A Candidate Set based analysis of Subgoal Interactions in conjunctive goal planning

Subbarao Kambhampati*, Laurie Ihrig and Biplav Srivastava
Department of Computer Science and Engineering
Arizona State University, Tempe AZ 85287-5406
email: { rao , biplav , ihrig }@asu.edu
WWW: http://sod90.asu.edu:8001/yochan.html

Abstract

Subgoal interactions have received considerable attention in AI Planning. Earlier analyses by Korf [11] and Joslin and Roach [6] were done in terms of the topology of the space of world states. More recent analyses by Barrett and Weld [1] and Veloso and Blythe [14] were done in terms of the nature of the planner. In this paper, we will argue that subgoal interactions are best understood in terms of the candidate sets of the plans for the individual subgoals. We will describe a generalized representation for partial plans that applies to a large class of refinement planners, and discuss the notion of mergeability and serial extensibility of these partial plans. The concepts of independence and serializability of subgoals are derived by generalizing mergeability and serial extensibility over classes of partial plans. Unlike previous work, our analysis also applies to multi-method refinement planners such as UCP [7]. We will show that all existing characterizations of serializability differ only in terms of the specific class of partial plans that they implicitly address. Finally, we will use our interaction analysis to explore factors affecting the selection of a refinement planner for a given domain.

1 Introduction

One of the important ways of characterizing the complexity of planning in a domain involves analyzing the interrelation between the subgoals that arise in the problems in that domain. Korf's 1987 paper [11] characterized a set of subgoals as independent, serializable or non-serializable, and argued that the cost of planning increases as the subgoals go from independent to non-serializable. Intuitively, independence means that the subgoals can be achieved independently and their plans concatenated to form a plan for the conjunctive goal; while serializability means that there exists some ordering of the subgoals in which we can achieve the first goal, and then achieve the second one without ever passing through a state of the world where the first one does not hold. Korf's definition of serializability is couched in the state-space terminology and thus does not readily generalize to planners that do not search in the space of states. Barrett and Weld [1] extended Korf's analysis by shifting the emphasis from the state space characterization to the nature of the planner that is used to solve the problems. They showed that a given set of subgoals may be serializable for a plan space planner while being non-serializable for a state space planner. Veloso and Blythe extend this planner based analysis of subgoal interactions by defining a complementary characterization of subgoal interactions they call ''linkability'' [14], arguing that a given problem may be easily linkable for state space planners, while being laboriously linkable for plan space planners.

Given that there are a great variety of refinement planners (c.f. [10, 7]), characterization of subgoal interactions that depend on the details of the planners tie us to specific brand-name planners in terms of which the analysis is done, and make it hard to see the essential structure of the interactions. It would be better if the analysis framework were independent of planner details, and be dependent only on some common currency, such as the subgoals and the types of plans generated for them. Such an analysis has hitherto been impossible since the plan representations used by different types of planners have been incomparable.

More recently, as part of our efforts to develop a common framework for refinement planning [10, 7], we have developed a general representation for partial plans that is suitable for most classical refinement planners, including state-space, plan-space, goal protection oriented, and means-ends analysis planners. Semantics for these partial plans are provided in terms of candidate sets, i.e., sets of ground operator sequences that are consistent with the constraints on the partial plans.

Since subgoals interact in terms of their subplans, we will start by characterizing two important ways -- mergeability and serial extensibility -- in which partial plans may interact. The concepts of independence and serializability of subgoals are derived by generalizing mergeability and serial extensibility over classes of partial plans. We will then show that all existing characterizations of serializability can be cast as specializations of the notion of serializability developed here, and differ only in terms of the specific class of partial plans that they implicitly address. Our analysis is focused on partial plans as opposed to details of the plan-

*This research is supported in part by NSF research initiation award (RIA) IRI-9210997, NSF young investigator award (NYI) IRI-9457634 and ARPA/Rome Laboratory planning initiative grants F30602-93-C-0039 (Phase 2) and F30602-95-C-0247 (Phase 3). We Eric Jacopin and the AIPS reviewers for their critical comments.

ning algorithm and defines subgoal interactions in terms of the plans used to achieve individual subgoals. We will end with a discussion of how our analysis can help in selecting refinement planners for a given domain.

