Least-Commitment Action Selection

Marc Friedman and **Daniel S. Weld**
Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
friedman, weld@cs.washington.edu

Abstract

The principle of least commitment was embraced early in planning research. Hierarchical task networks (HTNs) reason about high-level tasks without committing to specific low-level steps. Partial-order planning arose from a complementary desire to avoid unnecessary ordering commitments. But most of today's partial-order planning algorithms commit to a single producing step when supporting an open condition — even when multiple alternatives exist. Each possibility results in a different child plan, and therefore a branch in the search tree. If the various choices have common features, computation may be duplicated unnecessarily on the various branches. The FABIAN planner attempts to avoid such waste. FABIAN separates the decision to add a step from the choice of which step to add. FABIAN supports an open condition by adding an abstract action to the plan representing the disjunction of possible new supporting steps; only later does it refine this choice to a particular concrete action. We show that this abstraction can lead to exponential savings, and argue that in the worst case FABIAN never explores more than a slightly larger space of plans than does a traditional planner such as SNLP.

Keywords: Abstraction planning, Causal-link planning

Introduction

The principle of least commitment was recognized by some of the earliest planning researchers (Sacerdoti 1975; Tate 1975; Stefik 1981). Indeed, subsequent work focused on formalizing partial-order planning algorithms (Chapman 1987; McAllester & Rosenblitt 1991; Hanks & Weld 1995) and on determining the possible efficiency gains due to the least-commitment approach to step ordering (Minton, Bresina, & Drummond 1991; Minton *et al.* 1992; Barrett & Weld 1994; Veloso & Blythe 1994).

Whereas HTN planning systems have shown the importance of delaying step-selection decisions, most of today's popular planning algorithms commit to a single producing step when supporting an open condition, even if multiple alternatives exist. For example, SNLP (McAllester & Rosenblitt 1991) establishes a plan's open condition by considering both existing plan actions (subject to ordering constraints) and appropriate action-templates. Each option generates a different child plan, and therefore a branch in the search tree. If the alternatives have common features, then numerous computations may be duplicated unnecessarily on the various branches.

In this paper we describe the fully-implemented FABIAN[1] planner which delays the choice of new establishing action. When confronted with an open condition, FABIAN must still exhaust all existing and new actions that may support that condition. If there are multiple new actions, FABIAN adds a generalized action representing the disjunction of those actions. It then focuses on the preconditions and threatened links shared by *all* such steps. This computation would be duplicated on all branches by SNLP. In the extreme, if these shared preconditions either cannot be satisfied, or lead directly to a solution, then the exact choice of concrete action is irrelevant, and FABIAN's search space is exponentially smaller. Other times, FABIAN will achieve the shared preconditions, and explore the various alternatives later at lower branching cost.

The FABIAN strategy addresses planning situations in which actions have a high degree of overlap. If two operators are similar enough to work equally well in a plan, then it is appropriate to consider them as a unit — *i.e.*, to delay distinguishing between them. Thus, FABIAN precompiles all the operators which have each effect into a single *abstract operator* which generalizes their preconditions and effects.

The next sections present the FABIAN algorithm and the design decisions that prompted it. We then show that it can lead to exponential savings over SNLP in the number of nodes searched, while in the worst case increasing the search space only logarithmically. Next we detail our empirical study of FABIAN's behavior on a large set of testbed domains, showing that the algorithm compares favorably to SNLP in practice. We

[1] The American Heritage Dictionary of the English Language (Third Edition) defines the adjective *Fabian* (fā′-bē-an) as follows: "a.) Of or relating to the caution and avoidance of direct confrontation typical of the Roman general Quintus Fabius Maximus. b.) Cautious or dilatory, as in taking action."

PLAN (PROB, OPS, FLAWSELECT, REFINE)
    Let **ToVisit** = {**NullPlan**(PROB)}
    While **ToVisit** is not empty do
        REMOVE a partial plan $\mathcal{P}$ from **ToVisit**
        If $\mathcal{P}$ has no flaws, return $\mathcal{P}$
        Else
            Use FLAWSELECT to pick *flaw*
            Add REFINE($\mathcal{P}$, *flaw*, OPS) to **ToVisit**
    Fail

Figure 1: A generic refinement planner.

conclude with a discussion of related and future work.

Planning Algorithm

A refinement planner (Figure 1) takes four inputs: a planning problem, PROB, composed of an initial state and a conjunctive goal; schemata for the available actions OPS; a flaw selection function FLAWSELECT; and a refinement function REFINE.[2] The planner searches the space of partial plans and returns either a solution or "Fail" indicating that there is none. Since each partial plan represents a set of totally ordered, completely specified action sequences, the refinement function splits the implicitly represented set along the axis chosen by the flaw selection function (Kambhampati, Knoblock, & Yang 1995). The planner starts from the null plan corresponding to the set of all action sequences,[3] and splits it into ever more constrained sets until it can verify that one contains a solution. The devil is in the details: the performance of any instance of such a planner varies widely depending on the flaw selection strategy, the refinement function, and the search technique. In the next two sections we describe the refinement functions that differentiate SNLP and FABIAN. Our comparative analysis is independent of search strategy, but assumes related flaw-selection strategies.

