Challenges in bridging plan synthesis paradigms

Subbarao Kambhampati[\]
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
rao@asu.edu; http://rakaposhi.eas.asu.edu/rao.html

Abstract

In the last three years, several "radically new" and promising approaches have been developed for tackling the plan synthesis problem. Currently, these approaches exist in isolation as there is no coherent explanation of their sources of strength *vis a vis* the traditional refinement planners. In this paper, I provide a generalized view of refinement planning, that subsumes both traditional and newer approaches to plan synthesis. I will interpret the contributions of the new approaches in terms of a new subclass of refinement planners called "disjunctive planners". This unifying view raises several intriguing possibilities for complementing the strengths of the various approaches. I will identify and pose these as challenges to the planning community.

1. Introduction

Most traditional approaches to plan generation, developed over the last twenty years, work by searching in the space of partial plans, extending a plan incrementally until it becomes a solution, and backtracking when a plan can no longer be fruitfully refined. More recently, several approaches have been developed that relate plan synthesis to constraint satisfaction. These include Graphplan [Blum & Furst, 1995], SATPLAN [Kautz & Selman, 1996], COPS [Ginsberg, 1996], Descartes [Joslin & Pollack, 1996] and UCPOP-D [Kambhampati & Yang, 1996]. Graphplan and SATPLAN approaches, in particular, have demonstrated impressive scale-up potential in practice.

At present, there exists a huge gulf between these new breed of algorithms and the traditional refinement planning approaches. An important challenge for the planning community is to bridge these strands of research to see if their strengths can be complemented. To this end, I present a generalization of the refinement planning framework developed in our previous work [Kambhampati, Knoblock & Yang, 1995; Kambhampati & Srivastava, 1996] that covers most of the currently known approaches.

According to my general framework, partial plans are shorthand notations for sets of action sequences (called the candidate set of the plan). Plan synthesis involves a "*split and prune*" search [Pearl, 1980]. The pruning is carried out by applying refinement operations to a given set of partial plans (called a *planset*). Pruning attempt to incrementally get rid of non-solutions from the candidate set. The splitting part involves pushing the component partial plans of a planset into different branches of the search tree. Its main aim is to reduce the cost of applying refinements and termination check to the planset. Termination test involves checking if a solution can be extracted from the (possibly exponential number of) minimal candidates of the current planset. The extraction process can in general be cast as a constraint satisfaction search.

Within this framework, traditional planners can be seen as doing both pruning (refinement) and full splitting (pushing each component of the planset into a different search branch). They thus refine and terminate on individual partial plans (rather than plansets). The scale-up problems associated with these planners can be related to their full splitting. An obvious way of curing this malady involves avoiding splitting or controlling it intelligently. To implement this idea--which I call "**disjunctive planning**"-- we need to find ways of effectively refining and terminating on large sets of plans. All the newer planners can be seen as providing potential solutions to these two problems. Graphplan (and COPS) can be seen as clarifying the issues involved in compactly representing and reasoning with plansets. SATPLAN demonstrates how the problem of extracting solutions from a planset can be posed as a SAT problem, so that it can be solved by the new breed of efficient SAT solvers [Kautz et. al., 1992; Crawford & Auton, 1996; Bayardo & Schrag, 1997]. Finally, Descartes and UCPOP-D can be seen as exploring the effect of controlled (rather than complete or no) splitting of the plansets.

Viewing existing planners as points in a spectrum of possible disjunctive planners opens several exciting avenues of focused research: Could the missing planners corresponding to the other points in the spectrum be more efficient? What are the tradeoffs governing the efficiency of these spectrum of planners? I will identify specific short-term and intermediate-term challenges that have to be undertaken to answer these questions.

The rest of the paper is organized as follows. Section 2 presents my generalized framework for refinement planning. Section 3 motivates the idea of disjunctive planning, and interprets several newer approaches as providing guidelines for implementing disjunctive planners. Section 4 relates SATPLAN family of algorithms to disjunctive planning. Section 5 lists and motivates a set of challenge problems that arise from this unifying view. Section 6 briefly discusses the logistics of coordinating the research into the challenge problems.