The rest of this paper is organized as follows. The next section contains an overview of our planning framework. It describes a representation for partial plans that supports a variety of classical refinement planners, and discusses how state-space and plan-space refinements can be modeled on top of it. Section 3 describes two candidate set based characterizations of subgoal interactions called mergeability and serial extensibility, and discusses their properties. Section 4 generalizes the analysis for individual plans over classes of subplans for the subgoals. Section 5 describes how mergeability and serial extension cover the previous analyses of subgoal interactions Section 6 discusses how the analysis of subgoal interactions can be useful in selecting an appropriate refinement planner given a domain. Section 7 summarizes the contributions of the paper.

2 Overview of Refinement Planning

A planning problem is a 3-tuple $\langle I, G, \mathcal{A}\rangle$, where I is the description of the initial state, G is the (partial) description of the goal state, and $\mathcal{A}$ is the set of actions (also called ''operators''). An action sequence (also referred to as ground operator sequence) S is said to be a solution for a planning problem, if S can be executed from the initial state of the planning problem, and the resulting state of the world satisfies all the goals of the problem.

Refinement planners [10, 8] attempt to solve a planning problem by navigating the space of *sets of potential solutions (ground operator sequences)*. The potential solution sets are represented and manipulated in the form of ''partial plans.''[1] Syntactically, a partial plan $\mathcal{P}$ can be seen as a set of constraints. Semantically, a partial plan is a shorthand notation for the set of ground operator sequences that are consistent with its constraints. The latter set is called the *candidate set* of the partial plan, and is denoted by $\langle\langle\mathcal{P}\rangle\rangle$.

Refinement planning consists of starting with a ''null plan'' (denoted by $\mathcal{P}_\emptyset$, whose candidate set corresponds to all possible ground operator sequences) and successively refining the plan (by adding constraints, and thus splitting their candidate sets) until a solution is reached. Semantically, a *refinement operator* $\mathcal{R}$ maps a partial plan $\mathcal{P}$ to a set of partial plans $\{\mathcal{P}'_i\}$ such that the candidate sets of each of the children plans are proper subsets of the candidate set of $\mathcal{P}$ (i.e., $\forall_{\mathcal{P}'_i} \langle\langle\mathcal{P}'_i\rangle\rangle \subset \langle\langle\mathcal{P}\rangle\rangle$). Refinement planning involves repeatedly applying refinement operators to a partial plan until a solution can be picked up from the candidate set of the resulting plan.

A refinement operator $\mathcal{R}$ is said to be **complete** if every solution belonging to the candidate set of $\mathcal{P}$ belongs to the candidate set of at least one of the children plans. It is easy to see that as long as a planner uses only refinement operators that are complete, it never has to backtrack over the application of a refinement operator. In [7], we showed that state-space planning and plan-space planning approaches can essentially be modeled as different varieties of refinement operators operating on the same partial plan representation.

[1] For a more formal development of the refinement search semantics of partial plans, see [8, 10]

2.1 Representation of partial plans

In this section, we will develop a syntactic and semantic representation of partial plans that is adequate to support both state-space and plan-space refinements.