Refinement in SNLP and FABIAN

The refinement function of SNLP, a sound and complete, systematic, partial-order, causal-link planner, is sketched in Figure 2. SNLP's plan representation contains a partially ordered set of partially bound actions and a set of causal links. A causal link, denoted $A_p \xrightarrow{g} A_c$, records an irrevocable commitment to protect the goal g during the interval between execution of the producing action, A_p, and the consuming action A_c. SNLP plans may be flawed in two ways: conditions may be open, or links may be threatened. When SNLP refines an open condition, it must consider every existing action and potential added action whose effect may establish the condition.

[2]We ignore the *plan* selection function which governs the nondeterministic search in this paper.

[3]This set is encoded as a plan with two dummy actions: A_0 has the initial conditions as its effects, while A_∞ has the problem's goal as precondition.

SNLPREFINEMENTS(**P**, *flaw*, OPS)
Case type(*flaw*):
    OpenCondition:
        For each **existing** or **new** action A_p producing g
            Return a new plan that adds to P:
                the causal link $A_p \xrightarrow{g} A_c$
                the action A_p, if it is new,
                any new bindings and orderings
                any new threats and subgoals of A_p
    Threat:
        Return if consistent:
            (Promotion) P with $\{A_c < A_t\}$, and
            (Demotion) P with $\{A_t < A_p\}$

Figure 2: The SNLP refinement algorithm.

FABIANREFINEMENTS($\mathcal{P}$, f, ABSTRACTS)
Case type(f):
    OpenCondition: f = (subgoal g of step A_c)
        For each **existing** action A_p producing g
            Return a new plan that adds to $\mathcal{P}$:
                the causal link $A_p \xrightarrow{g} A_c$
                any new bindings and orderings
                and any new threats,
                restricting A_p to consistent actions
            And also return **one** plan adding to $\mathcal{P}$:
                the unique abstract action A_g
                the causal link $\mathsf{A}_g \xrightarrow{g} A_c$
                orderings, bindings, and subgoals of A_g
    Threat:
        {Same as SNLP}
    Abstract: f = an abstract action A
        Return each consistent concretization $\mathcal{Z}$ of A_g
            in which all references to A_g are replaced
            with $\mathcal{Z}$, adding any new threats, bindings
            and preconditions.

Figure 3: The FABIAN refinement algorithm.

Since SNLP creates a new plan for each alternative added step, in future planning it is committed to that branching choice. A considerable amount of reasoning may be duplicated on each branch, but SNLP will grind through them all. Even careful flaw ordering cannot prevent all of this duplication.

A complete planner must consider all establishment options, but it need not distinguish between them so soon. The branching in some planning domains very quickly overwhelms SNLP even for small problems. The FABIAN refinement algorithm (Figure 3) exploits common features of steps by separating the decision *to add a step*, from the decision of *which step to add.* Delaying the latter decision decreases early commitment and branching. FABIAN adds a single *abstract action* A_g representing the set of *concrete actions* with an effect matching g (call this set $\mathcal{S}_g$). A_g functions as a disjunction over $\mathcal{S}_g$; FABIAN adds to the plan only those constraints required by the disjunction.

Preprocessing in FABIAN

For each goal g, FABIAN constructs a hierarchy of abstract action schemata before planning. The root of the tree is O_g, the schema for the entire set $\mathcal{S}_g$. The leaves are the original *concrete* schemata. The intermediate nodes, representing the proper subsets, are computed as needed. Figure 4 shows a sample hierarchy for the predicate `(cylinder ?x)` with only one intermediate node shown.

Notice that while preconditions are formed from the intersection of the preconditions of the children, the effects of abstract schemata are the union of the effects of the schemata they represent. These effects may be used to establish open conditions, but only those in the intersection, which are rendered in **boldface**, cause threats. This dual treatment of effects assures that constraints introduced by an abstract operator are only those required by *all* of its children.

Consider establishing `(cylindrical rod)`. SNLP tries adding each concrete action individually. But FABIAN adds an instance of O_g (call this instance A_g). FABIAN then adds the precondition `(free ?type ?machine)` to its set of flaws. It does not consider `(soft rod)` or `(metal rod)` at this point. FABIAN must focus on what all of the operators have in common. In this case, if no free machines are available of *any* type, it will discover this fact only once, rather than three times, and will backtrack to some more promising avenue.

