIPS [7,11]. Each operator has sets of preconditions, an add list and a delete list (the members of which are propositional schemata that are function-free atomic) and a set of codesignation (and noncodesignation) constraints. For example, the blocks world operator (`puton ?x ?y`),[4] which takes block `?x` from `?z` and puts it on `?y`, is shown in Fig. 1.

Note the codesignation constraints listed in the `:equals` field. They specify that `?x`, `?y`, and `?z` must refer to different blocks. Also neither `?x` nor `?y` can refer to `Table`. The variables mentioned in an action are only used to define constraints between a step's variables. A unique set of variables is created and used whenever a new step is created. Codesignation constraints between variables of different steps are added to an incomplete plan to constrain a step's possible effects. For example, a step with action (`puton` $?x_1$ $?y_1$) can be constrained to clear block `C` by adding the codesignation constraint: (`=` $?z_1$ `C`).

[4] Symbols that start with question marks denote variables (which are also known as formal objects [20]).

```
(defoperator :action  '(puton ?x ?y)
             :precond '((on ?x ?z) (clear ?x) (clear ?y))
             :add     '((on ?x ?y) (clear ?z))
             :delete  '((on ?x ?z) (clear ?y))
             :equals  '((<> ?x ?y) (<> ?x ?z) (<> ?y ?z)
                        (<> ?x Table) (<> ?y Table)))
```

Fig. 1. An operator to move a block ?x off of ?z and onto ?y.

Although the limitations of this action representation have been clearly documented [5], we have succeeded in encoding a number of domains, including the blocks world, several artificial worlds, a discrete time version of Minton's scheduling world [21], a simple transportation scheduling world, and an approximation of Stefik's MOLGEN molecular biology domain [31].

The planners each require three arguments: a set of operators, a set of initial conditions, and a set of goal conditions; they return sequences of steps. All planners treat variables the same way in that they use least-commitment, constraint-posting techniques when reasoning about the arguments to the operators, and all planners operate via backward chaining. To ensure fairness, the planners were implemented in COMMON LISP using a shared set of data structures and subroutines. The most important such subroutine is the variable binding and unification code that handles all of the variable constraints.

2.1. Planning as search

Like [17], we view planning as a search problem. In order to discuss our planners, we need to define a planning problem, and how it can be considered as a search. From [19] we adopt:

Definition 2.1. A *STRIPS operator* consists of an operator name plus a *precondition list*, an *add list* and a *delete list*. The elements of the precondition, add, and delete lists are all function-free, atomic expressions. A *STRIPS planning problem* is a triple $\prec \mathcal{O}, \Sigma, \Omega \succ$ in which $\mathcal{O}$ denotes a set of STRIPS operators, Σ denotes a set of initial propositions, and Ω denotes a set of goal propositions.

Previous analyses of planning problems [13,17] viewed planning as a search through a graph of world-states—i.e., a graph in which nodes are labeled with a set of propositions that specify what is true in that state of the world. A STRIPS planning problem can be solved by searching through such a graph. Σ specifies the initial world-state, Ω specifies a set of goal world-states, and the operators in $\mathcal{O}$ specify the directed edges. If an operator's preconditions are satisfied in a world-state, then an edge leads from that node to the node denoting the effect of applying that operator. The purpose of the search is to find a path from the initial world-state to a goal world-state. The solution to the planning problem consists of the actions associated with the edges of this

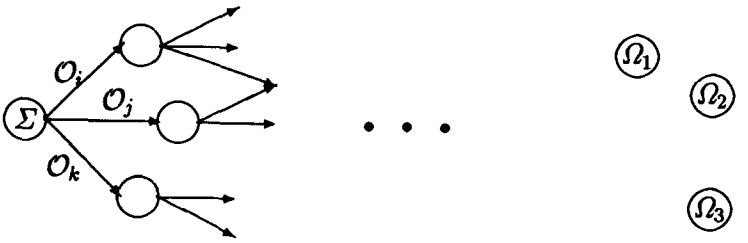

Fig. 2. A STRIPS planning problem $\prec\mathcal{O}, \Sigma, \Omega\succ$ can be solved by searching through a graph of world-states. The goal is to find a path from Σ to a world-state labeled with $\Omega_i \supseteq \Omega$. Edges out of a world-state correspond to operators that can be performed in that world-state.

path. An example of such a search space appears in Fig. 2.

One of the major contributions of Sacerdoti's NOAH [29] was a conceptual shift: instead of viewing planning as search through a space of world-states, NOAH searched through a space of (possibly incomplete) plan-states. Our planners perform similar searches.

Definition 2.2. A *plan-state* is a triple: $\prec S, O, B\succ$ in which S denotes a set of plan steps (also known as actions), O denotes a set of ordering constraints that specify a (possibly partial) order on S, and B denotes a set of binding constraints over the variables mentioned by the steps in S.

This shift makes the structure of a search space dependent on the planner as well as the domain. The arcs between world-states were just determined by domain actions, but the arcs between plan-states represent the extension of an incomplete plan (i.e., the addition of a step, ordering constraint or binding constraint) rather than the regression of a world description.

In plan-state search, a planning problem is encoded in an *initial plan-state* $\prec S, O, B\succ$ consisting of two steps s_0 and s_∞. The step s_0 adds Σ, and s_∞ has Ω for preconditions. This plan-state has no variable-binding constraints, but O has one constraint to force s_0 before s_∞. The set $\mathcal{O}$ defines what can be added to the set S in a plan-state. The goal of the search is to find a solution plan-state.

Definition 2.3. A *solution plan-state* $\prec S, O, B\succ$ is a plan-state in which the preconditions of each step $s_i \in S$ are all necessarily true in the input situation of s_i.

To make this definition precise, we recall the following terminology from Chapman's formalization of planning [5]. The *input situation* of a step is a set of propositions that are true immediately prior to the execution of that step [5]. A proposition is *necessarily true* in the input situation of a step in a partially ordered plan-state $\prec S, O, B\succ$ when it is true in all completions that extend the partial constraints of O and B into total constraints.

2.2. Algorithms

In order to test our intuitions regarding how representations of the space of plan-states affect planning difficulty, we implemented three different planning algorithms. Each of these algorithms is sound and complete and each exhibits what McAllester terms the "systematic" property [19].[5] Loosely speaking, systematicity means that each algorithm is guaranteed to search among plan-states in an irredundant fashion—visiting every possible plan-state exactly once. This property is reflected in the structure of each planner's search space in that the directed edges form a tree rooted in the initial plan-state.

The first algorithm, called *POCL* and shown in Fig. 3, uses a partially ordered step representation for defining plans. *POCL* is a lifted version of McAllester's algorithm. The algorithm is loosely descended from TWEAK [5] and NONLIN [33], but is conceptually simpler. Like some previous planners (e.g., [14,33–35]) but unlike TWEAK, McAllester's algorithm uses *causal links* to record the purpose for introducing a step into a plan and to protect that purpose. If a step S_i adds a proposition p to satisfy a precondition of step S_j, then $S_i \xrightarrow{p} S_j$ denotes the causal link. McAllester's key innovation is a clever, methodical technique for creating and protecting causal links. We say that a link $S_i \xrightarrow{p} S_j$ is *threatened* if some step S_k may possibly be ordered between S_i and S_j, and S_k either deletes or adds[6] a proposition that possibly unifies with p. Two propositions possibly unify if they can be unified by adding variable-binding constraints to a plan-state without making that plan-state's variable constraint set B inconsistent.

Each precondition p of a plan-state step S_j is an *open condition* if it has no corresponding causal link $S_i \xrightarrow{p} S_j \in L$. The algorithm searches for a solution plan-state $\prec S, O, B \succ$ by eliminating open conditions in G while ensuring the safety of causal links in L. In the initial invocation of *POCL*, $\prec S, O, B \succ$ is the initial plan-state, G is the set of preconditions of s_∞, and L is the empty set.

Our use of least commitment for variable bindings has a subtle effect on the causal-link-protection step. In the absence of unbound variables, protecting a causal link $s_i \xrightarrow{p} s_j$ from a step s_k simply involved ordering s_k before s_i or after s_j. With the introduction of unbound variables we get the extra possibility of adding variable constraints between the effects of s_k and the proposition p. For example, there are five different sets of constraints that can be added to protect

---

[5] We note that the utility of systematicity has not been clearly documented. We suspect that for some domains systematic planners will be more efficient than nonsystematic algorithms while the converse will hold in other domains. Since a detailed evaluation of the utility of systematicity is beyond the scope of this paper, we feel that holding systematicity constant in our experiments increases their validity.

[6] Steps that add p threaten the causal link $S_i \xrightarrow{p} S_j$ because they negate the purpose for adding step S_i. *POCL* would not be systematic if it ignored these threats.

Algorithm: $POCL(\prec S, O, B \succ, G, L)$

(1) **Termination:** If G is empty, report success and stop.

(2) **Goal selection:** Let c be a proposition in G, and let S_{need} be the step for which c is a precondition.

(3) **Operator selection:** Let S_{add} be a step that adds c (either a new step or an existing step possibly prior to S_{need}). If no such step exists then backtrack. Let $L' = L \cup \{S_{add} \xrightarrow{c} S_{need}\}$, $S' = S \cup \{S_{add}\}$, $O' = O \cup \{S_{add} \prec S_{need}\}$, and $B' = B \cup$ {the set of variable bindings to make S_{add} add c}. *Backtrack point: Each existing and possibly addable step must be considered for completeness.*

(4) **Update goal set:** Let $G' = (G - \{c\}) \cup$ {preconditions of S_{add}, if new}.

(5) **Causal-link protection:** A step s_k *threatens* a causal link $s_i \xrightarrow{p} s_j$ when it occurs between s_i and s_j and it adds or deletes p. For every step s_k that might threaten a causal link $s_i \xrightarrow{p} s_j \in L'$:
- Ensure that s_k does not threaten $s_i \xrightarrow{p} s_j$ by adding constraints to O' and/or B'. *Backtrack point: Each way to protect $s_i \xrightarrow{p} s_j$ from s_k must be considered for completeness.*

(6) **Recursive invocation:** $POCL(\prec S', O', B' \succ, G', L')$.

Fig. 3. The partial-order, causal-link (*POCL*) algorithm.

$$s_i \xrightarrow{\text{(on ?x ?y)}} s_j$$

from a step s_k that deletes `(on ?a ?b)`.

(1) {s_k before s_i},

(2) {s_k after s_j},

(3) {s_k between s_i and s_j, `?x` $\neq$ `?a`, `?y` $\neq$ `?b`},

(4) {s_k between s_i and s_j, `?x` $\neq$ `?a`, `?y` $=$ `?b`},

(5) {s_k between s_i and s_j, `?x` $=$ `?a`, `?y` $\neq$ `?b`}.

The addition of codesignation as well as noncodesignation constraints is required to ensure systematicity. In general, the number of ways to protect a causal link is exponential in the number of unbound variables involved.

The second algorithm, *TOCL*, is similar to *POCL*, but it restricts its use of plan-states by only generating plans comprised of totally ordered sets of steps. It can insert steps anywhere between s_0 and s_∞ in a plan, but it can never reorder two existing steps. The algorithm (Fig. 4) is similar to *POCL* except there is an extra linearization step, and causal links are only used to determine possible locations for new steps.

Since *TOCL* is just a modification of *POCL*, its calling conventions are identical, and its search space is very similar to that of *POCL*. The similarity is due to the fact that they start with the same initial plan-state and each partially ordered plan produced by *POCL* corresponds to a set of totally ordered plans. Each of these totally ordered plans are generated by *TOCL* whenever *POCL* generates the partially ordered plan. This similarity lets us compare the two

Algorithm: $TOCL(\prec S, O, B \succ, G, L)$

(1) **Termination:** If G is empty, report success and stop.

(2) **Goal selection:** Let c be a proposition in G, and let S_{need} be the step for which c is a precondition.