2. Refinement Planning: Overview

Since a solution for a planning problem is ultimately a sequence of actions, plan synthesis in a general sense involves sorting our way through the set of *all* action sequences until we end up with a sequence that is a solution. This is the essential idea behind refinement planning. The

[\] This research is supported in part by the NSF NYI award IRI-9457634, the ARPI Initiative grant F30602-95-C-0247 and the ARPA AASERT grant DAAH04-96-1-0231. I would like to thank Bart Selman, Laurie Ihrig, Amol Mali and the anonymous reviewers for critical comments on a previous draft.

sets of action sequences are represented and manipulated in terms of **partial plans** which can be seen as a collection of constraints**.** The action sequences denoted by a partial plan, i.e., those that are consistent with its constraints, are called its **candidates**. For technical reasons that will become clear later, we find it convenient to think in terms of *sets* of (instead of single) partial plans. A set of partial plans is called a **planset** with its constituent partial plans referred to as the **components.** The candidate set of a planset is defined as the union of the candidate sets of its components.

A refinement (pruning) operation narrows the candidate set of a planset by adding constraints to its component plans. If no solutions are eliminated in this process, we will eventually progress towards the set of all solutions. Termination can occur as soon as we can pick up a solution using some bounded time operation -- called the *solution extraction function.* To make these ideas precise, we shall now look at the syntax and semantics of partial plans and refinement operations.

2.1 Partial Plan Representation: Syntax

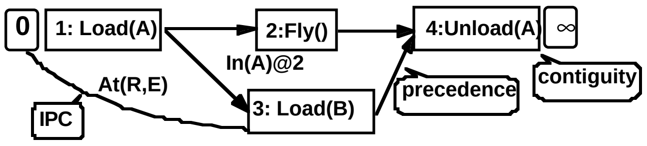

Figure 1. An example (partial) plan in rocket domain.

A partial plan can be seen as any set of constraints that together delineate which action sequences belong to the plan's candidate set and which do not. One commonly used representation models partial plans as a set of steps, ordering constraints between the steps, and auxiliary constraints (we shall discuss alternative representations, such as that proposed by Ginsberg [1996], later). The ordering constraints may require steps to precede each other or be contiguous. The auxiliary constraints demand the preservation of a condition over a time interval between two steps (called IPCs) or the truth of a condition at a time point (called point truth constraints). An example plan in this representation from the one-way rocket domain (involving transportation of two packages from earth to moon using a single one-way rocket) is shown in Figure 1.

2.2 Partial Plan Representation: Semantics

The semantics of the partial plans are given in terms of candidate sets. A candidate can be seen as a model of the partial plan constraints. An action sequence belongs to the candidate set of a partial plan if it contains the actions corresponding to all the steps of the partial plan, in an order consistent with the ordering constraints on the plan, and it also satisfies all preservation constraints. For the example plan shown in Figure 1, the action sequences shown on the left in Figure 2 are candidates, while those on the right are non-candidates.

Notice that the candidates may contain more actions than are present in the partial plan. Because of this, a plan's candidate set can be potentially infinite. We define the notion of "**minimal candidates**" to let us restrict our attention to a finite subset of the possibly infinite candidate set. Specifically, minimal candidates are candidates that only contain the actions listed in the partial plan (thus their length is equal to the number of steps in the plan other than 0 and ∞). The top candidate on the left of Figure 2 is a minimal candidate while the bottom one is not. There is a one-to-one correspondence between the minimal candidates and the syntactic notion of safe linearizations of a plan, where a safe linearization is a permutation of plan steps that satisfies the auxiliary preservation constraints. For example, the minimal candidate on the top left of Figure 2 corresponds to the safe linearization 0-1-3-2-4-∞ (as can be verified by translating the step names in the latter to corresponding actions).