Restricting Sets of Actions

FABIAN can support an open condition with an effect of an abstract action that is not common to all its children. However, if FABIAN continued to do so recklessly, it would be allowed to use multiple effects of an operator that were not supplied by any single child. So when FABIAN uses an effect of A_g, it immediately *restricts* the set of operators represented by A_g to consist only of those that have the desired effect. Consider again the example where the abstract operator A_g has been added to establish `(cylindrical rod)`. If A_g is used next to establish the open condition `(hot rod)`, action `EXTRUDE` becomes inconsistent because it does not supply that effect. FABIAN replaces A_g with an instance of R_g, the disjunction of just `LATHE` and `ROLL`. FABIAN constructs R_g, and any other intermediate nodes, on an as-needed basis.

The restricted plan P' is more constrained than the original, though it merely makes explicit constraints that are already in the original plan P. Doing this sort of constraint propagation may be an important source of new flaws. Restrictions of operators may increase the set of preconditions (and therefore open conditions), and the intersection of common effects (and therefore threats). In this case FABIAN uncovered a new precondition, `(metal rod)`, which may be a crucial clue to the solution. Furthermore, restriction eliminates inconsistent plans whenever no consistent concretizations remain.

Concretization: The New Refinement

Each abstract action in a plan constitutes an "abstract flaw" that must be repaired eventually. Concretization repairs that flaw by nondeterministically replacing it with one of its concrete descendents – the leaves of the abstraction hierarchy. Each refinement further constrains the plan, possibly adding open conditions, threats, or bindings that are present in the child but not in the parent. Concretization obeys the *refinement* property: if a partial plan P is concretized to P', then P' represents a subset of the ground, totally ordered plans that P represents (Kambhampati, Knoblock, & Yang 1995). Stated another way, concretizing cannot remove constraints from a plan.

Design Choices

In summary, FABIAN reasons about disjunctions of operators by precompiling abstract operators that generalize about domains. But its algorithm is not the only conceivable one. Some engineering decisions which motivate FABIAN are described below.

Intersections

FABIAN uses a fast polynomial-time algorithm to compute *conservative* intersections of logical conjunctions. Conjuncts must be exactly the same to be in the intersection. Finding *maximal* intersections, on the other hand, is isomorphic to finding S-sets in the version space algorithm for generalization learning (Mitchell 1982). In this case, that algorithm must find graph isomorphisms as a subroutine, a procedure for which no polynomial-time algorithm is known (Garey & Johnson 1979). The version space approach proved too slow in practice for some domains, despite the small size of the conjunctions.

Restrictions

FABIAN only restricts an abstract operator producing the effect used to support an open condition. However, any establishment refinement can add binding constraints that make a concretization of some abstract action inconsistent. For example, suppose the abstract action P contains action A_g. If open condition `(free ?type ?machine)` of action A_g is resolved in such a way as to add the binding `<?type = roller>`, then A_g's descendents `EXTRUDE` and `LATHE` become inconsistent, and A_g should be restricted to just `ROLL`.

It is possible for some domains that restricting operators when bindings are added may win. However, preliminary results indicate that it is not usually profitable, so it is not implemented in FABIAN.

Concretization

Concretization increases the possible flaw selection options. Completeness demands only that all concrete actions be considered – explicitly, or implicitly through abstract actions above them in the hierarchy. But which node in the abstraction hierarchy should FABIAN choose? At one extreme, it may inch down the abstraction hierarchy by splitting the set of leaves in two. Or it may, on the other hand, concretize immediately to the individual concrete actions. We chose the latter, conservative approach, which limits the increase in solution depth. Thus FABIAN can cut its losses in domains where its blind abstraction policy is not profitable.

Flaw ordering

FABIAN always waits to concretize an abstract flaw until all other flaws are resolved. It always repairs flaws shared by concrete actions before unshared flaws. Then it concretizes in LIFO order. Here FABIAN aggressively seeks the minimum commitment, a strategy whose worst-case behavior is bounded as described in the next section.

Comparison with the Committed Approach

In this section we argue that the space searched by FABIAN is never substantially larger than the space explored by SNLP, but that in some cases SNLP's search space is exponentially greater than that of FABIAN. Both planners make two kinds of choices: selection of a plan to refine, and selection of a flaw (*e.g.*, a specific open condition) to repair.

Definition 1 *A* flaw selection strategy *is a function that maps a plan to a flaw in that plan (or to* $\perp$ *if the plan is a solution).*

Whereas the search strategy requires backtracking for completeness, the flaw selection strategy does not. However, the order in which flaws are resolved is important – it determines what the search space *is*. For a meaningful comparison of the two planners, we force SNLP to use a flaw selection strategy corresponding to the one FABIAN is using. For instance, if FABIAN prefers threats to open conditions, and uses a LIFO ordering within the categories, then we make SNLP pick the same flaw that FABIAN would pick at the *corresponding* point in its search[4].