(3) **Operator selection:** Let S_{add} be a step that adds c (either a new step or an existing step which is necessarily prior to S_{need}). If no such step exists then backtrack. Let $L' = L \cup \{S_{add} \xrightarrow{c} S_{need}\}$, $S' = S \cup \{S_{add}\}$, and $B' = B \cup$ {set of variable bindings to make S_{add} add c}. If S_{add} is a new step, let $R = (S_{initial}, S_{need})$, the ordered pair of existing steps that bound the places to insert S_{add} into the plan. *Backtrack point: Each existing and possibly addable step must be considered for completeness.*

(4) **Update goal set:** Let $G' = (G - \{c\}) \cup$ {preconditions of S_{add}, if new}.

(5) **Causal-link protection:** For every step s_k that might threaten a causal link $s_i \xrightarrow{p} s_j \in L'$:
- Protect the causal link from s_k by adding constraints to B'. Also, if S_{add} is new, either s_i or s_k is S_{add} and the link can be protected by replacing one of the bounds in R. *Backtrack point: Each way to protect* $s_i \xrightarrow{p} s_j$ *from* s_k *must be considered for completeness.*

(6) **Linearization:** If S_{add} is a new step, let $O' = O \cup$ {constraints to insert S_{add} into the plan-state at a point between the steps in R}. Otherwise, let $O' = O$. *Backtrack point: Each insertion point between R's steps must be considered for completeness.*

(7) **Recursive invocation:** $TOCL(\prec S', O', B' \succ, G', L')$.

Fig. 4. The total-order, causal-link (*TOCL*) algorithm.

algorithms using techniques developed in [22].

The third algorithm, *TOPI*, only adds steps to the beginning of the plan (i.e., immediately after s_0). Thus it can be seen that *TOPI* is equivalent to the regression planner of Nilsson [25, Section 7.4] which performs backward chaining search through the space of lifted world-states.

TOPI works by defining the goal conditions as planning subgoals and building a plan backwards (Fig. 5). It considers all steps that could possibly add a subgoal without deleting any other unsolved subgoal. When it finds such a step it creates a new plan-state by adding that step between s_0 and every other step already in the plan, eliminating the resolved subgoals and adding the weakest preconditions for the new step as new subgoals. The planner terminates when all of the open goals of a plan G unify with the initial conditions I.

Since causal links were used to guide the placement of new steps in a plan, and since *TOPI* only inserts steps at one point, causal links are not needed by *TOPI*. Although *TOPI* does not require data structures for links or step ordering, it shares with the other algorithms the data structures and routines for variable bindings and constraints.

Algorithm: $TOPI(\prec S, O, B \succ, I, G)$

(1) **Termination:** If $G \subseteq I$, report success and stop.

(2) **Operator selection:** Let S_{add} be a new step that adds a set of conditions A such that $(A \cap G) \neq \emptyset$, does not delete $g \in G$, and has a set of preconditions C. Let $S' = S \cup \{S_{add}\}$, $O' = O \cup$ {ordering constraints that make S_{add} come after s_0 but before any other step in S}, and $B' = B \cup$ {constraints to make S_{add} add A and not delete any element of G}. *Backtrack point: All possibly added steps and variable constraints must be considered for completeness.*

(3) **Update goal set:** Let G' be the set $(G - A) \cup C$

(4) **Recursive invocation:** $TOPI(\prec S', O', B' \succ, I, G')$.

Fig. 5. The total-order, prior-insertion (*TOPI*) algorithm.

2.3. Per-step complexity

Comparing the performance of these three algorithms requires looking at the number of plan-states each algorithm generates when solving a planning problem and the computational cost per plan-state. Comparing the number of plan-states generated requires looking at planning problems, but the per-plan-state complexity can be inferred from the algorithm descriptions.

From the *POCL* and *TOCL* algorithm descriptions we see that steps to detect termination, select goals, and update the goal set all require constant time. The major points where the algorithms differ in terms of per-plan-state complexity are the steps selecting an operator to solve a goal and protecting a causal link. They arise from the different ways that step orderings are performed. One of the problems in operator selection is to determine the existing steps that might solve the selected goal; this takes $O(|S|^2)$ time for partially ordered steps and $O(|S|)$ time for totally ordered steps. Similarly, causal-link-protection only involves protecting a new link from existing steps and existing links from a new step. For this reason there are only $O(|S|)$ threats to resolve, and the loop only executes $O(|S|)$ times. Detecting these threats requires $O(|S|^2)$ time for *POCL* and only $O(|S|)$ time for *TOCL*. Resolving a threat takes constant time.

In contrast with the causal-link algorithms, the *TOPI* algorithm's per-plan-state complexity does not depend on the number of steps at all. It depends on the number of unsolved goals $|G|$. The most costly step in *TOPI* is the termination detection step because finding a set of variable bindings to make G a subset of I takes exponential time. We can prove that this problem is NP-hard by reducing the 3-Dimensional Matching problem [12], which is NP-complete, to it. The step by step per-plan-state complexity comparison is summarized in Table 1.

Table 1
The complexity of each step in the three planning algorithms for a plan-state $\prec S, O, B \succ$. The number of ordering constraints O can be $O(|S|^2)$. For *TOPI* the sets I and G are the initial conditions and open goals respectively

	Algorithm		
Step	*POCL*	*TOCL*	*TOPI*
Termination	$O(1)$	$O(1)$	$O(\|I\|^{\|G\|})$
Goal selection	$O(1)$	$O(1)$	—
Operator selection	$O(\|O\|)$	$O(\|S\|)$	$O(\|G\|)$
Update goal set	$O(1)$	$O(1)$	$O(1)$
Causal-link protection	$O(\|O\|)$	$O(\|S\|)$	—
Linearization	—	$O(1)$	—

3. Analysis of domains

In the view of [13,17] planning is modeled as search through a directed graph of world-states, and the difficulty of a problem is measured in terms of how hard it is to break it up into subproblems and use these subproblems to guide the search for a final solution.

We wish to analyze the effect of reformulating problems as plan-state searches. In order to compare the different search spaces of our planners we need to consider them in the context of solving problems. We start by discussing subgoals and how they decompose world-state searches and plan-state searches. Next we review and extend Korf's subgoal hierarchy. Finally we define example domains and show how the classification of subgoals varies from planner to planner.

3.1. Subgoals

Like [13,17] we are concerned with analyzing the effect of using various subgoals on the speed of planning algorithms. In the simplest case a subgoal is an intermediate state on the path from initial state to goal. Intuitively, it is clear that searching from the initial state to the subgoal and then again from the subgoal to the goal might be faster than searching all the way in one step. Korf presents a broader definition of *subgoal*, which we adopt for this paper.

> In general, a subgoal is not a single state but rather a property that is true of a number of states. For example, if we establish a subgoal for the Eight Puzzle of correctly positioning a particular tile, this subgoal is satisfied by any state in which that tile is in its goal position, regardless of the positions of the remaining tiles. Therefore, we formally define a *subgoal* to be a set of states, with the interpretation that a state is an element of a subgoal set if and only if it has properties that satisfy the subgoal. [17, p. 68]

It is frequently awkward to refer to subgoals explicitly as sets of states. A common technique used in [13,17] is to use elements of Ω to specify subgoals. A world-state is in a subgoal if the subgoal's associated goal proposition is true in the world-state. This supposes that world-states with more elements of Ω are closer to a goal world-state than those with less elements of Ω.

The conceptual shift to planning with plan-states affects this technique for specifying a subgoal. When planning with plan-states we think in terms of the satisfiability of various elements of Ω. We can do this with the following formal definition.

Definition 3.1. Let $\prec\mathcal{O}, \Sigma, \Omega\succ$ be a STRIPS planning problem and let $P \in \Omega$ denote one of the goal propositions. The *subgoal specified by P with respect to* $\prec\mathcal{O}, \Sigma, \Omega\succ$ is a set, U, of plan-states such that every $\prec S, O, B\succ \in U$ satisfies:

(1) S contains a step s_0 which only adds Σ.
(2) S contains a step s_∞ which only requires Ω.
(3) Every total order of S consistent with O has s_0 and s_∞ as the first and last steps, respectively.
(4) P is necessarily true in the input situation of s_∞.

Proving that a proposition P is necessarily true can be done using Chapman's modal truth criterion [5]. Certain properties of our algorithms make this proof process easier. For example, in *TOPI* the operator selection step has the restriction that added steps cannot delete open goals. This and the fact that *TOPI* can only add steps to the beginning of an incomplete plan ensures that a proposition is necessarily true when it is either in the initial conditions or it is added by a step and all of that step's preconditions are necessarily true. Thus a goal proposition P is necessarily true once all of the open preconditions of steps added to solve P are in the initial conditions.

The causal-link-protection step of *TOCL* and *POCL* makes it easy to prove the necessary truth of a proposition. A goal proposition P is necessarily true if it has an associated causal link $s_i \xrightarrow{P} s_\infty$, and all of the preconditions of step s_i are necessarily true. The causal-link-protection step ensures that no steps ever interfere with the truth of P. Thus P is necessarily true once all of preconditions of all of the steps involved in solving P have associated causal links.

3.2. Subgoal hierarchy

Frequently, a problem is difficult enough to make it necessary to specify several subgoals in the effort to guide search. Korf classifies a set of subgoals in terms of how the members interact with each other. These interactions define a problem's complexity in terms of its subgoals. Fig. 6 summarizes Korf's hierarchy.

Independent subgoals are the rare ideal case—progress toward one has no effect on another. Korf defines independent subgoals in terms of the distance

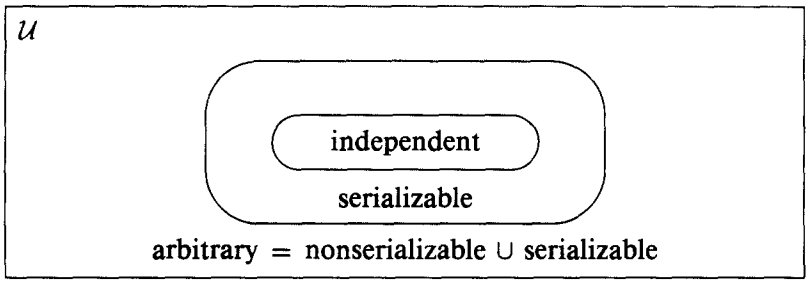

Fig. 6. Korf's hierarchy of subgoal collections.

between two states, $d(u,v)$, which denotes the length of the shortest path from u to v. This primitive distance function allows definition of the distance between two subgoals U and V with the equation

$$D(U,V) = \max_{u \in U} \min_{v \in V} d(u,v).$$

Using $D(U,V)$, Korf defines and motivates independent subgoals with the following statements.

> A collection of subgoals are *independent* if each operator only changes the distance to a single subgoal. ...One of the important properties of independent subgoals, which is clear from the definition, is that an optimal global solution can be achieved by simply concatenating together optimal solutions to the individual subproblems in any order. [17, p. 71]

Solving a single independent subgoal might be nontrivial, but the complexity of problems with independent subgoals increases only linearly with the number of subgoals. Korf defines serializable subgoals, those that do interact in a limited manner, with the following statements.

> We define a set of subgoals to be *serializable* if there exists an ordering among the subgoals such that the subgoals can always be solved sequentially without ever violating a previously solved subgoal in the order. [17, p. 71]

Thus, serializability means that for every state in the intersection of the first n subgoals there exists a path to a state in the $(n+1)$st subgoal that lies wholly within the intersection. Since these paths are ways to reach later subgoals without interfering with those previously achieved, the complexity of problems with serializable subgoals is linear, with the number of subgoals, if the subgoals are solved in the correct order. Each subgoal only has to be established once. Using the wrong order can lead to exponential complexity because solving a set of subgoals in the wrong order can require having to violate and reestablish a subgoal an exponential number of times.

Korf labels sets of subgoals with no serializable orderings as nonserializable:

> It is often the case that given a collection of subgoals, previously satisfied

> subgoals must be violated in order to make further progress towards the main goal, regardless of the solution order. Such a collection of subgoals will be called *non-serializable.* [17, pp. 72–73]

Since nonserializable subgoals may need to be violated and reestablished many times, they offer little guidance to a planner: solution time will likely rise superlinearly with the number of subgoals.