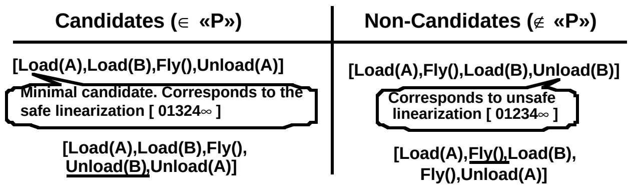
Candidates (∈ «P»)
Non-Candidates (∉ «P»)
[Load(A),Load(B),Fly(),Unload(A)]
Minimal candidate. Corresponds to the safe linearization [01324∞]
[Load(A),Load(B),Fly(), Unload(B),Unload(A)]
[Load(A),Fly(),Load(B),Unload(B)]
Corresponds to unsafe linearization [01234∞]
[Load(A),Fly(),Load(B), Fly(),Unload(A)]

Figure 2. Candidate set of a plan

2.3 Refinement Strategies

A refinement strategy R maps a planset P to another planset P^R such that the candidate set of P^R is a subset of the candidate set of P. R is said to be **complete** if P^R contains all the solutions of P. It is said to be **progressive** if the candidate set of P^R is a strict subset of the candidate set of P. R is said to be **strongly progressive** if the length of the minimal candidates increases after the refinement. The degree of **progressiveness** is measured in terms of the reduction in the candidate set size. It is said to be **systematic** if no action sequence falls in the candidate set of more than one component of P^R.

Completeness ensures that we don't lose solutions by the application of refinements. Progressiveness ensures that refinement narrows the candidate set (i.e., has pruning power). Strongly progressive refinement strategies simultaneously shrink the candidate set of the plan, and increase the length of its minimal candidates. This provides an incremental way of exploring the (potentially infinite) candidate set of a planset for solutions: *Examine the minimal candidates (corresponding to safe linearizations) of the planset after each refinement to see if any of them correspond to solutions.* Systematicity ensures that we never consider the same candidate more than once, if we were to explore the components of the planset separately.

Refinements are best seen as canned inference procedures that compute the consequences of the meta-theory of planning, and the domain theory (in the form of actions), in the specific context of the current partial plan constraints. Traditional planners use four types of refinement strategies -- forward state space, backward state space, plan space and task-reduction [Kambhampati, 1997]. Figure 3 shows a forward state space refinement of a partial plan in the one-way rocket domain. It takes the null planset, corresponding to all action sequences and maps it to a planset containing 3 components. In this case, given the theory of planning, which says that solutions must have actions that are executable in their respective states, and the current planset constraint that the state of the world before the first step is the initial state, the forward state space refinement infers that the only actions that can come as the second step in the solution are Load(A), Load(B) and Fly().

This particular refinement is complete since no solution to the rocket problem can start with any other action for the given initial state. It is progressive since it eliminated the action sequences not beginning with Load(A), Load(B) or Fly() from consideration. Finally, it is systematic since no action sequence will belong to the candidate set of more than one component (the candidates of the three components will differ in the first action).

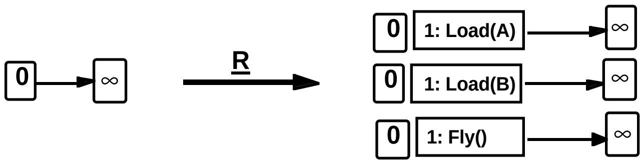

Figure 3. An example refinement strategy (Forward State Space)

2.4 Plan Synthesis

We are now in a position to present the general refinement planning template, which we do in Figure 4. If the current planset has an extractable solution—which is checked by inspecting its minimal candidates to see if any of them is a solution— we terminate. If not, we select a refinement strategy *R* and apply it to the current planset to get a new planset. This is the pruning step. The third step is the splitting step, and involves pushing the components of the refined planset into different branches. The splitting is controlled by the parameter *k*. If *k* equals 1, no splitting is done. If *k* equals the number of components of the refined planset, full splitting is done. Intermediate values of *k* correspond to intermediate levels of splitting. After the splitting step, one of the search branches is selected non-deterministically, and pruning and splitting is applied to it recursively.

```
Refine ( P : a plan set)

  0*.  If «P » is empty, Fail.
  1.   If a minimal candidate of P is a solution, terminate.
  2.   Select a refinement strategy R .
         Appply R  to  P  to get a new plan set P'
  3.   Split P' into k  plansets
  4.   Simplify plansets by introducing disjunction and
         constraint propagation.
  5.   Non-deterministically select  one of the plansets P'_i
       Call Refine(P'_i)
```

Figure 4. Refinement planning with pruning and splitting

As long as the selected refinement strategy *R* is complete, the algorithm will never lose a solution. As long as the refinements are strongly progressive, for solvable problems, the algorithm will eventually reach a planset one of whose minimal candidates will be a solution.