A partial plan is a 5-tuple $\langle T, O, \mathcal{B}, \mathcal{ST}, \mathcal{L}\rangle$ where: T is the set of steps in the plan; T contains two distinguished step names t_0 and t_∞. $\mathcal{ST}$ is a symbol table, which maps step names to actions. The special step t_0 is always mapped to the dummy operator `start`, and similarly t_∞ is always mapped to `finish`. The effects of `start` and the preconditions of `finish` correspond, respectively, to the initial state and the desired goals of the planning problem. O is a partial ordering relation over T. $\mathcal{B}$ is a set of codesignation (binding) and non-codesignation (prohibited binding) constraints on the variables appearing in the preconditions and post-conditions of the operators. A ground linearization of $\mathcal{P}$ is a permutation on its steps that is consistent with O, with all variables instantiated to values that are consistent with $\mathcal{B}$. $\mathcal{L}$ is a set of auxiliary constraints that restrict the allowable orderings and bindings among the steps. Three important types of auxiliary constraints are:

Interval Preservation Constraints: An *interval preservation constraint* (IPC) is specified as a 3-tuple: $t \overset{p}{-} t'$. Syntactically, it demands that the condition p be preserved between t and t' in every ground linearization of the plan. Semantically, it constrains the candidates of the partial plan such that in each of them, p is preserved between the operators corresponding to steps t and t'.

Point Truth Constraints: A *point truth constraint* (PTC) is specified as a 2-tuple: $\langle p@t\rangle$. Syntactically, it demands that the condition p be necessarily true [2] in the situation before the step t. Semantically, it constrains all solutions of the partial plan to have p true in the state in which the operator corresponding to t is executed.

Contiguity Constraints: A contiguity constraint is specified as a relation between two steps: $t_i * t_j$. Syntactically, it demands that no step intervene between t_i and t_j in any ground linearization of $\mathcal{P}$. Semantically, it constrains all candidates of the partial plan such that nothing intervenes between the operators corresponding to t_i and t_j.

Example: Figure 1 illustrates these definitions through an example plan $\mathcal{P}_E$, which contains the steps $\{t_0, t_1, t_2, t_3, t_4, t_5, t_\infty\}$, the symbol table $\{t_0 \rightarrow \texttt{start}, t_\infty \rightarrow \texttt{end}, t_1 \rightarrow o_1, t_2 \rightarrow o_2, t_3 \rightarrow o_3, t_4 \rightarrow o_4, t_5 \rightarrow o_5\}$, the ordering constraints $\{t_1 \prec t_2, t_2 \prec t_3, t_3 \prec t_5, t_1 \prec t_4, t_1 \prec t_5, t_4 \prec t_5\}$, the contiguity constraints $\{t_0 * t_1, t_5 * t_G\}$, the interval preservation constraints $\{t_1 \overset{q}{-} t_2, t_3 \overset{r}{-} t_\infty\}$, and the point truth constraints $\{\langle r@t_\infty\rangle, \langle u@t_\infty\rangle, \langle v@t_\infty\rangle, \langle w@t_\infty\rangle\}$. The ground operator sequence $o_1o_2o_4o_3o_2o_2o_5$ is a candidate of the plan $\mathcal{P}_E$. *Notice that the candidate of a partial plan may contain operators that are not present in the partial plan, as long as those operators do not violate the auxiliary constraints.* In contrast, the ground operator sequences $o_3o_1o_2o_3o_3o_5$ and

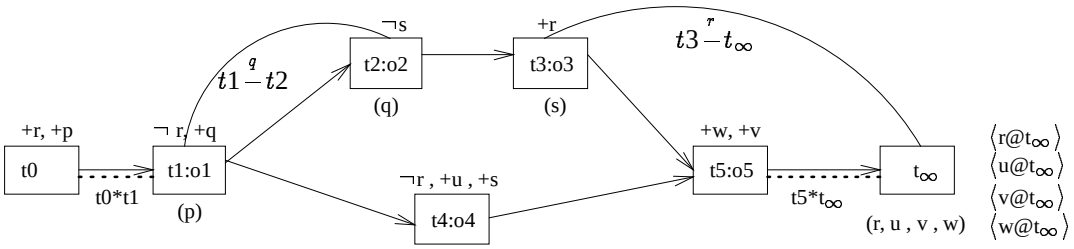

Figure 1: Example Partial Plan $\mathcal{P}_E$. The effects of the steps are shown above the steps, while the preconditions are shown below the steps in parentheses. The ordering constraints between steps are shown by arrows. The interval preservation constraints are shown by arcs, while the contiguity constraints are shown by thick dotted lines. The PTCs are used to specify the goals of the plan

$o_1o_2o_3o_4o_5$ are not candidates of $\mathcal{P}_E$ (the former violates the contiguity constraint $t_0 * t_1$, and the latter violates the interval preservation constraint $t_3 \overset{r}{-} t_\infty$).