This completes our review of Korf's subgoal hierarchy. We will be using this hierarchy to analyze our planners' performances in various domains, but first we observe several limitations. First, while it may be possible to determine if a set of subgoals is independent, little work has been done on the problem of determining that a set of subgoals is serializable and finding the order [3,4,15]. The obvious method for verifying the serializability of a set of subgoals is harder than simply solving the problem without subgoals. Second, the knowledge that a set of subgoals is serializable just indicates that there exists an order such that they can be solved monotonically, but provides no guidance in the task of finding the order. Third, the knowledge that a set $\mathcal{U}$ of subgoals is serializable says little about the properties of *subsets* of $\mathcal{U}$. We make this precise with

Proposition 3.2. *Let $\mathcal{U}$ be a set of subgoals and let $\mathcal{V} \subseteq \mathcal{U}$.*

(1) *If $\mathcal{U}$ is independent, then $\mathcal{V}$ is independent.*
(2) *If $\mathcal{U}$ is serializable but not independent, then $\mathcal{V}$ may be independent, serializable, or nonserializable.*
(3) *If $\mathcal{U}$ is nonserializable, then $\mathcal{V}$ may be independent, serializable, or nonserializable.*

In some sense, the only surprising aspect of this result is that a subset of a set of serializable subgoals may be nonserializable. In fact, the proof of this is due to an observation of Korf's [17] regarding the Sussman Anomaly. He showed that the goal set {(on A B), (on B C)} is nonserializable (in the space of world-states), but the superset {(on A B), (on B C), (on C Table)} is serializable.[7]

Our experiments showed that Korf's hierarchy is by no means complete—there are a number of interesting classes of subgoals between independent collections and arbitrary serializable collections. Recall that a set of subgoals is independent when progress towards one subgoal implies that the distance towards all others is unchanged; this is an extremely restricted definition. In the effort to provide a more refined taxonomy, we define the following term.

Definition 3.3. A set of subgoals is *trivially serializable* if the subgoals can be solved in any order without ever violating a previously solved subgoal.

We note the following important properties:

[7] When mapped into the search space of partially ordered plans, both {(on A B), (on B C)} and its superset are serializable.

Proposition 3.4. *Let $\mathcal{U}$ be a set of subgoals.*
(1) *If $\mathcal{U}$ is trivially serializable and $\mathcal{V} \subseteq \mathcal{U}$, then $\mathcal{V}$ is trivially serializable.*
(2) *If $\mathcal{U}$ is independent, then $\mathcal{U}$ is trivially serializable.*
(3) *The converse of property* (2) *does not hold.*

Trivial serializability is more general than independence because it is strictly a topological property while independence is metric (i.e., defined in terms of a distance function). In particular, two trivially serializable subgoals are not independent if some operator helps to achieve both of them. Of course there is a price for the extra generality—trivial serializability does not carry the compositional properties that are entailed by independence. In particular, solutions to the separate subgoals cannot be concatenated to achieve a solution to the conjunct. Nevertheless, trivial serializability is much more common than independence and it appears to make the complexity of planning close to linear in the number of subgoals.

Just as trivially serialized subgoals represent an ideal collection, it is natural to consider collections of subgoals that are pathological while still being serializable. For such a collection of subgoals, it is difficult to *determine* the correct order. Solving problems with *laboriously serializable* subgoals, without prior knowledge of the order, may take time exponential in the number of subgoals.

Definition 3.5. A set of n subgoals is *laboriously serializable* if there exists at least one serializable ordering, yet at least $1/n$ of the subgoal orders can not be solved sequentially without possibly violating a previously solved subgoal.

Even if the majority of subgoal orderings are fine, the exponential cost of backtracking on the few pathological cases will dominate average planning time. After all, when a planner chooses a bad subgoal ordering, the result is effective nonserializability—the planner will be forced to repeatedly resolve the subgoals listed early in the ordering as subsequent subgoals induce backtracking. Since bad orderings can require exponentially more time than good orderings, tractability requires that the number of bad orderings be exponentially decreasing in the number of orderings. But if $1/n$ (or any only polynomially decreasing percentage) of the orderings are bad, then intractability will dominate.

3.3. Experiments with independent subgoals

To test our algorithms on problems consisting of independent subgoals we created a domain, called D^0S^1, with fifteen operators. A template for such an operator is illustrated below. In general, we named our domains D^xS^y because they contain x entries in each operator's delete set and it takes y steps to achieve a goal.

```
(defoperator :action Ai :precond {Ii} :add {Gi}
             :delete {}).
```

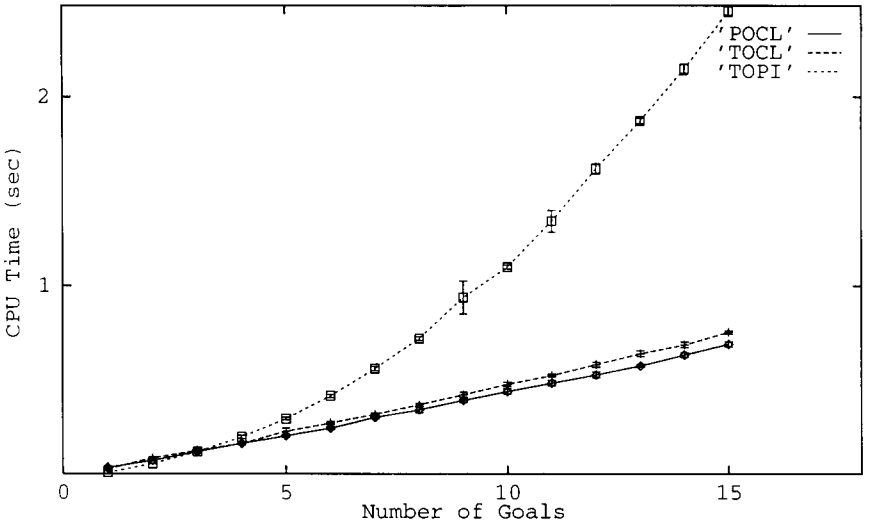

Fig. 7. The causal-link algorithms appear to exhibit linear-time complexity when given problems consisting of independent subgoals (as with D^0S^1), but the prior-insertion algorithm requires time that appears quadratic in the number of goals.

Note that each operator adds a different goal condition G_i when its individual initial condition I_i is present. The operators are independent—neither preconditions nor add lists overlap, and every operator's delete set is empty. As a result, both the order in which a problem's goal conditions were handled and the eventual order of the steps were irrelevant to the performance of every algorithm.

Given this domain, we generated 75 solvable problems each consisting of 15 randomly permuted initial conditions and between 1 and 15 randomly selected and permuted goal conditions such that for each number of goal conditions, 5 problems were generated. Each problem was given to all three algorithms; the results are shown in Fig. 7. Each point on the graph represents the average of five random tests with that number of goal conditions; 90% confidence intervals are included, but are often too small to discern. Performance was measured in seconds of Dec 5000 CPU time.

It is clear from the graph that both causal-link planning algorithms have close to linear time complexity in the number of goals[8] for this unconstrained domain, and that the prior-insertion algorithm has close to quadratic time complexity. Graphs showing the number of incomplete plans created during the search were all linear. *TOPI*'s quadratic performance was caused by its termination step. It must be noted that the performance shown depends on the fact that only solvable problems were generated. While *POCL* would have

[8] In this discussion and in subsequent analyses, we assume that each planner can successfully solve individual subgoals in a fixed amount of time. In other words, we describe performance in terms of the complexity of integrating the solutions to the subgoals, assuming that the cost of solving these isolated subgoals is fixed.

quickly quit attempting to achieve an impossible goal, the total-order algorithms might have explored an exponential number of plans in a futile attempt to find a satisfactory order. Similarly, the performance of the total-order planners depends on the use of a bounded depth-first search strategy.

3.4. Experiments with serializable subgoals

Our investigation of serializable subgoals consisted of two domains, D^mS^1 and D^1S^1, with large and small delete sets respectively. Elements of delete sets cause steps in each domain to interact, however detecting the extent of interaction is easier in the first domain than the second. This caused *TOCL* to perform as well as *POCL* in the first domain, D^mS^1, while it degraded terribly in the harder D^1S^1. We show that each planner performs well if the domain is trivially serializable and poorly otherwise.

3.4.1. Goal interactions are manifest in D^mS^1

The D^mS^1 domain resembles D^0S^1 except that the temporal order of plan steps is tightly constrained by the delete sets of each operator. The D^m part of D^mS^1 signifies that there are *many* entries in each operator's delete set. A template for an operator is illustrated below.

```
(defoperator :action A_i :precond {I_i} :add {G_i}
             :delete {I_j | j < i}).
```

Note that operator A_i deletes the preconditions of operators A_j for all j less than i. This implies that for any set of goals, there exists a single ordering of steps that will achieve that set of goals. The reason is illustrated in Fig. 8.

For a real world analog to this domain, consider sealing a set of differently sized boxes such that box i fits inside box $i + 1$. Once one box is sealed, all of the boxes inside of it are not accessible. The accessibility of a box b_i is represented by initial condition I_i, operator A_i seals the box, and its being sealed is represented by goal condition G_i. The difficulty of problems in this domain is summarized by Proposition 3.6.

Proposition 3.6. *The problems in D^mS^1 have laboriously serializable subgoals for TOPI, and trivially serializable subgoals for TOCL and POCL.*

The proof of this proposition follows directly from Lemmas 8.1 and 8.3 (see Appendix A).

We generated 75 solvable problems in the same fashion as the previous experiment: 5 random problems for each number of goal conditions between 1 and 15. Fig. 9 illustrates the results of this experiment. Both causal-link planners appear to have linear time complexity, but *TOPI* was incapable of solving even moderately sized problems before the resource cutoff. We attribute this difference to the fact that the domain is laboriously serializable for *TOPI*, but trivially serializable for the causal-link planners.

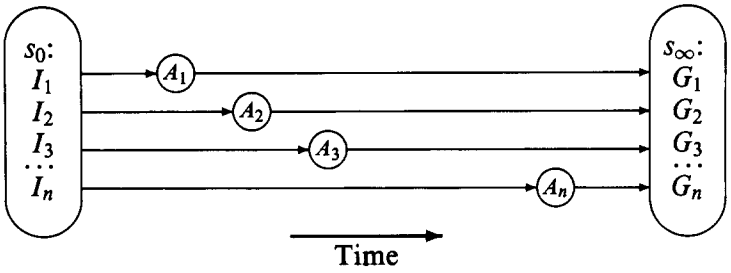

Fig. 8. The causal structure of solutions to problems in D^mS^1 and D^1S^1. Time progresses to the right and each step deletes the preconditions of all steps above it or the immediate step above it respectively.

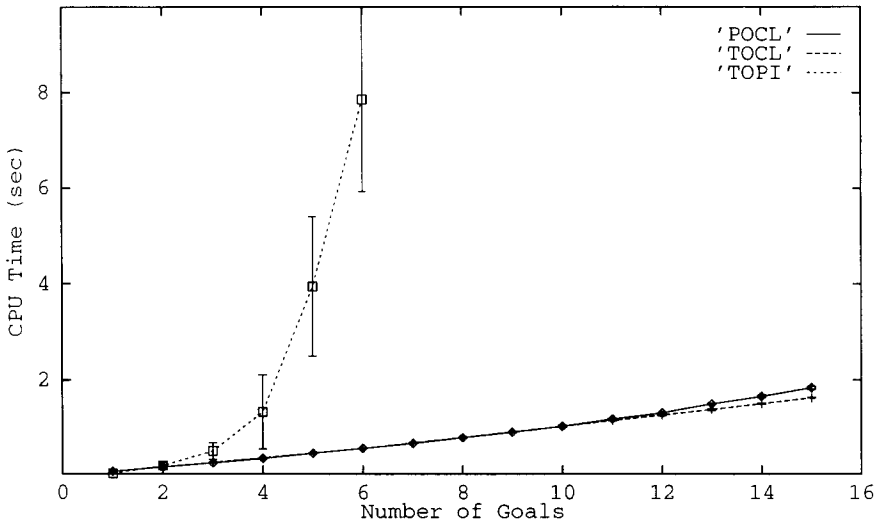

Fig. 9. A total-order planner can excel in a domain, such as D^mS^1, in which tight ordering constraints result in trivial serializability.