The solution extraction process involves searching through the possibly exponential number of minimal candidates of a planset for a solution. As *k* increases, individual plansets have fewer components, and consequently the cost of solution extraction reduces.

The algorithm template in Figure 4 covers both traditional and newer plan synthesis approaches (see [Kambhampati, 1997] for a more elaborate discussion). Traditional refinement planners, such as UCPOP, Prodigy, SNLP, etc. correspond to complete splitting (i.e., *k* equals the number of components of the planset) differing mainly in the type of refinements they employ. The newer approaches such as Graphplan, COPS, Descartes and UCPOP-D can be seen as handling plansets without splitting. The first two do not do any splitting, while the last two do controlled splitting. Finally, SATPLAN approaches can be understood as posing the solution extraction as a SAT problem.

3. Disjunctive planning

We saw that traditional planners do refinement planning with full splitting, differing mainly in terms of the specific refinement strategies they use. Unfortunately, these planners tend to generate huge search spaces and have in practice shown disappointing scale-up potential. Viewing the newer approaches such as Graphplan and SATPLAN as instances of our framework suggests that a promising general solution involves handling plansets without splitting. This idea, referred to as disjunctive planning, raises three immediate issues: *How do we (a) represent, (b) refine and (c) extract solutions from large plansets?* I will discuss the first two issues in this section. The most promising approach for solution extraction at present seems to be to pose it as a CSP/SAT problem. I will discuss this further in Section 4, in the context of SATPLAN.

3.1 Disjunctive Representations

First off, keeping plansets together may lead to very unwieldy data structures. The way to get around this is to "internalize" the disjunction in the plansets so that we can represent them more compactly. The general idea of disjunctive representations is to allow disjunctive step, ordering, and auxiliary constraints into a plan. Figure 5 and Figure 6 show two examples the idea. The three plans on the left in Figure 5 can be combined into a single disjunctive step, with disjunctive contiguity constraints. Similarly, the two plans in Figure 6 can be combined by using a single disjunctive step constraint, a disjunctive precedence constraint, a disjunctive interval preservation constraint and a disjunctive point truth constraint.

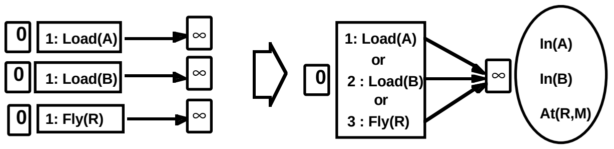

Figure 5. Disjunction over state-space refinements.

Candidate set semantics for disjunctive plans follow naturally from the fact that the presence of the disjunctive constraint $c_1 \vee c_2$ in a partial plan constrains its candidates to be consistent with either c_1 or c_2. So, solution extraction on disjunctive plans can still be posed as the problem of searching through minimal candidates for a solution.

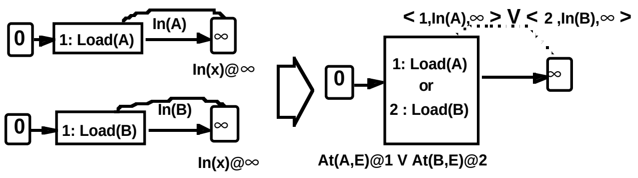

Figure 6. Disjunction over plan-space refinements

3.2 Refining Disjunctive Plans

In refining the disjunctive plans, we face a tradeoff between the pruning power of the refinement and its cost. The maximal pruning power is obtained by externalizing the disjunction, applying the traditional refinements to each of the resulting non-disjunctive partial plans, and combining them back into a disjunctive plan. For example, to refine the plans on the left in Figures 5 and 6, we simply go back to the plans on the right, refine them, and combine the results into a new disjunctive plan. This approach is however infeasible in practice since it makes the refinement stage exponentially hard. We need to generalize the traditional refinement strategies so that they can apply directly to the disjunctive plans. I will now attempt to outline the critical issues in doing this.