Definition 2 (Search trees) *Let* $\mathcal{RA}$ *be a planner refinement algorithm,* PROB *be a planning problem, and* F *be a flaw selection strategy for* $\mathcal{RA}$*. Define the search tree generated by* $\mathcal{RA}$ *on* PROB *given* F*,* **SearchTree**($\mathcal{RA}$*,* PROB*,* F)*, to be a tree such that*

1. *the root is* **NullPlan**(PROB) *and*
2. *the children of a node are the refinements generated by* $\mathcal{RA}$ *as directed by* F*.*

A node is a leaf iff it is either a solution *(*i.e.*, all completions solve the planning problem* PROB*) or it is a* dead end *(*i.e.*,* $\mathcal{RA}$ *generates no children for the flaw chosen by* F*).*

Two factors affect planner performance on a problem, regardless of the search strategy used: the per-plan cost, and the number of plans searched. FABIAN spends a constant amount more time than SNLP per plan, since it may *restrict* one abstract operator per node. The number of plans searched, on the other hand, depends on the branching factor, solution depth, and solution density. FABIAN's salubrious effects on these factors are summarized in Theorem 1.

Theorem 1 *Let* PROB *be a planning problem and* F_f *be a flaw selection strategy for* FABIAN*. Let* T_f *denote* **SearchTree**(FABIANREFINEMENTS*,* PROB*,* F_f) *and suppose* T_f *is finite. There exists a corresponding flaw selection strategy for* SNLP*,* F_s*, and a tree* T_s = **SearchTree**(SNLPREFINEMENTS*,* PROB*,* F_s) *such that*

1. $\mathbf{Depth}(T_f) \leq 2 \times \mathbf{Depth}(T_s)$
2. $\mathbf{Size}(T_f) \leq \mathbf{Depth}(T_s) \times \mathbf{Size}(T_s)$
3. $\mathbf{Size}(T_s)/\mathbf{Size}(T_f)$ *may be exponential in the problem size*

Proof. The proof hinges on the notion of plan *correspondence*, a mapping which is tedious to define rigorously. Informally, a plan in T_f corresponds to a plan in T_s if the path to each from its respective root node contains the same sequence of flaw and repair choices, except for concretizations. We can construct F_s from F_f inductively, in a way that maximizes correspondence, as follows: Let $\mathrm{F}_s(\mathbf{NullPlan}(\text{PROB})) = \mathrm{F}_f(\mathbf{NullPlan}(\text{PROB}))$. Whenever $\mathrm{F}_f(\mathcal{P})$ is an open condition or threat, and $\mathcal{P}'$ in T_s corresponds to $\mathcal{P}$, choose $\mathrm{F}_s(\mathcal{P}') = \mathrm{F}_f(\mathcal{P})$. Whenever $\mathrm{F}_f(\mathcal{P})$ is an abstract flaw, let $\mathrm{F}_s(\mathcal{P}') = \mathrm{F}_f(\mathbf{child}(\mathcal{P}))$ for the child with the concretization matching $\mathcal{P}'$.

Part 1. An illustration of T_s and T_f for a simple case is given in Figure 5. Here F_f always chooses to concretize right away. This corresponds to the strategy F_s which establishes the same subgoals by just adding the concrete steps in the first place. T_f has (at most) two nodes (separated by a "link to abstract action" refinement) for every one in T_s. This construction demonstrates that given T_s, there is always a F_f generating a T_f at most double in size, with corresponding nodes at most double in depth, since there can be no more concretizations added along a path to a plan than there are steps in that plan.

Delaying concretization will frequently make the number of nodes in T_f far smaller, but there is no guarantee; the ratio $\mathbf{Size}(T_f)/\mathbf{Size}(T_s)$ may be as much as $\mathbf{Depth}(T_s)$, in trees where concretization commonly results in only one child with consistent bindings.

Part 2. (Worst case) To see why T_f has at most $\mathbf{Depth}(T_s)$ times as many nodes as T_s, consider any valid flaw selection strategy F_f for FABIAN. F_f can concretize an abstract operator at any time between its introduction and a point where there are no other flaws. We inductively construct F_s from F_f as explained above, leading to the corresponding SNLP tree T_s. T_f has two kinds of nodes: those with abstract operators but no other flaws (call them **white nodes**), and the others (**black nodes**). The correspondence between F_s and F_f leads to an inductively defined surjection, M, that maps every node of T_s onto a black

[4]This does not imply using the exact *same* flaw selection function that FABIAN uses, but rather one derived from it. SNLP has a superset of FABIAN's flaws (minus abstract flaws) to choose from at any given time.