3.4.2. *Goal interactions are more subtle in D^1S^1*

To show the effect of the arbitrary ordering decisions made by *TOCL*'s linearization step we created a domain D^1S^1 similar to D^mS^1. Operators in D^1S^1 contain only one entry in their `:delete` fields, but that entry makes solutions to problems in D^1S^1 identical to those in D^mS^1. However, this successful ordering can only be discovered by looking at numerous steps together and considering their combined constraints. In contrast, the redundant deletes in D^mS^1 made the correct placement of a step clear in isolation. In Fig. 8, where a step in D^mS^1 deleted all of the preconditions of steps above it, a step in D^1S^1 only deletes the precondition of the step immediately above it.

```
(defoperator :action A_i :precond {I_i} :add {G_i}
             :delete {I_{i-1}}).
```

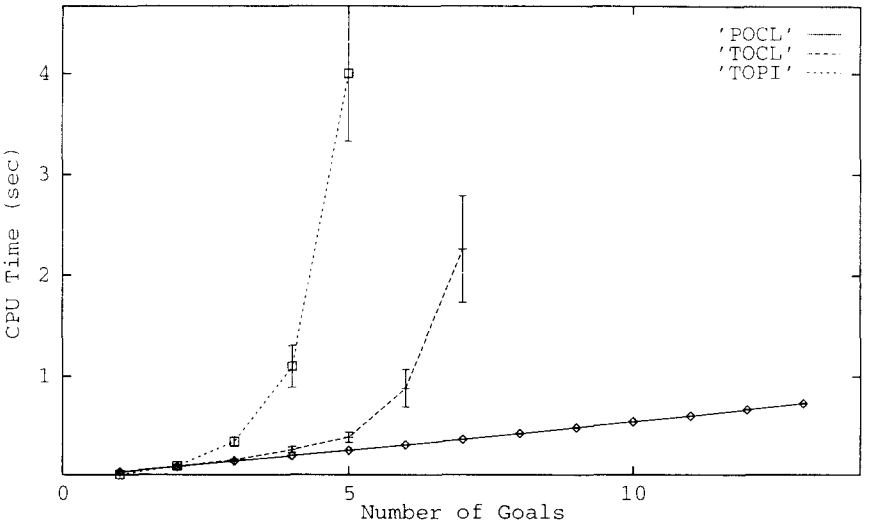

Fig. 10. Some domains, such as D^1S^1, give all total-order planners problems, but are easily solved by partial-order planners.

The blocks world is a familiar analog to this artificial domain. Consider building a tower of N blocks from an initial state where all blocks are on the table. The initial condition I_i represents that block b_i is clear, action A_i puts block b_i on top of block b_{i-1}, and the goal G_i represents that block b_i is on top of block b_{i-1}.

Proposition 3.7 shows that eliminating the redundant delete constraints makes this domain considerably harder for *TOCL*, transforming it from trivially to laboriously serializable:

Proposition 3.7. *The problems in D^1S^1 have laboriously serializable subgoals for TOPI and TOCL. They have trivially serializable subgoals for POCL.*

The proof of this proposition follows directly from Lemmas 8.1, 8.4, and 8.5 in Appendix A.

We generated 390 solvable problems in the same fashion as the earlier experiments: 30 random problems for each number of goal conditions between 1 and 13. As shown in Fig. 10, *POCL* maintained its near-linear performance, while both total-order planners exhibited apparently exponential time complexity. Because this domain is laboriously serializable for *TOCL*, the algorithm branched intractably when considering arbitrary ordering constraints between steps A_{i-1} and A_{i+1} before adding step A_i, which gives the correct ordering constraint.

3.5. Experiments with nonserializable subgoals

In this section we explore the performance of our algorithms on three domains, all of which are nonserializable when considered in terms of either the space of world-states or the search space of *TOPI*, but are serializable for

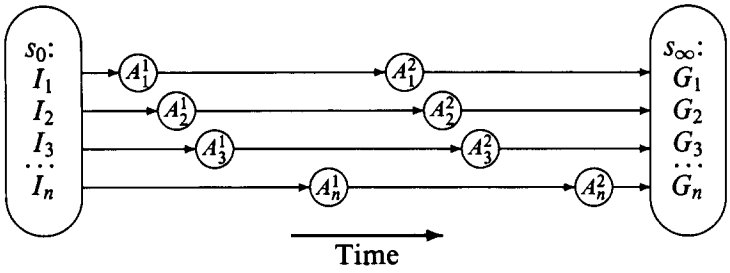

Fig. 11. The causal structure of solutions to problems in D^1S^2 and D^mS^2.

the causal-link planners. The first two are readily solved by *POCL*, but the last one is more difficult.

3.5.1. *POCL finds D^mS^2 and D^1S^2 trivially serializable*

To experiment with nonserializable subgoals we created two different domains, D^mS^2 and D^1S^2, by modifying D^mS^1 and D^1S^1. The difference, of course, lies in the S^2 superscript which signifies that subgoals require subplans of length 2 unlike the singleton subplans of the S^1 domains. The pattern of the delete sets of the new domains force the planners to interleave the steps introduced for each subgoal, and (as in the previous section) determining the correct ordering is easier for D^m due to the redundant constraints.

As a concrete example of the domains, templates for the operators needed to achieve goal condition G_i in domain D^mS^2 are shown below.

(defoperator :action A_i^1 :precond $\{I_i\}$:add $\{P_i\}$
:delete $\{I_j \mid j < i\}$).

(defoperator :action A_i^2 :precond $\{P_i\}$:add $\{G_i\}$
:delete $\{I_j \mid \forall j\} \cup \{P_j \mid j < i\}$).

A step in D^mS^2 deletes the preconditions of all prior steps, while a step in D^1S^2 deletes the preconditions of the only the immediately prior step. Templates for the operators needed to achieve goal condition G_i in domain D^1S^2 are shown below.

(defoperator :action A_i^1 :precond $\{I_i\}$:add $\{P_i\}$
:delete $\{I_{i-1}\}$).

(defoperator :action A_i^2 :precond $\{P_i\}$:add $\{G_i\}$
:delete $\{I_j \mid \forall j\} \cup \{P_{i-1}\}$).

Thus, there exists a single ordering of steps that will achieve a set of goal conditions. The causal structure of a solution to a problem in these domains appears in Fig. 11.

For each domain we generated 120 problems. Each problem type consisted of 16 initial conditions and between 1 and 8 goal conditions. Fifteen problems

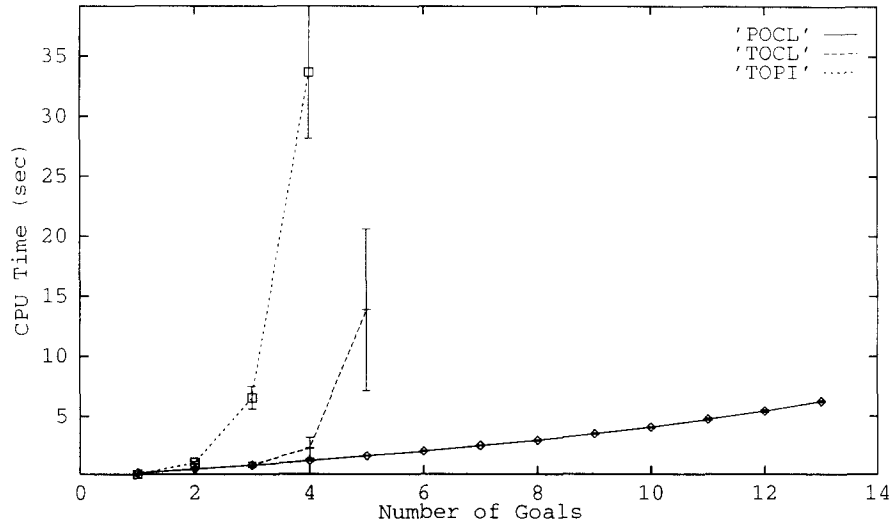

Fig. 12. In D^1S^2 *POCL* outperformed all other planners.

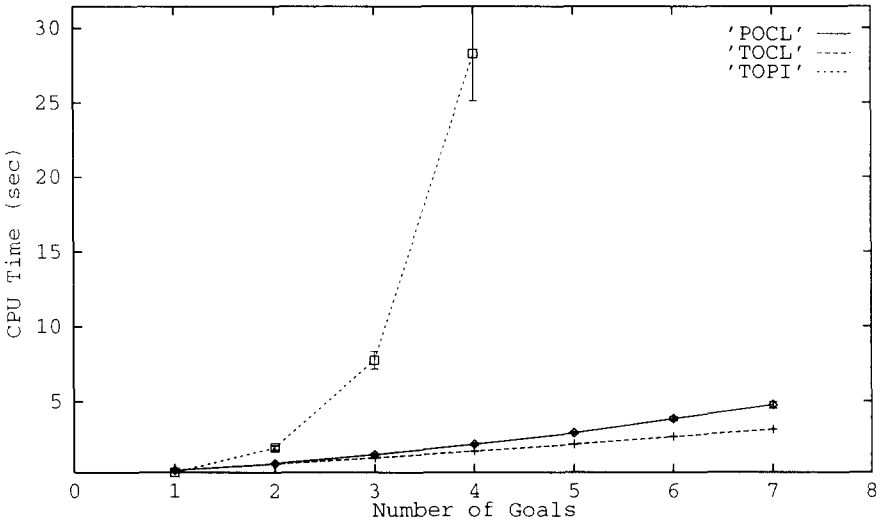

Fig. 13. Solving problems in D^mS^2 was easy for both causal-link algorithms.

were generated for each problem type. As in previous sections, the initial conditions and goal conditions were randomly permuted, and performance was measured in seconds of Dec 5000 CPU time. The difficulties of these domains are summarized in Propositions 3.8 and 3.9, and the results of the experiments appear in Figs. 12 and 13.

Proposition 3.8. *The problems in D^1S^2 have subgoals that are nonserializable for TOPI, laboriously serializable for TOCL, and trivially serializable for POCL.*

The proof is derived simply from Lemmas 8.2, 8.4, and 8.5 in Appendix A.

Proposition 3.9. *The problems in D^mS^2 have subgoals that are nonserializable for TOPI and trivially serializable for TOCL and POCL.*

The proof follows directly from Lemmas 8.2 and 8.3 in Appendix A.

3.5.2. Ordering decisions in D^mS^2* are difficult

So far, in all our experiments, the set of subgoals was trivially serializable for *POCL*. In order to create a harder domain for *POCL* we created D^mS^{2*}. The subgoals in this domain are laboriously serializable for *POCL*. Once again, there is only one way to achieve a subgoal, and finding it is trivial. The problem comes in fitting the solutions together. There are two types of subgoals. The first is specified by G_i and templates to achieve this type appear below.

(defoperator :action A_i^1 :precond $\{I_i\}$:add $\{P_i\}$
:delete $\{P_j \mid j < i\}$).

(defoperator :action A_i^2 :precond $\{P_i\}$:add $\{G_i\}$
:delete $\{P_j \mid j < i\}$).

There are an exponential number of solutions to problems that solely consist of subgoals like G_i, but the number reduces to one when subgoal G_* is included in the problem. Solving subgoal G_* only requires an instance of the following operator.

(defoperator :action A_* :precond $\{I_*\}$:add $\{G_*\}$
:delete $\{I_i \mid \forall i\} \cup \{G_i \mid \forall i\}$).

The causal structure of the solution to a problem in D^mS^{2*} appears in Fig. 14. The step A_* affects all the causal links that do not appear directly below it, and the steps A_i^1 and A_i^2 affect the middle link of the causal chains above them. The difficulty of this domain is summarized in the following proposition:

Proposition 3.10. *The problems in D^mS^{2*} have subgoals that are nonserializable for TOPI and laboriously serializable for POCL and TOCL.*

The proof follows from Lemmas 8.2 and 8.6 in Appendix A.