Since refinement strategies essentially derive the consequences of planning and domain theory in the context of the current plan constraints, it is clear that internal disjunction in the partial plan may lead to the derivation of weaker consequences (thus reducing their pruning power). For example, for the disjunctive plan on the right in Figure 5, we don't know which of the three steps will actually occur in the eventual solution. Consequently, we don't exactly know what the state of the world will be after the disjunctive step. This means that the forward state space refinement will not be able to infer exactly which actions can appear in the second step of the solution. Similarly, for the disjunctive plan in Figure 6, we don't know whether steps 1 or 2 or both will be present in the eventual solution. Thus a plan space refinement won't know whether it should introduce steps relevant to *At(A,E)* precondition, those relevant to *At(B,E)* precondition, or both.

All is not lost however, as the refinements can still infer a superset of the relevant actions. For example, for the plan in Figure 5, although we do not know the exact state after the first (disjunctive) step, we know that it can only be a subset of the union of conditions in the effects of the three steps. Knowing that only Load(A), Load(B) or Fly(R) can be the first steps in the plan tells us that the state after the first step can only contain the conditions In(A), In(B) and At(R,M). We can thus generalize forward state space refinement to add only those actions whose preconditions are subsumed by the union of propositions comprising the effects of the three steps. Similarly, plan-space refinement can be generalized to introduce actions to achieve all parts of a disjunctive precondition.

These "naive" generalizations of the standard refinements are still complete, but have far less pruning power than the standard refinements operating on non-disjunctive plansets. In the case of the generalized forward state space refinement, even if the preconditions of an action are in the union of effects of the preceding disjunctive step, there may be no real way for that action to actually be take place. For example, in Figure 5, although the preconditions of "unload at moon" action may seem satisfied, it is actually never going occur as the second step in any solution because Load() and Fly() cannot be done at the same time. In fact, this naive generalization may allow an exponential number of additional actions that will not be considered by traditional forward state space refinement operating on non-disjunctive plans [Kambhampati & Lambrecht, 1997].

Although the loss of progressiveness in refining a disjunctive plan cannot be completely avoided, it can be reduced to a significant extent by *extending the types of consequences inferred by the refinements*. Traditional refinement strategies concentrate exclusively on inferring the identities of new actions that must be part of any eventual solution. We can widen the focus in the context of disjunctive plans. For example, in Figure 5, using the domain and planning theory, we can recognize that actions Load(A) and Fly(R) cannot both occur in the first step (since their preconditions and effects are interacting). Propagating this information tells us that the second state may either have In(A) or At(R,M), but not both. Here the interaction between the steps 1 and 3 propagates to make the conditions In(A) and At(R,M) "mutually exclusive" in the next disjunctive state. Thus any action which needs both In(A) and At(R,M) can be ignored at the next level. The particular strategy described here is similar to the one employed by Blum and Furst's [1995] Graphplan algorithm, and has been shown to be a major source of its efficiency [Kambhampati & Lambrecht, 1997]. Similar techniques need to be developed for plan-space refinements.

To summarize, disjunctive plans can be refined directly and efficiently, at the expense of some of the progressiveness (pruning power) of the refinement. The loss of pruning power can be countered by widening the scope of the refinements to infer more than just the potential actions in the solution.

4. Relating SATPLAN to disjunctive planning

The SATPLAN approach involves generating a SAT encoding, all models of which will correspond to *k*-length solutions to the problem (for some fixed integer *k*). Model-finding is done by efficient SAT solvers [Selman et. al., 1992; Crawford & Auton, 1996; Bayardo & Schrag, 1997]. Kautz et. al. propose to start with some arbitrary value of *k*, and increase it if they do not find solutions of that length. They have considered a variety of ways of generating the encodings, corresponding loosely to different traditional planning algorithms.

In the context of the general refinement planning framework, we can offer a rational basis for the generation of the various encodings. Specifically, the natural place where SAT solvers can be used in refinement planning is in the "solution extraction phase". As illustrated in Figure 7, after doing *k* "complete" and "strongly progressive" refinements on a null plan, we get a planset whose minimal candidates contain all *k*-length solutions to the problem. So, picking a solution boils down to searching through the minimal candidates-- which can be cast as a SAT problem. This account naturally relates the character of the encodings to the type of refinements used in coming with the k^{th}-level planset and how the plansets themselves are represented.