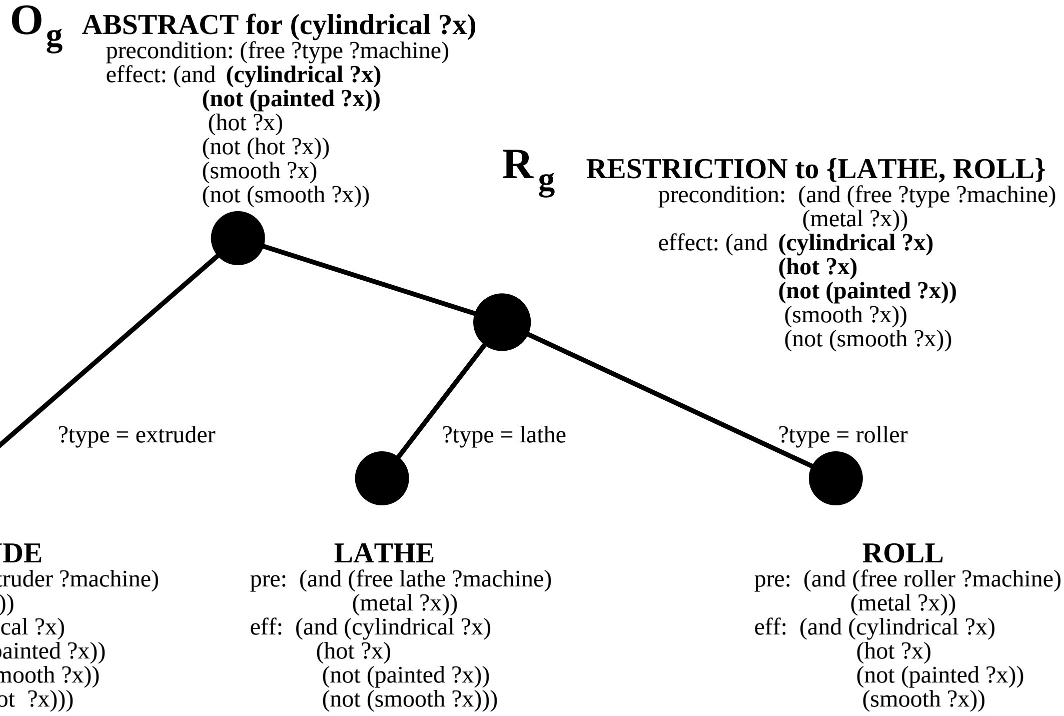

Figure 4: Abstraction hierarchy for goal `(cylindrical ?x)`.

node of T_f. Distinct nodes of T_s may map to the same black node, but every black node is the image of some node in T_s, so $\mathbf{Black}(T_f) \leq \mathbf{Size}(T_s)$. ($M$ is surjective because every black node has at least one corresponding SNLP node, or it would be inconsistent and we assume consistency.)

The consistency assumption also ensures that every white node has a consistent concretization, hence every T_f leaf is black. There may be no more white nodes on any path from root to leaf than there are steps in that leaf plan, which is at most $\mathbf{Depth}(T_s) - 1$. Hence

$$\begin{aligned}\mathbf{Size}(T_f) &= \mathbf{Black}(T_f) + \mathbf{White}(T_f)\\ &\leq \mathbf{Black}(T_f) + \\ &\quad\ \mathbf{Black}(T_f)(\mathbf{Depth}(T_s) - 1)\\ &\leq \mathbf{Depth}(T_s)\mathbf{Size}(T_s)\end{aligned}$$

Part 3. (Best case) Fabian tactics, properly applied, can reduce exponential search spaces to linear ones. Consider an artificial domain IDEAL-n with n pairs of identical operators $\{O_1, O'_1, \ldots, O_n, O'_n\}$. Each operator O_i has a single precondition P_i supplied by the effect of O_{i-1}, so plans form a linear chain of length n. To attempt to satisfy the goal P_n, when P_0 is false, SNLP searches a complete binary tree of size 2^n before it realizes that P_0 is a necessary prerequisite for P_n. But as long as FABIAN only concretizes when it can do nothing else, it knows after only n steps! Moreover, when no solution plan exists, there is no plan selection or flaw selection strategy that will shave a single node from SNLP's exponential search. A similar argument applies to solvable problems with dead-end subtrees structured like the IDEAL-n problem. □

For infinite search trees, the analysis is substantially the same. We find finite trees that are subgraphs of the infinite search trees.

Corollary 2 (Infinite search trees) *Suppose T_f is infinite. Define T_s and M by induction. Consider any finite connected component T'_f of T_f including the root node. There is a connected component T'_s of T_s which M maps surjectively to the black nodes of T'_f. Then all three parts of Theorem 1 apply to T'_f and T'_s.*

Proof Sketch. T_s and M are defined inductively from any tree T_f. Since the three parts of Theorem 1 only depend on the mapping M relating the two trees, it is sufficient to show that T'_s is connected (and thus a tree). But by construction, $M(\mathbf{parent}(P))$ is an ancestor of $\mathbf{parent}(M(P))$ for any node n in T_s. Thus if $P \in T'_s$, so is $\mathbf{parent}(P)$. □

Theorem 1 and its corollary show that FABIAN has the potential for exponential speedup while risking only a logarithmic (in the common case of bushy T_s) slowdown in the worst case. But in some respects the results are less satisfying than desired. In particular, they only promise that for every FABIAN flaw selection strategy *there exists* a corresponding strategy for SNLP that is not much better. For some domains, there may yet be an SNLP strategy that beats all FABIAN strategies. This concern is real: since the precondition of