Given this domain we generated 60 solvable problems in the same fashion as the earlier experiments: 20 randomly generated problems for each number of goal conditions between 1 and 6. Fig. 15 plots the results: all planners exhibited exponential degradation when confronted with more goal conjuncts; this confirms the expectation that laborious serializability results in intractability. When solving problems with 6 goals, the mean performances of *TOCL* and *POCL* were quite variable, resulting in large 90% confidence intervals. This was caused by the fact that in a third of the problems the planners were

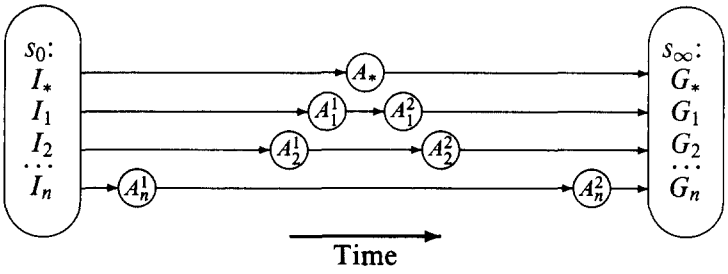

Fig. 14. The causal structure of solutions to problems in $D^m S^{2*}$.

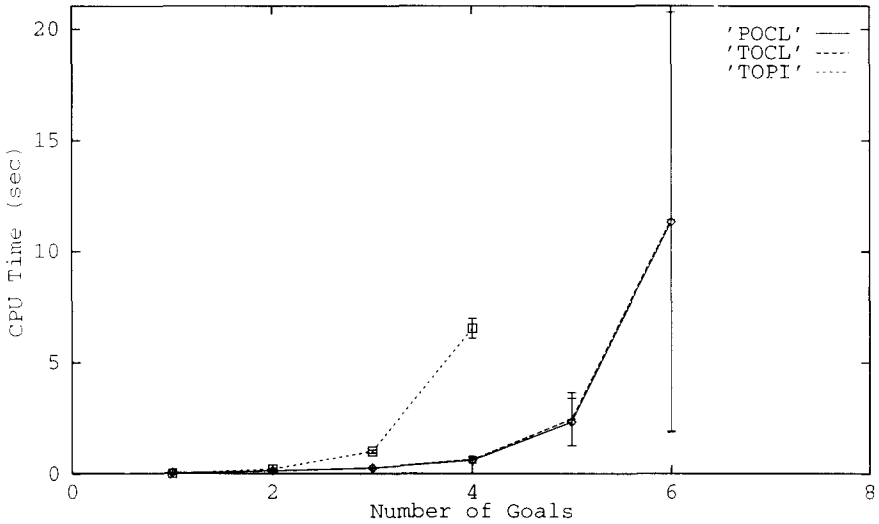

Fig. 15. Some simple domains, such as $D^m S^{2*}$, give all planners problems.

lucky, chanced upon a good serialization ordering, and thus took only a linear amount of time. For the majority of problems, however, the planners required exponential time which dominated the average problem solving time. See the proof of Lemma 8.6 for further elaboration.

3.6. Experiments with operator selection decisions

The domains considered in previous sections are much simpler than those encountered in many real planning problems because the artificial domains provided only one way to achieve each subgoal. This meant that step-ordering decisions were the only source of combinatorial search since the operator selection decision was always trivial. Because the complexity due to operator selection is important for most planning tasks, this section explores the interaction between the selection and ordering decisions in the three planners. To make this analysis, we introduce a general transformation Θ_n for the previously defined domains and scrutinize several illuminating instances.

We illustrate this transformation by taking D^mS^1 and constructing the new domain $\Theta_2 D^mS^1$. Our construction begins by taking each operator in D^mS^1 (Section 3.4.1) and creating the following two[9] operators for the new domain by adding the precondition P_α or P_β.

(defoperator :action A_i^α :precond $\{I_i, P_\alpha\}$:add $\{G_i\}$
:delete $\{I_j \mid j < i\}$).

(defoperator :action A_i^β :precond $\{I_i, P_\beta\}$:add $\{G_i\}$
:delete $\{I_j \mid j < i\}$).

When the terms P_α and P_β are in a problem's initial conditions, there are two ways to achieve any subgoal G_i. One uses operator A_i^α and the other uses operator A_i^β. The construction is completed by adding the following operator A_α to the domain.

(defoperator :action A_α :precond $\{\}$:add $\{G_\alpha\}$
:delete $\{P_\beta\} \cup \{G_i \mid \forall i\}$).

Problems with goal G_α require adding a step of type A_α to the plan, but this step threatens the causal link $A_i^x \xrightarrow{G_i} s_\infty$ which was created while achieving a goal G_i with either step A_i^α or A_i^β. This threat can be resolved by ordering step A_α before step A_i^x, but when x is β, A_α also threatens the causal link $s_0 \xrightarrow{P_\beta} A_i^\beta$. This latter threat cannot be resolved. Thus, complete solutions cannot contain A_i^β steps, but no planner can determine this until it plans for goal G_α. This means that any serializable ordering in the $\Theta_2 D^mS^1$ domain must begin with the subgoal for G_α. The following proposition describes the general case:

Proposition 3.11. *For POCL and TOCL, the ratio of orderings that are serializable for an M goal problem in a Θ_n-transformed domain ($n \geqslant 2$) is R/M, where R is the ratio of orderings that are serializable for an $M-1$ goal problem in the original domain.*

This proposition has interesting consequences:

Corollary 3.12. *For POCL and TOCL, problems with two or more subgoals are laboriously serializable in any domain which has been transformed by Θ_n (for $n \geqslant 2$).*

The corollary makes intuitive sense: since Θ_n domains have multiple ways to achieve each subgoal and the different methods interfere, backtracking is required for most subgoal orderings. The proof is straightforward. Since R can never exceed 1 and $(M-1)/M \geqslant 1/M$ for $M \geqslant 2$, the corollary follows directly from Proposition 3.11 and Definition 3.5.

[9] To increase the complexity of step selection further one can replace with an arbitrary number of new operators, but setting $n = 2$ suffices to complicate planning by an exponential factor.

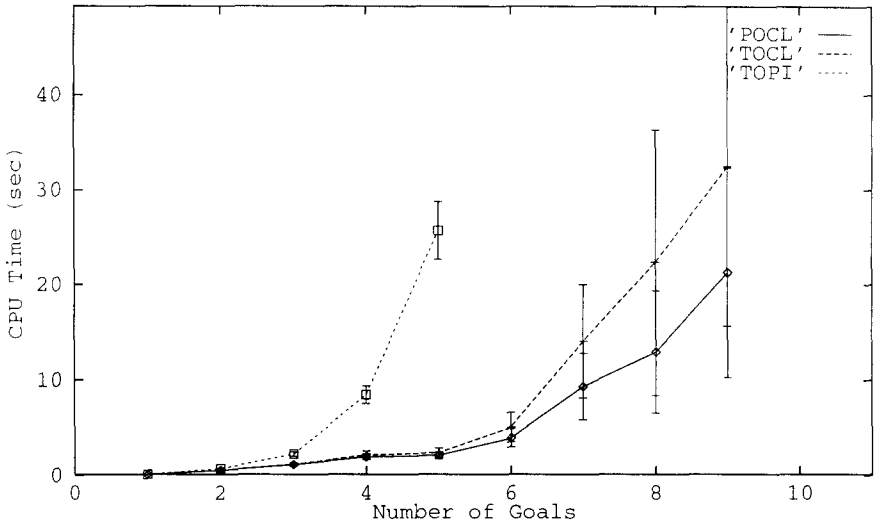

Fig. 16. In domain $\Theta_2 D^m S^1$, *TOCL*'s performance mimicked that of *POCL*.

Since *TOPI* does not use causal links, the above arguments do not directly apply. Still, just like *POCL* and *TOCL*, *TOPI* cannot know which step to use in achieving a subgoal G_i until it achieves the subgoal for G_α. At this point we note that G_α has to be achieved last to make step A_α appear first in the plan. This leads to Proposition 3.13.

Proposition 3.13. *For TOPI, all problems in Θ_n transformed domains have nonserializable subgoals.*

Empirically, we explore the interaction of step-ordering decisions with operator selection decisions by constructing the domains $\Theta_2 D^m S^1$ and $\Theta_2 D^0 S^1$ from $D^m S^1$ and $D^0 S^1$ respectively. For each of these domains we generated 300 solvable problems. Each problem type consisted of 17 initial conditions, and between 1 and 10 goal conditions. Performance was measured in seconds of Dec 5000 CPU time. The results are shown in Figs. 16 and 17. Each data point represents a planner's average performance over 30 randomly generated problems. The 90% confidence intervals show how much the performance varied from one problem to another.

In the $\Theta_2 D^m S^1$ domain, only one step ordering is legal, and *TOCL* could infer this ordering early in the planning process. As a result, *TOCL* avoided pointless backtracking over equivalent step-ordering decisions and exhibited the same performance as *POCL* in this domain.

In the $\Theta_2 D^0 S^1$ domain, any step ordering beginning with A_α is legal. Since the placement of A_α is easily determined from all of its delete conditions, the difficulty of these problems is solely caused by the operator selection decisions. The main lesson learned from this experiment is that arbitrary step-ordering decisions interact with arbitrary operator selection decisions. All

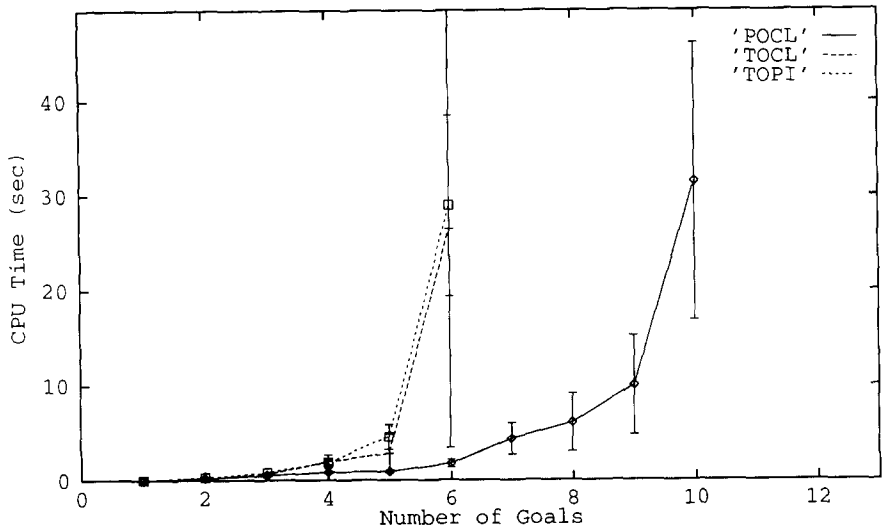

Fig. 17. In domain $\Theta_2 D^0 S^1$, *TOCL* took as long as *TOPI*.

planners exhibited exponential performance, but the early commitment on step ordering resulted in a higher branching factor for the total-order planners. Although the order of the steps was immaterial to the success or failure of the plan under consideration, more flawed plans were considered on average leading to poor performance.

These experiments are interesting because the Θ_2 transformation had different effects on the relative performance of the planners in the two cases. Although *POCL* and *TOCL* performed equally in D^0S^1 and D^mS^1, their performance differed in the Θ_2 derivatives. This shows that a partial-order planner can have a performance advantage which is solely due to operator selection issues.

The phenomena can be explained by realizing that the Θ_2 domains cause both planners to make mistakes and eventually backtrack. But since each *POCL* plans corresponds to many totally ordered plans, *TOCL* requires more search to regain the path after each mistake.

3.7. Experiments with heterogeneous sets of subgoals

Until this point we have been concentrating on problems that contain sets of related subgoals. In order to explore how the difficulty of a problem behaves when it contains two unrelated sets of subgoals, we constructed a test that contained subgoals from D^mS^{2*} and D^0S^1.

In this domain we generated 1620 solvable problems. Each problem type consisted of 12 initial conditions, between 0 and 8 D^0S^1 goal conditions, and between 0 and 5 D^mS^{2*} goal conditions. Thirty problems were generated for each problem type. Each problem had its initial and goal conditions randomly permuted. Performance was measured in seconds of Dec 5000 CPU time.