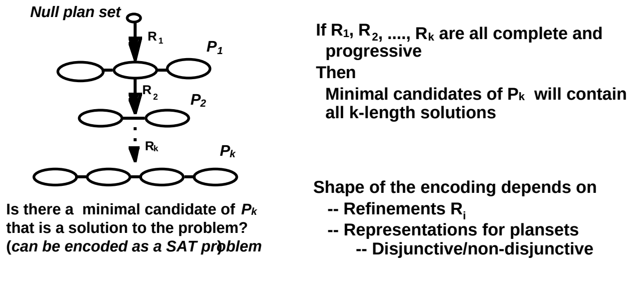
Null plan set
R_1
P_1
R_2
P_2
R_k
P_k
Is there a minimal candidate of P_k that is a solution to the problem?
(can be encoded as a SAT problem)
If R_1, R_2, ..., R_k are all complete and progressive
Then
Minimal candidates of P_k will contain all k-length solutions
Shape of the encoding depends on
-- Refinements R_i
-- Representations for plansets
-- Disjunctive/non-disjunctive

Figure 7. Relating refined plan at k-th level to SATPLAN encodings.

Kautz et. al. concentrate primarily on direct translation of planning problems to SAT encodings, sidelining refinement planning [Kautz et. al., 1996] (the only exception is their Graphplan based encoding; see below). I believe that such an approach confounds two orthogonal issues: (1) Reaching a disjunctive plan whose minimal candidates contain all the solutions and (2) Posing the solution extraction as a SAT problem.

It is best to separate these issues and see the main contributions of SATPLAN as pertaining to the second one. The issues guiding the first are by and large specific to planning, and are best addressed in terms of refining disjunctive plans. The issues guiding the second are general heuristics regarding the effective ways of compiling a combinatorial search problem into a SAT instance, and are only loosely tied to planning. The main concern here is to end up with a SAT instance that is "small" (e.g., in terms of number of variables). The techniques used to achieve this involve syntactic optimizations such as converting *n*-ary predicates into binary predicates, or compiling out dependent variables.

Separating planset construction and its compilation into a SAT instance allows SATPLAN techniques to exploit, rather than *re-invent*, (disjunctive) refinement planning. It also removes the arbitrariness involved in "guessing" the length of the solution as a pre-requisite for coming up with a SAT encoding, since solution extraction is interleaved with refinements. It may also be possible to develop heuristics that attempt to predict whether it is worth doing solution extraction on the current plan (as for example, Graphplan does).

In addition, I speculate that basing encodings on k^{th} level plansets may also lead to SAT instances that are "smaller" on the whole. Specifically, the pruning done by disjunctive refinements can also lead to a planset with fewer number of minimal candidates than can be achieved by direct encodings. This can in turn lead to a tighter SAT encoding. My speculation is supported to some extent by the results reported in [Kautz and Selman, 1996] which show that SAT encodings based on *k*-length planninggraphs generated by Graphplan, can be solved more efficiently than the "linear" encodings generated by direct translation.[1]

5. Challenge Problems

The framework of relations that we have outlined in this paper gives rise to several open issues that are worth investigating. I pose them below as challenge problems to the researchers in the planning community:

5.1 Disjunctive refinements

In Section 3, I argued that the success of Graphplan can be interpreted in terms of the idea of disjunctive planning. The only implemented and widely tested disjunctive planner is Graphplan, and it is based on forward state space refinements. Given the known disadvantages of forward state-space refinements *vis a vis* the other goal directed refinements, it seems likely that disjunctive planners based on these other refinements may scale-up even better. This leads to the first challenge:

Challenge 1. *Develop disjunctive planners based on (1.a) backward state space, (1.b) plan-space and (1.c) task reduction refinements, either working independently or in combination.*

Section 3 explicates some of the critical issues involved in implementing disjunctive planners, but a lot remains to be done. An easy first step in this direction would be to have a version of Graphplan algorithm that uses backward state space refinements (which can in principle out-perform Graphplan by being goal-directed). A more interesting step is to develop a disjunctive planner based on plan-space refinement, and see if the advantages of plan-space refinement over state-space ones in the traditional planners carry over to disjunctive planners. Given the dominance of task reduction refinements in practical planners, it is also imperative to develop disjunctive planners based on task reduction refinement. Finally, since interleaving refinements is known to have advantages in traditional planners [Kambhampati & Srivastava, 1996], it would be worth considering disjunctive planners that interleave refinements. The key in all these cases will be to handle the tradeoff between cost of the refinement and the loss of pruning power by generalizing the corresponding refinements to make them infer more than just action constraints. The discussion in Section 3 may suggest some ideas in this regard. Progress on the first challenge can put us at a good vantage point to answer a more global one:

Challenge 2. *Characterize tradeoffs offered by different refinements in supporting disjunctive planning.*

The most interesting issues to be resolved here include whether and how the tradeoffs between the various refinements change when we shift from traditional refinement planners to disjunctive planners (c.f. [Barrett & Weld, 1994]): Would plan-space and task-reduction refinements maintain their advantages over in disjunctive planning algorithms too, or would they be less effective (presumably because they do not support effective refinement and constraint propagation)? The answer may depend on how effectively the corresponding refinements can be generalized to disjunctive plans.

5.2 Issues in SAT compilation

Given that SAT compilation seems to be the most promising way of doing solution extraction on disjunctive plans, development of effective disjunctive planners will require attention to the SAT compilation issues. The problem-independent tradeoffs in SAT compilation are the subject of a challenge paper by Selman et. al. [1997], and will not be discussed here. Our interest here is to understand the specific compilation tradeoffs directly related to planning:

Challenge 3. *Characterize the tradeoffs offered by SAT-encodings based on disjunctive plans derived by different types of refinements (alone or working in an interleaved combination).*

The issues to be resolved here include understanding (1) how easy would it be to solve the SAT encodings resulting from different refinements and (2) how to effectively exploit the shared structure of the SAT encodings corresponding to successive levels of disjunctive plans. The first issue is especially interesting in light of the current experience that SNLP encodings, despite being more compact, are harder to solve [Selman et. al., 1997]. The second issue is motivated by the fact that solution extraction is interleaved with incremental refinement operations, and thus the disjunctive plans at successive levels share a significant amount of substructure. Indeed, one of the sources of efficiency for Graphplan algorithm is the way the backward search phase on successive planning-graphs can use the information computed in the previous levels (e.g. memoizing).

It would also be interesting to understand why SAT compilation is so effective in the first place. Kautz and Selman's [1996] original arguments seemed to imply that the effectiveness has a lot to do with stochastic search methods used in solving SAT problems. This implication been called into question by the recent results of Bayardo and Schrag [1997], which show that systematic approaches can do just as well. So, a pertinent question is: Is the compilation into SAT justified purely by modularity concerns (in that different planning algorithms do not have to worry about specialized solution extraction procedures), or is there any inherent ad-

[1] Kautz and Selman interpret the superior performance of graphplan encodings over linear encodings to the fact that the former allow action parallelism. However, as shown in [Kambhampati & Lambrecht, 1997], the real contribution of graphplan is not supporting parallel actions, but rather allowing forward state space refinements over disjunctive plans.

vantage to SAT representation? A partial attempt at resolving this issue would be to enrich the backward search (solution extraction) phase of the Graphplan algorithm with the CSP techniques such as dependency directed backtracking, to see if it could rival the SAT performance.

5.3 The utility of controlled splitting

In terms of our general refinement planning template, traditional planners split each of their component plans into a different search branch, while Graphplan and SATPLAN can be thought of in terms of not splitting the plansets at all. Research in constraint satisfaction and operations research shows that pruning and splitting can have synergistic interactions. In fact, the best CSP search strategies combine low-level constraint propagation (pruning) techniques, such as forward checking, with splitting [Bayardo & Schrag, 1997]. This raises the possibility that best disjunctive planners may also do controlled splitting of plansets (rather than no splitting at all) to facilitate refinements with more pruning power. Controlled splitting may also be helpful in scenarios where planning and execution need to be interleaved of necessity. Although some initial exploration has been done in this direction by Descartes and UCPOP-D planners, a more systematic analysis of tradeoffs is still needed. This leads to the next challenge:

Challenge 4. *Develop and explore the tradeoffs offered by planners that split some of the disjunction into the search space, while keeping the rest together in disjunctive format.*

One immediate question is exactly how many branches should a planset be split into? Since the extent of propagation depends on the amount of shared sub-structure between the disjoined planset components, one way of controlling splitting may be to keep plans with shared sub-structure together. Of particular theoretical interest would be exhibiting domains and problems where full disjunction is counter-productive. A first step in this direction would involve experimenting with versions of Graphplan algorithm that maintain multiple planning-graphs at each level [Kambhampati & Lambrecht, 1997].