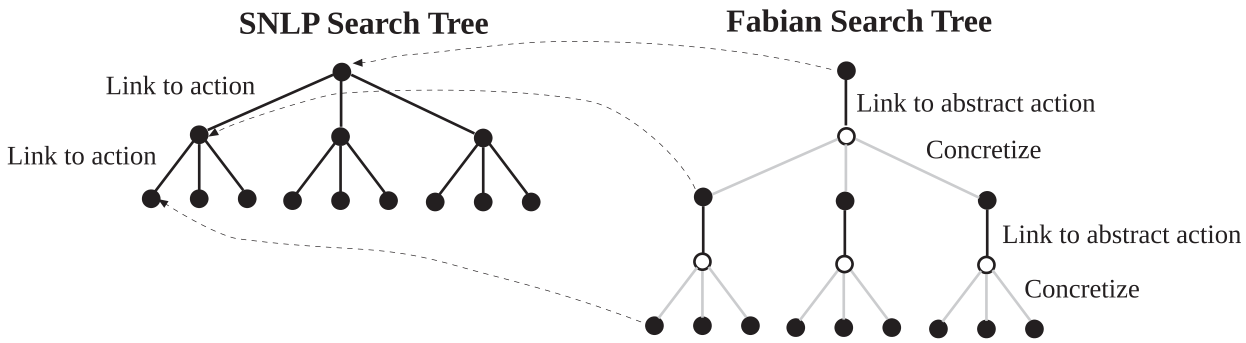

Figure 5: Sample corresponding search trees for SNLP and FABIAN.

an abstract action is defined as the *intersection* of the preconditions of its concretizations, FABIAN can only consider a subset of the open-condition flaws that are available to SNLP. Since one common form of domain-dependent search control is a careful ordering of operator preconditions combined with a LIFO flaw selection strategy (Williamson & Hanks 1996), FABIAN may be harder for users to control.

Another caveat concerns the proof's assumption that all FABIAN plans have a consistent concretization. This would amount to an NP-complete consistency check at each node, which is not implemented. FABIAN's restriction procedure will catch those cases where the constraints on a single abstract action rule out all of its concretizations. However, there may be plans with multiple abstract actions whose concretizations are all mutually exclusive. The situation is analogous to arc and path consistency in constraint satisfaction. For most domains we have studied, our arc-consistency-like implementation seems to be avoiding the pitfalls of refining inconsistent subtrees.

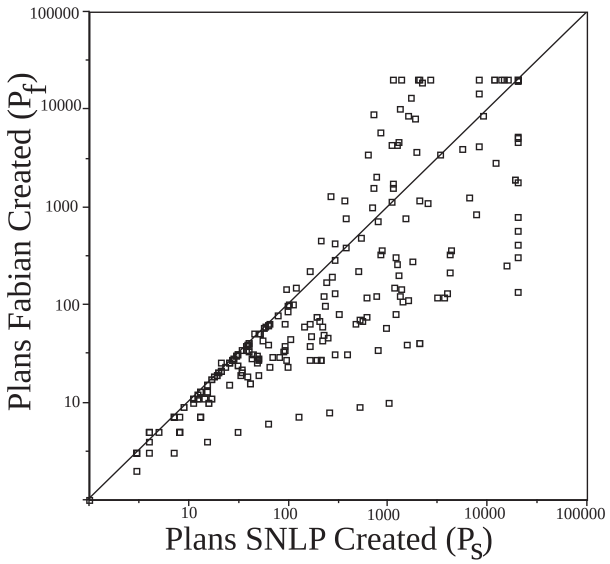

Figure 6: Log/log scale graph of # plans created.

Empirical Investigation

The theoretical results we have presented depend on SNLP using a corresponding flaw selection strategy to FABIAN's. This is unfairly restrictive of SNLP. To see if FABIAN's benefits would hold up in practice, we have instead allowed FABIAN and SNLP to use their default flaw selection strategies. We have run them indiscriminately on all the domains we could find, and on a few (fewer than 10%) that we created.[5]

Figure 6 shows the number of plans generated by FABIAN and SNLP for 312 problems, with a search limit of 20000 plans. The log/log scale best presents the trends in a suite of problems varying so widely in difficulty. The 45 degree line divides those problems on which FABIAN does better (below) from those problems on which SNLP does better (above). The best-fit line for the data is $\log P_f = .89 \log P_s$, with a degree of confidence (R^2) of .8. Since the slope is less than one, FABIAN does better on average, and by a larger margin on larger problems.

Figure 7 shows the "internal time" reported by Lisp for the same test runs. The data is visibly more chaotic, but the linear regression is reassuring. The best-fit line is $\log T_f = .91 \log T_s - .06$, with $R^2 = .76$. The slope is just slightly closer to one, despite FABIAN's increased per-plan cost.