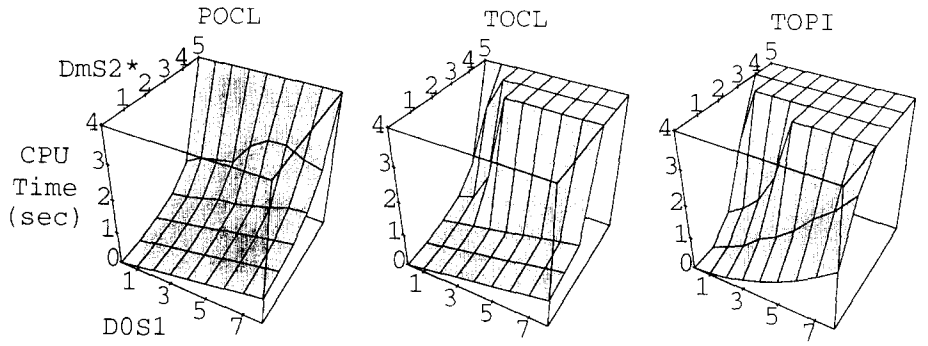

Fig. 18. For total-order planners the difficulty of solving a problem can rise exponentially with the number of independent subgoals.

The results are shown in Fig. 18.

The main lesson learned from this experiment is that the difficulty of solving problems with heterogeneous subgoals is not simply additive because the number of plan-states which need violated subgoals to reach a solution can increase dramatically when combining different sets of subgoals. The most illustrative example of this occurs in the graph for *TOCL* where the number of D^mS^{2*} goals is held at 3 and the number of D^0S^1 goals varies from 0 to 8. The complexity appears to rise exponentially with the number of independent goals.

The behavior of *TOCL* can be explained by noting that there are two plan-states, $[A_1^1, A_1^2, A_2^1, A_2^2]$ and $[A_2^1, A_2^2, A_1^1, A_1^2]$, in three goal problems of D^mS^{2*} that cannot be modified into a solution without violating previous subgoals. The number of such plan-states rises exponentially with the number of independent goals. *TOPI* has a similar problem. In this example, *POCL* avoids the problem because the number of such minimal plan-states only rises linearly with the number of independent goals. This is not the case in general.

3.8. Application to real domains

Previously, we argued that real world domains typically include many different types of subgoal interaction which impedes any understanding of the source of computational intractability. By restricting our attention to artificial domains, we've teased apart the different aspects of domain complexity and compared the scaling properties of different planning algorithms. However, our theory can also be applied to more complex domains as we now illustrate.

We focus on the *Tyre world* domain which encodes repair actions on a British automobile. While not completely "real", Tyre world's 14 operators make it fairly complex. We selected the domain for several reasons. First, it was written independently by Stuart Russell at Berkeley and thus represented an independent test for our theory. Second, Russell had posed a difficult planning problem for the domain (replacing a flat tire by jacking the wheel, unbolting the lugs, etc., eventually restoring all tools to the trunk) whose

Table 2
Summary of the three experiments with *Tyre world* problems. Times are in CPU seconds; the ">" appears when the mean includes time spent planning before failure induced by resource cutoff

Serializations for	*POCL*		*TOCL*	
	Completed	Mean time	Completed	Mean time
TOCL & POCL	100%	123	100%	863
Just *POCL*	100%	102	50%	> 6769
Neither	0%	> 10323	0%	> 11594

optimal solution required 19 steps and took six hours to solve (e.g., required exploring 3 million partial plans) even when using extremely efficient search techniques (in fact, Russell invented the problem to test his bounded-memory IE search technique [28]). We took as our challenge, the problem of rendering this problem tractable.

Our theory predicted that the eight subgoals of the problem were nonserializable for *TOPI* yet laboriously serializable for *POCL* and *TOCL*. Concentrating on the causal-link planners, we noted four constraints on subgoal ordering that would render the problem trivially serializable for *POCL*.[10] Analysis showed that only 9408/40320 = 23.3% of the possible orderings denoted correct serializations for *POCL*. Two extra constraints were needed to render the problem trivially serializable for *TOCL* so only 1576/40320 = 3.91% of all orderings were correct serializations for *TOCL*

To test if our theory of subgoal interactions could lead to significant performance improvement in this domain, we generated three sets of planning problems. All planning problems encoded Russell's tire changing task with the same eight goal conjuncts—the only difference was the order of the conjuncts. The first ten problem were randomly chosen serializations for both *TOCL* and *POCL*. The second ten problems were serializations for *POCL* but not for *TOCL*. Finally, the last ten problems weren't serializations for either planner. As Table 2 shows, the performance we measured corresponds perfectly with our theory.

These experiments suggest that our theory of subgoal interactions *does* provide useful insight about the performance of planners on real world domains. When given a serializable ordering, both partial- and total-order planners exhibit comparable performance. The challenge is finding a serialization ordering. A partial-order planner's ability to delay step-ordering decisions often increases the number of serializable orderings, sometimes quite drastically (i.e., trivial serializability).

[10] For example, three conditions force a planner to generate steps that use a tool prior to considering goals to put that tool away (e.g., the planner must consider how it will inflate the spare before determining how and when the pump will be put in the trunk). Space restrictions preclude a detailed description of this domain and our experiments, but the complete encoding, source code for the planners, and our data is available by anonymous FTP. Send mail to `bug-snlp@cs.washington.edu` for details.

4. Discussion

One of the major advances of Sacerdoti's NOAH [29] was the shift from searching through a space of world-states to searching through a space of partially ordered plan-states. This paper analyzed that contribution by comparing the performance of three different planning algorithms on a variety of domains. The first planner, *POCL*, delays the ordering of steps in a plan until they interact in a way that affects the plan's correctness. The second planner, *TOCL*, is a modification of *POCL* that restricts generated plans to contain totally ordered steps. This restriction makes *TOCL* add premature step-ordering constraints that *POCL* avoids.

Although the first two planners add steps anywhere in a plan, the third planner, *TOPI*, only adds steps prior to existing steps. This last algorithm is radically different from the previous two: although it uses plan-states, it structures the search space in a way that makes it equivalent to a backward chaining world-state search such as the regression planner of Nilsson [25].

The performance of each algorithm is determined by the number of plan-states that it visits to find a solution and the complexity of visiting each plan-state. We first compared the three algorithms in terms of the complexity of visiting a plan-state. *TOCL* is the most efficient at visiting a plan-state: its cost is linear in the number of existing steps. The use of partial orderings made *POCL*'s cost per step be quadratic in the number of steps. Finally, *TOPI*'s complexity did not depend on the number of steps, but it was exponential in terms of the number of open goal conditions (assuming $P \neq NP$). This cost applies to any backward chaining state-space search that uses a least-commitment binding strategy for variables.

To compare the number of plan-states each planner visited, we had to classify the difficulty of problems for the various planners. We made this characterization by considering the set of subgoals specified by a problem's goal conjuncts, and classifying that set in Korf's subgoal hierarchy. Since Korf's subgoal hierarchy is overly general, we extended it to include the trivial and laborious subclasses of serializable subgoals (Fig. 19). Solving problems with independent or trivially serializable subgoals required visiting a number of plan-states that was linear in the number of subgoals, but problems with laboriously serializable or nonserializable subgoals required visiting an exponential number of plan-states.

To explore this refined hierarchy, we generated several artificial domains and compared the different planners. We observed that the partial-order planner never took significantly longer than either total-order planner, and sometimes the partial-order planner performed much better. In all cases the planners only performed well when given a problem with trivially serializable subgoals. With one simplifying assumption this result can be proven true:

Proposition 4.1. *Assuming that a problem's subgoals can be achieved in constant time, the expected time to solve a problem rises linearly with the number of*

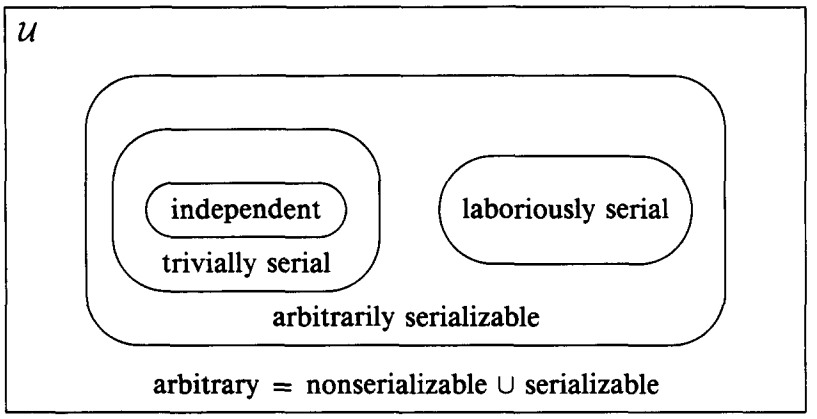

Fig. 19. An extended hierarchy of subgoal collections.

subgoals if the problem is trivially serializable, but rises exponentially if the problem is laboriously serializable or nonserializable.

The proof follows easily from the fact that a subgoal only has to be achieved once given a serialization ordering, but may have to be solved an exponential number of times when given some other ordering.

A close examination of our experiments reveals even more interesting regularities as summarized in Table 3. Notice that the difficulty of a set of subgoals was never harder for *POCL* than it was for the total-order planners, and that it was always hardest for *TOPI*.

Attempting to prove these observations are guaranteed leads us to Proposition 4.2 which implies that solving a set of subgoals is never more difficult for *POCL* than it is for *TOCL* (our experiments show that it is often much easier).

Table 3
Summary of the subgoal classifications of each domain for each planner. Subgoals are independent (I), trivially serializable (T), laboriously serializable (L), or nonserializable (N)

	Algorithm		
Domain	*POCL*	*TOCL*	*TOPI*
D^0S^1	I	I	I
$\Theta_2 D^0 S^1$	L	L	N
D^1S^1	T	L	L
D^mS^1	T	T	L
$\Theta_2 D^m S^1$	L	L	N
D^1S^2	T	L	N
D^mS^2	T	T	N
D^mS^{2*}	L	L	N
Tyre	L	L	N

Proposition 4.2. *Any serializable subgoal ordering for TOCL is also a serializable subgoal ordering for POCL.*

The proof follows from Lemma 8.8 in Section 8.2.

Perhaps surprisingly, a similar domination result for *POCL* and *TOPI* cannot be made. There exist problems with subgoals that are trivially serializable for *TOPI* and laboriously serializable for the causal-link algorithms. For example, consider the following domain which we call $D^*S^1C^2$:

(defoperator :action A_i^1 :precond $\{I_i\}$:add $\{G_i\}$
:delete $\{G_*\}$)

(defoperator :action A_i^2 :precond $\{I_i\}$:add $\{G_i\}$
:delete $\{\}$)

Problems in $D^*S^1C^2$ have initial conditions $\{G_*, I_1, \ldots, I_n\}$ and goals $\{G_*, G_1, \ldots, G_n\}$. For the causal-link algorithms, all serializable subgoal orderings have to start with the subgoal for G_*. For *TOPI*, the operator selection step ensures that any subgoal ordering is a serializable ordering.

Finally, although one might expect that the partial-order representation provides no benefit when planning complexity results from the need to choose which operator should be used to achieve open conditions, we showed that this intuition is false. In some domains, a partial-order planner can rule out bad combinations of operator selection decisions more efficiently than can a total-order planner. In our last experiment, we explored the interaction of different sets of subgoals. We discovered that even though two sets are independent with respect to each other, solving problems with both sets together is harder than solving each set separately.

5. Related work

In previous work [2,30] we reported on preliminary experiments regarding the effect of step-order representations on planning. Besides an increased number of experiments, this paper analyzes the results in terms of an extended version of Korf's [17] taxonomy of subgoals and domain complexity. Joslin and Roach [13] extend Korf's analysis of nonserializable subgoals[11] with a topological analysis of subgoals in terms of their connected components. Our extensions to Korf's taxonomy are independent of Joslin and Roach's contribution since they do not consider the number of viable subgoal orderings while this is the key concept underlying our notions of trivial and laborious serializability.

[11] Unfortunately, Joslin and Roach did not phrase their work in these terms, appearing unaware of Korf's work.