5.4 Exploring alternative Representations

We have noted in Section 2.2 that there is nothing sacrosanct about the particular representation for partial plans that is used by most existing planners. Alternative representations may either be dictated by necessity (for example, the time-map based plan representations used in planners like HSTS [Muscettola, 1993] to support the use of expressive resource constraints in planning), or as more flexible ways of representing and handling sets of action sequences (for example, the representation used in COPS [Ginsberg, 1996]). The general framework and ideas described in this paper are independent of the specific representations used for partial plans. For example, Ginsberg's COPS planner can be seen as an instance of the disjunctive planning approach, that uses a very different partial plan representation (albeit with the same candidate set semantics). The approach shares the ideas of direct refinement of disjunctive plans and solution extraction by enumerating minimal candidates. Representations can however have a significant impact on the kinds of refinements that can be supported. Ginsberg describes a type of refinement on his plan representation that has the ability compute both upper and lower approximations on the set of solutions. This leads us to an interesting, but somewhat open-ended challenge.

Challenge 5. *Develop and investigate the utility of alternative plan representations on the efficiency of plan generation.*

While development of new representations is an open-ended task, a first feasible step in this direction would be to do a direct empirical comparison of currently available alternative representations, such as those used in COPS or HSTS. It would also be interesting to undertake each of the previous challenges in the context of the alternative representations.

6. Conclusion

In this paper, I presented a generalized view of refinement planning that bridges several hither-to distinct strands of planning research. I have outlined how the insights from this unification can help both traditional and SAT-based planners. Finally, I have proposed as a challenge to the community, a set of research problems, tackling which can lead to a significantly improved understanding of the issues involved in efficient plan synthesis.

To support research into these challenge problems, I intend to maintain a web site that will act as a clearing house for information on our evolving understanding. Its URL will be *http://rakaposhi.eas.asu.edu/challenge.html*. This site will contain references to other relevant literature, pointers to benchmark problems and test-domains, as well as more fleshed out versions of the challenges and their up-to-date status.

References

Barrett, A. and Weld, D. 1994. Partial Order Planing: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71-112.

Bayardo, R and Schrag, R. 1997. Using CSP look-back techniques to solve real-world SAT instances. In Proc. AAAI-97.

Blum, A. and Furst, M. 1995. Fast planning through plan-graph analysis. In Proc. IJCAI-95.

Crawford, J. and Auton, L. 1996. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81.

Ginsberg, M. 1996. A new algorithm for generative planning. In Proc. KR-96.

Joslin, D and Pollack, M. 1996. Is least commitment always a good idea? In Proc. AAAI-96.

Kambhampati, S. and Srivastava, B. 1996. Unifying classical planning approaches. ASU CSE TR 96-006. (Preliminary version appeared in Proc. 3rd European workshop on planning).

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(1-2):167-238.

Kambhampati, S. and Yang, X. 1996. On the role of disjunctive representations and constraint propagation in refinement planning. In Proc. KR-96.

Kambhampati, S and Lambrecht, E. 1997. Why does Graphplan work? ASU CSE TR 97-005. (Poster at IJCAI-97)

Kambhampati, S. 1997. Refinement planning as a unifying framework for plan synthesis. AI Magazine. Summer issue.

Kautz, H. and Selman, B. 1996. Pushing the envelope: Planning Propositional Logic and Stochastic Search. In Proc. AAAI-96.

Kautz, H., McAllester, D. and Selman, B. 1996. Encoding plans in propositional logic. In Proc. KR-96.

Muscettola, N. 1993. HSTS: Integrating planning and Scheduling. In: *Intelligent Scheduling.* M. Fox and M. Zweben (eds). Morgan Kaufmann.

Pearl, J. *Heuristics*. Addison-Wesley. 1984.

Selman, B., Levesque, H.J., and Mitchell, D. 1992. GSAT: a new method for solving hard satisfiability problems. In Proc. AAAI-92.

Selman, B., Kautz, H. and McAllester, D. 1997. Computational challenges in propositional reasoning and search. In Proc. IJCAI-97.