These results are quite positive. A line on a log/log graph with slope .89 disguises just how much better FABIAN does on large problems; if $P_s = 20000$, then $P_f = 6728$. In fact, if we believe that a straight line reasonably approximates the data, then FABIAN's improvement is exponential, since $P_f = P_s^{.89}$. FABIAN spends a constant amount more time per plan doing restrictions, but this effect is dwarfed by the improvement FABIAN is accruing on the large problems.

We have verified FABIAN's potential for exponential speedup with the artificial domain family from the proof. On the problem IDEAL-n, SNLP does indeed

[5] FABIAN is implemented on top of Barrett's UCPOP version 4.0 (Penberthy & Weld 1992), which subsumes SNLP, running on a Silicon Graphics Indy under Allegro Common Lisp 4.2. Both FABIAN and SNLP repair unseparable threats before open conditions before abstract flaws, with a LIFO ordering within these groupings. Both prefer plans with fewer `steps + flaws`.

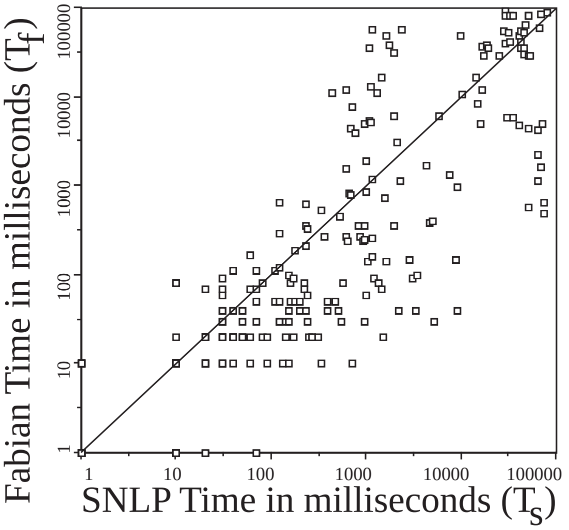

Figure 7: Log/log scale graph of internal times.

create and search $2^n - 1$ plans before failing, while FABIAN creates and searches only n of them.

The cost of operator precompilation is negligible. Our preprocessor took .15 CPU seconds average on our ten domains, or under one second of real time. Taking unions and intersections of conjunctive formulae costs $O(n^2)$ unifications in the domain size.

Related Work

We mentioned work evaluating the efficiency that planners could possibly gain by adopting a least-commitment approach to action ordering (Minton, Bresina, & Drummond 1991; Minton *et al.* 1992; Barrett & Weld 1994; Veloso & Blythe 1994). Our work applies the least commitment strategy to the choice of new action to support an open condition.

Kambhampati's formulation (Kambhampati 1992) of the multi-contributor causal link approach originally introduced in NONLIN (Tate 1975) is also similar to our work since it adopts a reduced commitment approach to the specific producer of a causal link; essentially this amounts to supporting a consuming step with a causal link containing a disjunction of concrete steps. With FABIAN every causal link is supported by a single producing step, but that step may be abstract — *i.e.*, represent a disjunction of concrete steps. The approaches are actually orthogonal solutions to two different sources of combinatorial blowup. Multi-contributor causal links merge partial plans containing the same steps but different causal links, while FABIAN merges plans with different steps but the same causal links. It would be interesting to combine our approaches.

Peot and Smith (Peot & Smith 1993) considered different flaw selection strategies for SNLP-style planners. They showed that delaying consideration of certain threats could lead to search trees that are sometimes smaller, but never bigger than those of SNLP. In many ways, we were motivated by Peot and Smith's result; unfortunately our upper bound is not as good, since FABIAN must sometimes consider a tree that is larger than that of SNLP. Note that the approaches are complementary, since we consider open condition flaws while Peot and Smith addressed threats.

FABIAN uses abstraction in a way radically different from *state-abstraction* planners like ABSTRIPS (Sacerdoti 1974) and ABTWEAK (Yang & Tenenberg 1990), which solve incrementally more and more concrete planning problems using the previous abstraction as a guide. State abstraction can cause exponential speedup in certain regular domains (Knoblock 1992), but it depends on the *downward refinement property* to prevent backtracking over abstraction levels. Recently, Backstrom and Jonsson showed that there are domains for which the ALPINE (Knoblock 1991) and HIPOINT (Bacchus & Yang 1994) algorithms may generate exponentially longer plans than optimal, *despite* the downward refinement property (Backstrom & Jonsson 1995), because of the requirement that plans build on solutions from previous levels. FABIAN's conservative abstractions never overconstrain plans, so FABIAN need not backtrack over abstraction decisions. Nor does it impose the restrictions of incremental search, as illustrated by our superior complexity results. FABIAN could certainly benefit, though, from the state-abstraction notion of predicate criticality to improve its abstraction decisions.

Kramer and Unger (Kramer & Unger 1993) considered a form of abstract planning which is conceptually very similar to the FABIAN approach. But Kramer and Unger's system used a domain-specific, manually-created hierarchy of abstract operators in contrast to FABIAN's automatic compilation algorithm. It is unclear if Kramer and Unger's system was implemented, but no empirical tests are reported. Furthermore, in contrast with the strong result of our Theorem 1, Kramer and Unger do not attempt any analytic assessment of their approach's benefit.