Besides our earlier papers, there has been little other work comparing the performance of partial-order and total-order planners. A notable exception is the excellent work of Minton et al. [22]. This paper considers the TO and UA algorithms which resemble propositional versions of our *TOCL* and *POCL* algorithms with one difference: unlike our planners, TO and UA are not systematic. Minton et al. demonstrate the existence of an isomorphism $\mathcal{L}$ which maps from nodes in the UA's search space into nonempty equivalence classes that partition the search space of TO. The existence of $\mathcal{L}$ proves that the search space of their partial-order planner is no larger than that of the total-order algorithm, and that it is possibly exponentially smaller. Systematicity and our proof of Lemma 8.8 proves that $\mathcal{L}$ also exists between *POCL* and *TOCL*. Minton et al. argue that since UA's cost to evaluate a search space node is only slightly more than that of TO, the partial-order planner should run faster. They also report on experiments that suggest that UA's advantage increases with a decreasing number of interactions between plan steps, but this is the only domain characteristics they consider. Minton et al. also show that partial-order planners can exploit certain types of heuristics more effectively than their total-order siblings; given the crucial need for heuristics to guide search through the exponential spaces of real problems, this result is of great importance. In a recent extension to their earlier work, Minton et al. [23] consider the effects of different search strategies and the distribution of solutions on performance.

6. Future work

There are several other limitations to this work that point to future research directions. We need a way to analyze the difficulty of solving a single subgoal and techniques for analyzing domains which contain subgoals of varying difficulty. Also, characterizing serializable subgoals as trivial or laborious is still too coarse. There is room for more refinement in Korf's hierarchy, especially in the critical cases where there are only a few (perhaps an exponentially decreasing percentage of) bad subgoal orderings.

Our conclusion that domains with laboriously serializable subgoals are intractable is based on the assumption that good serializations cannot be predicted before planning commences. If it were possible to construct some sort of domain theory compiler which identified good and bad orderings, then the benefits would be considerable. In fact, much of the work on abstraction in planning can be viewed as doing exactly this. Perhaps it might be possible to generalize the techniques in ALPINE [15,16] or the subgoal interaction analysis of STATIC [9] in this direction.

A major weakness in our work is its dependence on the STRIPS representation. We plan to use UCPOP [27] to explore whether partial-order representations are useful given more expressive domains, such as ADL [26], which include conditional effects and universally quantified effects. Also using UCPOP, we hope to replicate the experiments of Minton [21] to see if

explanation-based learning can speed up a partial-order planner as much as it has PRODIGY [24].

All of the experiments reported in this paper challenged planners only with solvable problems. A natural extension would be to investigate the performance of the algorithms when confronted with impossible goals. Our intuition is that the advantage of *POCL* would be amplified. Another research direction that begs for attention is consideration of other subgoal focusing mechanisms. In this paper we assumed that each subgoal was completely solved before attempting the next, but there are numerous other control strategies. For example, it would be interesting to consider iterative sampling techniques described in [18,23].

On a more basic level, it is unclear that our definition of subgoals for plan-state searches is the best one. Our work, as well as [13,17], defines subgoals using elements of Ω, but such need not be the case. Our causal-link algorithms focused on reaching subgoals, as we defined them, by using a FILO strategy for selecting open goals to resolve. Just as there are other strategies for selecting open goals, there are other ways to define a subgoal.

For example, one selection strategy prioritizes open goals based on predicate type and leads to a form of abstraction [19]. A plan-state is in a subgoal when it contains an abstract solution to a problem, and the next less abstract solution is the next subgoal. We have performed some preliminary experiments using *POCL* with different strategies for selecting open goals and noted that the difficulty of problems is strongly influenced by the strategy. More work needs to be done to relate strategies with definitions of subgoals and to characterize which strategy is best for a problem.

7. Conclusions

In this paper we have evaluated the effect of a partial-order plan representation on planning performance both empirically and in terms of the effect of representation on subgoal structure. Our paper makes several major contributions:

- We demonstrate that Korf's [17] subgoal taxonomy fails to differentiate between classes that have vastly different computational properties. In particular, we argue that some serializable sets are easy to solve while others are difficult. We conclude that the distinguishing feature is the number of feasible serialization orderings and this leads to our definitions of *trivially serializable* and *laboriously serializable* subgoals. Since all orderings of trivially serializable subgoals lead to a global solution, this class is computationally tractable. We note that pure trivial and pure laborious serializability are but two points on a continuum of arduousness and many real problems may be intermediate in difficulty.
- We present and analyze three planning algorithms on eight artificial domains, which were created to illustrate different types of subgoal inter-

actions. From the results of our experiments, we make the following observations:

(1) The total-order planners never performed significantly better than the partial-order planner.
(2) In domains with complex ordering interactions the partial-order planner performed exponentially better than either total-order planner. In particular, the D^1S^1 and D^1S^2 domains (Sections 3.4.2 and 3.5) have ordering interactions that involve numerous steps in a pairwise fashion; only *POCL* was able to deduce a successful ordering efficiently.
(3) Planning problems which involved operator selection decisions (i.e., multiple, interacting ways to achieve open conditions) were easier for a partial-order planner—even when the corresponding single-operator problems (e.g., D^0S^1 and $\Theta_2 D^0S^1$) were easy for planners with either representation.
(4) The performance difference of the planners correlates perfectly with the subgoal classification of the domains. Since the algorithms use different plan representations, they have different search spaces. Thus the natural subgoal decomposition of a D^1S^2 problem was trivially serializable for *POCL*, laboriously serializable for *TOCL* and nonserializable for *TOPI*. As predicted by Proposition 4.1, all three planners performed well only when confronted with trivially serializable subgoals.

- We prove (Proposition 4.2) that compared to *TOCL*, the partial-order *POCL* planner renders a strictly greater number of problems trivially serializable.

8. Appendix A. Proofs

There is one feature of our planners (used in all the proofs) that needs to be discussed prior to proving the lemmas of this section. All of our planners perform bounded depth-first searches. This search interacts with the subgoal classification of a problem to affect the planners' performances. For instance, solving a set of serializable subgoals in the correct order ensures that a previously solved subgoal need never be violated. This affects the depth-first search by limiting the amount of backtracking that needs to be performed. Once a plan-state in a subgoal is found, the search will never need to backtrack back through it.

We can use this feature to prove that a certain subgoal ordering is not a serializable ordering. An ordering is not a serializable ordering when a planner reaches a plan-state where it will have to backtrack and violate a previous subgoal.

8.1. *Prior-insertion algorithm (TOPI)*

As mentioned in Section 3.1, plan-states for *TOPI* are in the subgoal for a

proposition P when all of the open conditions of steps added to solve P are in the initial conditions. There are two ways to violate this subgoal. One is to backtrack, delete a step, and make one of the open conditions for P not be in the initial conditions. The other is to add a step for one of the open conditions where one of that step's preconditions is not in the initial conditions. We will focus our proofs on the first way to violate a subgoal; the second does not happen in our domains.

The two main features of *TOPI* that we will use in the following proofs are that steps are only added prior to existing steps, and no steps that delete open conditions can be added.

Lemma 8.1. *Problems in D^1S^1 and D^mS^1 domains have laboriously serializable subgoals for TOPI.*

Proof. The subgoals of problems in these domains are obviously serializable because it only takes the addition of one step, A_i, to achieve a subgoal, G_i. Thus, ordering the subgoals to add the steps in the right order assures that *TOPI* never needs to violate a previous subgoal. The actual serializable order is $G_n, G_{n-1}, \ldots, G_1$.

Consider any subgoal ordering other than the above serializable ordering. In this order there is a case where G_i comes before G_j when $i < j$. In such a case there exists a reachable plan-state in G_i and all of its preceding subgoals that does not contain step A_j. Since A_j deletes a precondition of A_i, *TOPI* cannot add A_j to the beginning of the plan-state until step A_i is removed, but this would violate subgoal G_i.

Thus any serializable ordering other than the actual ordering cannot ensure that some goal G_j can be solved without violating a previous subgoal G_i. This means that for a problem with n subgoals only one of $n!$ orderings assures that the subgoals can be solved sequentially without ever violating a previous subgoal. □

Lemma 8.2. *The problems in D^1S^2, D^mS^2, and D^mS^{2*} domains have subgoals that are nonserializable for TOPI.*

Proof. From the definitions of these domains there is only a single ordering of steps that can achieve the goal. Take any ordering of the subgoals and consider plan-states in all subgoals except the last subgoal, for G_x. One of the steps required for achieving G_x is A_x^2. Since A_x^2 has to appear after steps A_y^1, for all y, these steps have to be deleted before *TOPI* can add A_x^2 to the plan, but this violates *all* of the previous subgoals. Therefore, the subgoals of problems in S^2 domains are nonserializable for *TOPI*. □

8.2. Causal-link algorithms (*TOCL and POCL*)

Unlike *TOPI*, the causal-link algorithms can add steps into the middle of a plan, but they have other features that we can use in our proofs. The first such feature is that at each point in the planning process the planners focus on a single subgoal. They solve that subgoal and then move on to the next in the subgoal ordering. Solving the subgoal for G_x is a two step process for problems in the S^1 domains and a three step process for the S^2 domains. In each problem, the planners perform these processes for the current subgoal and later move onto the next.

- Achieving a subgoal in S^1 domains.
  (1) Add step A_x and $A_x \xrightarrow{G_x} s_\infty$. Protecting existing causal links from step A_x can add ordering constraints to the plan-state.
  (2) Add $s_0 \xrightarrow{I_x} A_x$. Protecting $s_0 \xrightarrow{I_x} A_x$ from any existing step A_j can add ordering constraints to the plan-state.
- Achieving a subgoal in S^2 domains.
  (1) Add step A_x^2 and $A_x^2 \xrightarrow{G_x} s_\infty$. Protecting existing causal links from step A_x^2 can add ordering constraints to the plan-state.
  (2) Add step A_x^1 and $A_x^1 \xrightarrow{P_x} A_x^2$. Protecting existing causal links from step A_x^1 can add ordering constraints to the plan-state.
  (3) Add $s_0 \xrightarrow{I_x} A_x^1$. Protecting $s_0 \xrightarrow{I_x} A_x^1$ from any existing step A_j^1 can add ordering constraints to the plan-state.

The second feature comes from the causal-link-protection step. This step ensures that the necessary truth of a solved subgoal is never violated while the relevant causal links exist. Thus, the only way to violate a previously solved subgoal is to get rid of a causal link through backtracking.

Lemma 8.3. *Problems in D^m domains have trivially serializable subgoals for both POCL and TOCL.*

Proof. Consider an arbitrary ordering of the subgoals for a problem in the $D^m S^1$ domain. The first subgoal is achievable from the *null* plan by performing the S^1 subgoal achievement process. No protection is necessary because there are no other steps in the plan. Now consider the plan-state reached by focusing on the first m subgoals. Suppose that its m steps are ordered such that A_i precedes A_j when $i < j$. This is trivially true of the plan-state reached when focusing on the first subgoal. Reaching the $(m + 1)$st subgoal involves performing the S^1 subgoal achievement process, and protecting causal links ensures that all $m + 1$ steps in the new plan-state are ordered such that A_i precedes A_j when $i < j$. Since this ordering is consistent, *POCL* can always achieve the next subgoal without violating a previous subgoal.

The same argument used for *POCL* works for *TOCL* because the ordering of a plan-state's steps upon achieving the mth subgoal, for any m, is a total ordering. This argument can also be extended to cover the $D^m S^2$ domain

because of the similarity or the domain steps and the subgoal achievement processes. □

Lemma 8.4. *Problems in D^1 domains are trivially serializable for POCL.*

Proof. Consider an arbitrary ordering of the subgoals for a problem in the D^1S^1 domain. The first subgoal is achievable from the *null* plan by performing the S^1 subgoal achievement process. No protection is necessary because there are no other steps. Now consider the plan-state reached by focusing on the first m subgoals. Suppose that its m steps are ordered such that A_{i-1} precedes A_i. This is trivially true of the plan-state reached when focusing on the first subgoal. Reaching the $(m+1)$st subgoal involves performing the two step process, and protecting causal links ensures that all $m+1$ steps in the new plan-state are ordered such that A_{i-1} precedes A_i. Since this ordering is consistent, *POCL* can always achieve the next subgoal without violating a previous subgoal.

This argument extends to the D^1S^2 domain by replacing the subgoal achievement process. Both steps added to achieve the next subgoal get ordering constraints similar to those in the D^1S^1 domain. □

Lemma 8.5. *Problems in D^1 domains are laboriously serializable for TOCL.*

Proof. Consider orderings of n subgoals that start with a subgoal G_i in D^1S^1. The next subgoal in the ordering must be G_{i-1} or G_{i+1}. Otherwise, the steps A_i and A_j, for the second subgoal G_j, are not ordered with respect to each other because they do not affect each other. The lack of an ordering constraint gives the linearization step license to choose an arbitrary order. Since the steps in the final solution are totally ordered, only one of these arbitrary orders is the correct one. This means that subgoal G_j will have to be violated, via backtracking, several times to find the order.

In general the valid serializable orders are those where a subgoal G_i only appears at the beginning or after the appearance of G_{i-1} or G_{i+1}. Any other ordering would have a case like the one described above. There are $\binom{n-1}{k-1}$ such orders that start with G_k, and the total number of such orders is 2^{n-1}. But this means that the number of bad orderings is $(n! - 2^{n-1})/n!$ which is greater than or equal to $1/n$ for $n \geqslant 3$. Thus, problems in D^1S^1 are laboriously serializable for *TOPI*.

This same argument holds for D^1S^2. □

Lemma 8.6. *Problems in the D^mS^{2*} domain are laboriously serializable for POCL and TOCL.*

Proof. Consider any ordering where G_* and G_i are the first two subgoals. The only plan-state found while focusing on these subgoals contains the totally ordered plan $[A_i^1, A_*, A_i^2]$. Now consider plan-state reached by focusing on the

first m subgoals. Suppose that its $2m-1$ steps are ordered as shown in Fig. 14. This is trivially true for the first two subgoals. Focusing on the $(m+1)$st subgoal G_x involves adding steps A_x^1 and A_x^2. Protecting links $s_0 \xrightarrow{I_x} A_x^1$ and $A_x^2 \xrightarrow{G_x} s_\infty$ forces A_* between A_x^1 and A_x^2. Protecting link $A_x^1 \xrightarrow{P_x} A_x^2$ from existing steps A_y^1 and A_y^2, where $x < y$, forces steps A_x^1 and A_x^2 between steps A_y^1 and A_y^2. Finally, protecting existing links $A_y^1 \xrightarrow{P_y} A_y^2$ from the new steps A_x^1 and A_x^2, where $y < x$, forces steps A_y^1 and A_y^2 between steps A_x^1 and A_x^2. Thus the resultant ordering of the plan for $m+1$ subgoals is ordered as shown in Fig. 14, and any ordering where G_* and G_i are the first two subgoals is a serializable ordering. Since the ordering of steps in a plan for the first m subgoals is always a total ordering, these serializable orderings are also valid for *TOCL*.

Consider any ordering where G_* is *not* one of the first two subgoals. Both *POCL* and *TOCL* can generate a totally ordered plan-state $[A_i^1, A_i^2, A_j^1, A_j^2]$ where G_i and G_j are the first two subgoals and $i < j$. This plan-state cannot be modified to include step A_* and achieve subgoal G_* without deleting A_j^1 and adding it before A_i^1. Since this deletion involves violating subgoal G_j, any ordering that does not include G_* as the first or second subgoal is not a serializable ordering.

So, since only $2/n$ of the orderings are good serialization orderings, at least $1/n$ are bad for large n. We conclude that problems in the $D^m S^{2*}$ domain are laboriously serializable for *POCL* and *TOCL*. □

Definition 8.7. Suppose O denotes a set of consistent constraints specifying a partial ordering on a set S of steps, and suppose the set $\{O_1, \ldots, O_n\}$ contains all total orderings which are consistent with O. Let $\mathcal{L}$ denote a function from invocations of *POCL* to a set of invocations of *TOCL* such that $\mathcal{L}(POCL(\prec S, O, B\succ, G, L)) = \{t_1, \ldots, t_n\}$ where $t_i = TOCL(\prec S, O_i, B\succ, G, L)$.

Lemma 8.8. *Given that both POCL and TOCL use the same goal selection strategy, for every invocation p that POCL can make, TOCL can (nondeterministically) make any of the invocations in* $\mathcal{L}(p)$.

Proof. The lemma is trivially true for the initial calls to *POCL* and *TOCL*. As an induction hypothesis, consider an invocation $p = POCL(\prec S, O, B\succ, G, L)$ at depth n of the recursion, and assume that *TOCL* can make all invocations in $\mathcal{L}(p)$. We prove by contradiction that this remains true at depth $n+1$.

Suppose that a sequence of nondeterministic choices leads p to make the recursive invocation $p' = POCL(\prec S, O_{p'}, B\succ, G, L)$, but there exists a $t' \in \mathcal{L}(p')$ which cannot be called from any $t_i \in \mathcal{L}(p)$. Since we are assuming that the lemma holds at depth n and since *POCL* and *TOCL* manipulate the sets S, B, G, and L in an identical manner, the only possible difference between p' and t' concerns their respective ordering constraints, $O_{p'}$ and $O_{t'}$. We now consider possible differences between these orderings.

Case 1. Suppose p does not add a new step. Then $O_{p'}$ is a simple refinement of O. By definition of $\mathcal{L}$, O'_t is a linearization of $O_{p'}$, so $O_{t'}$ must be a linearization of O. But this means that there exists a depth n invocation in $\mathcal{L}(p)$ with the same ordering and it could call t'.

Case 2. If p does add a step, S_{add}, then $O_{t'}$ is a total ordering of $S \cup \{S_{add}\}$. Removing references to S_{add} from $O_{t'}$ results in a total ordering O_t of S that is consistent with O. Thus $\exists t \in \mathcal{L}(p)$ with ordering O_t. But *TOCL*'s construction of range R mirrors *POCL*'s refinement of $O_{p'}$ so t can call t'.

Since these cases are exhaustive, we have verified the inductive hypothesis at depth $n + 1$. □

Proposition 8.9. *Any serializable subgoal ordering for TOCL is also a serializable subgoal ordering for POCL.*

Proof. Suppose that both *TOCL* and *POCL* use the same goal selection strategy, and there exists a serializable subgoal ordering for *TOCL* that is not a serializable subgoal ordering for *POCL*. This implies that there exists an invocation $p = POCL(\prec S, O, B \succ, G, L)$ where L contains just those causal links needed to achieve the first n subgoals, and $\prec S, O, B \succ$ cannot be further extended to achieve the first $n + 1$ subgoals.

Consider an invocation $TOCL(\prec S, O_i, B \succ, G, L) \in \mathcal{L}(p)$. From Lemma 8.8 we know that this invocation can be made by *TOCL*. Since the subgoal ordering is serializable for *TOCL*, the plan-state $\prec S, O_i, B \succ$ can be extended to achieve the first $n + 1$ subgoals, but since $\prec S, O, B \succ$ has fewer constraints, it too can be extended to achieve the first $n + 1$ subgoals. But this contradicts our hypothesis so any serializable subgoal ordering for *TOCL* must be a serializable subgoal ordering for *POCL*. □

9. Acknowledgement

This research was greatly improved by suggestions from Paul Cohen, Ernest Davis, Oren Etzioni, Steve Hanks, James Hendler, Craig Knoblock, Rao Kambhampati, David McAllester, Steve Minton, Edwin Pednault, Ying Sun, Josh Tenenberg, Brian Williams, Mike Williamson, and two anonymous referees. We thank Steven Soderland, with whom we started this project. This research was funded in part by National Science Foundation Grant IRI-8957302, Office of Naval Research Grant 90-J-1904, and the Xerox corporation.

References

[1] J.F. Allen, J.A. Hendler, and A. Tate, eds., *Readings in Planning* (Morgan Kaufmann, San Mateo, CA, 1990).

[2] A. Barrett, S. Soderland, and D.S. Weld, The effect of step-order representations on planning, Tech. Report 91-05-06, University of Washington, Department of Computer Science and Engineering, Seattle, WA (1991).
[3] T. Bylander, Complexity results for serial decomposability, in: *Proceedings AAAI-92*, San Jose (1992).
[4] P. Chalasani, O. Etzioni and J. Mount, Integrating efficient model-learning and problem-solving algorithms in permutation environments, in: *Proceedings KR-91*, Cambridge, MA (1991).
[5] D. Chapman, Planning for conjunctive goals, *Artif. Intell.* **32** (3) (1987) 333–377.
[6] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984).
[7] E. Davis, *Representations of Commonsense Knowledge* (Morgan Kaufmann, San Mateo, CA, 1990).
[8] M. Drummond and K. Currie, Goal ordering in partially ordered plans, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 960–965.
[9] O. Etzioni, Static: a problem-space compiler for Prodigy, in: *Proceedings AAAI-91*, Anaheim, CA (1991).
[10] O. Etzioni and R. Etzioni, Statistical methods for analyzing speedup learning experiments, *Mach. Learn.*, to appear.
[11] R.E. Fikes and N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, *Artif. Intell.* **2** (3/4) (1971) 189–208.
[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, New York, 1979).
[13] D. Joslin and J. Roach, A theoretical analysis of conjunctive-goal problems, *Artif. Intell.* **41** (1990) 97–106.
[14] S. Kambhampati and J.A. Hendler, A validation-structure-based theory of plan modification and reuse, *Artif. Intell.* **55** (1992) 193–258.
[15] C.A. Knoblock, Learning abstraction hierarchies for problem solving, in: *Proceedings AAAI-90*, Boston, MA (1990) 923–928.
[16] C.A. Knoblock, Automatically generating abstractions for problem solving, Ph.D. Thesis, Tech. Report CMU-CS-91-120, Carnegie Mellon University, Pittsburgh, PA (1991).
[17] R.E. Korf, Planning as search: a quantitative approach, *Artif. Intell.* **33** (1) (1987) 65–88.
[18] P. Langley, Systematic and nonsystematic search strategies, in: *Proceedings First International Conference on AI Planning Systems* (1992) 145–152.
[19] D. McAllester and D. Rosenblitt, Systematic nonlinear planning, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 634–639.
[20] D. McDermott, Regression planning, *Int. J. Intell. Syst.* **6** (1991) 357–416.
[21] S. Minton, Quantitative results concerning the utility of explanation-based learning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 564–569.
[22] S. Minton, J. Bresina and M. Drummond, Commitment strategies in planning: a comparative analysis, in: *Proceedings IJCAI-91*, Sydney, Australia (1991) 259–265.
[23] S. Minton, M. Drummond, J. Bresina and A. Phillips, Total order vs. partial order planning: factors influencing performance, in: *Proceedings KR-92*, Cambridge, MA (1992).
[24] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni and Y. Gil, Explanation-based learning: a problem-solving perspective, *Artif. Intell.* **40** (1989) 63–118; also: Tech. Report CMU-CS-89-103, Carnegie Mellon University, Pittsburgh, PA (1989)
[25] N.J. Nilsson, *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA, 1980).
[26] E.P.D. Pednault, ADL: exploring the middle ground between STRIPS and the situation calculus, in: *Proceedings KR-89*, Toronto, Ont. (1989).
[27] J.S. Penberthy and D.S. Weld, UCPOP: A sound, complete, partial order planner for ADL, in: *Proceedings KR-92*, Cambridge, MA (1992) 103–114.
[28] S. Russell, Efficient memory-bounded search algorithms, in: *Proceedings ECAI-92*, Vienna, Austria (1992).

[29] E.D. Sacerdoti, The nonlinear nature of plans, in: *Proceedings IJCAI-75*, Tblisi, Georgia (1975) 206–214.
[30] S. Soderland and D.S. Weld, Evaluating nonlinear planning, Tech. Report 91-02-03, Department of Computer Science and Engineering, University of Washington, Seattle, WA (1991).
[31] M. Stefik, Planning with constraints (MOLGEN: Part 1), *Artif. Intell.* **14** (2) (1981) 111–139.
[32] G.J. Sussman, *A Computer Model of Skill Acquisition* (American Elsevier, New York, 1975).
[33] A. Tate, Generating project networks, in: *Proceedings IJCAI-77*, Cambridge, MA (1977) 888–893.
[34] H.D. Warren, WARPLAN: a system for generating plans, Memo No. 76, Department of Computational Logic, University of Edinburgh, Edinburgh, Scotland (1974).
[35] Q. Yang and J. Tenenberg, ABTWEAK: abstracting a nonlinear, least-commitment planner, in: *Proceedings AAAI-90*, Boston, MA (1990) 204–209.