Conclusions and Future Work

We have presented FABIAN, a planning algorithm that exploits a new form of least commitment. When supporting an open condition, FABIAN separates the decision to add a step from the choice of which step to add. FABIAN's polynomial-time preprocessing strategy constructs abstract operators for each predicate in a domain theory. Later, when supporting an open condition during planning, FABIAN adds a single instance of an abstract operator instead of branching to consider all possible concrete actions. This abstract action is refined to a particular concrete action when necessary.

We show that in theory this abstraction can lead to exponential savings, and prove that in the worst case FABIAN never explores more than a slightly (roughly logarithmic) larger space of plans than a traditional planner such as SNLP. The power of this result is tempered by FABIAN's additional source of inconsistent plans, and by the requirement of our analysis that SNLP use a corresponding flaw selection strategy to

FABIAN's. Furthermore, FABIAN obfuscates domain-dependent search control. Although FABIAN inherently has more diverse options than SNLP it has difficulty capitalizing on the simple but powerful technique of precondition ordering that SNLP users have exploited to encode their search control knowledge, because FABIAN plans have fewer open conditions to choose from. On the other hand, it should be less hindered by poor precondition ordering by the programmer.

Our experimental results on a large test suite indicate that the FABIAN approach performs better on average than SNLP. While the benefits are significant, they are not revolutionary. Even if FABIAN's savings is exponential, we cannot do away with domain-dependent search control. We may have found a practical technique to improve domain-independent planning, but several serious problems remain before it will become widely adopted. Applying smarter domain-independent flaw selection strategies may make FABIAN's inability to exploit precondition ordering an advantage. Finally, extending FABIAN to handle quantification and conditional effects would vastly expand the applicability of least-commitment action selection.

Acknowledgements

Many thanks to Tony Barrett, Adam Carlson, Bob Doorenbos, Oren Etzioni, Keith Golden, Steve Hanks, Subbarao Kambhampati, Nick Kushmerick, Neal Lesh, Greg Linden, Rich Segal, Ying Sun, and Mike Williamson for helpful comments and discussions. This research was funded in part by Office of Naval Research Grant N00014-94-1-0060, by National Science Foundation Grant IRI-9303461, by ARPA / Rome Labs grant F30602-95-1-0024, by a gift from Rockwell International Palo Alto Research, and by an NSF Graduate Fellowship.

References

Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence* 71:43–100.

Backstrom, C., and Jonsson, P. 1995. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. 15th Int. Joint Conf. on A.I.*, 1599–1604.

Barrett, A., and Weld, D. 1994. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence* 67(1):71–112.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W. H. Freeman and Company.

Hanks, S., and Weld, D. S. 1995. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research* 319–360. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Kambhampati, S.; Knoblock, C.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* 76:167–238.

Kambhampati, S. 1992. Characterizing multi-contributor causal structures for planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, 116–125.

Knoblock, C. 1991. *Automatically Generating Abstractions for Problem Solving.* Ph.D. Dissertation, Carnegie Mellon University. Available as technical report CMU-CS-91-120.

Knoblock, C. 1992. An analysis of ABSTRIPS. In *Proc. 1st Intl. Conf. on A.I. Planning Systems.*

Kramer, M., and Unger, C. 1993. A generalizing operator abstraction. In *Proceedings of the Second European Workshop on Planning.*

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, 634–639.

Minton, S.; Drummond, M.; Bresina, J.; and Phillips, A. 1992. Total order vs. partial order planning: Factors influencing performance. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning.*

Minton, S.; Bresina, J.; and Drummond, M. 1991. Commitment strategies in planning: A comparative analysis. In *Proceedings of IJCAI-91*, 259–265.

Mitchell, T. 1982. Generalization as search. *Artificial Intelligence* 18:203–226.

Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

Peot, M., and Smith, D. 1993. Threat-removal strategies for partial-order planning. In *Proc. 11th Nat. Conf. on A.I.*, 492–499.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Sacerdoti, E. 1975. The nonlinear nature of plans. In *Proceedings of IJCAI-75*, 206–214.

Stefik, M. 1981. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence* 14(2):111–139.

Tate, A. 1975. Interacting goals and their use. In *Proceedings of IJCAI-75*, 215–218.

Veloso, M., and Blythe, J. 1994. Linkability: Examining causal link commitments in partial-order planning. In Hammond, K., ed., *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, 170–175. AAAI.

Williamson, M., and Hanks, S. 1996. Flaw selection strategies for value-directed planning. In *Proc. 3rd Intl. Conf. on A.I. Planning Systems.*

Yang, Q., and Tenenberg, J. 1990. ABTWEAK: Abstracting a nonlinear, least-commitment planner. In *Proc. 8th Nat. Conf. on A.I.*, 204–209.