2.2 Goal Achievement

A ground linearization is said to be a **safe ground linearization** if it syntactically satisfies all the contiguity constraints, and the interval preservation constraints [10]. The semantic notion of the candidate set of the partial plan is tightly related to the syntactic notion of safe ground linearizations [10, 8]. Specifically, safe ground linearizations correspond to minimal length candidates (or simply ''**minimal candidates**'') of the partial plan [10]. If a partial plan has no safe ground linearizations, it has an empty candidate set.

In the example in Figure 1, $t_0t_1t_2t_4t_3t_5t_\infty$ is a safe ground linearization, while $t_0t_1t_2t_3t_4t_5t_\infty$ is not a safe ground linearization (since the interval preservation constraint $t_3 \overset{r}{-} t_\infty$ is not satisfied by the linearization). The safe ground linearization corresponds to the minimal candidate (ground operator sequence) $o_1o_2o_4o_3o_5$.

A ground operator sequence G is said to achieve a goal g_1 if executing G from the initial state leads to a world state where g_1 is true. A partial plan P is said to achieve a goal g_1 when all its ground linearizations are safe ground linearizations, and the operator sequences corresponding to them achieve g_1.

Notice the dual view of plans that is being developed here. The solution-ness and the execution semantics of a plan P are defined in terms of its minimal candidates alone, while its refinements are defined in terms of its full candidate set. The candidate set of P may contain possibly infinitely more other ground operator sequences each of which may or may not be solutions.[2]

2.3 Refinement of Partial Plans

As mentioned earlier, a partial plan can be refined by replacing it with a set of partial plans whose candidate sets are subsets of the original plan. Three well known refinement operations are: Forward State Space Refinement (FSS), Backward State Space Refinement (BSS) and Plan Space Refinement (PS). FSS involves growing the prefix of a plan by introducing a contiguity constraint between the last step of the current prefix and any operator o (either currently in the plan, or taken from the library) that will be applicable in the state of the world after the steps in the current prefix are executed. An FSS refinement of the example plan shown in Figure 1 will be to make the operator o_2 (either the instance in the plan t_2 or a fresh instance from outside) contiguous to the step t_1 (i.e., put a constraint $t_1 * t_2$). This is allowed since the preconditions of o_2 will be true in the state after the execution of $t_1 : o_1$ in the initial state. The BSS refinement is similar to the FSS refinement, except that it considers all feasible suffixes of the plan. In the example in Figure 1, a BSS refinement will involve putting a contiguity constraint between t_3 and t_5. Finally, a plan space refinement involves establishing a precondition of a step with the help of existing or new steps, and introducing sufficient number of ordering, binding and IPC constraints to ensure this establishment. An example of plan space refinement of the plan in Figure 1 will be to use the effect s of the step t_4 to establish the precondition s of the step t_3. This will involve adding the constraints $t_4 \prec t_3$, and the constraint $t_2 \prec t_4$ (so that t_2 will not violate the established condition). Optionally, an IPC $t_4 \overset{s}{-} t_3$ can also be added to the plan to *protect* this establishment with respect to further step additions. A more complete discussion of these three types of refinements and their many variations can be found in [7].

If a planner uses FSS or BSS refinement alone, it is called a pure state-space planner, and if it uses a PS refinement alone, it is called a pure plan-space planner. Since all three refinements are complete -- in that they split the candidate set of a plan without losing any potential solutions, we can use them all within a single problem solving episode. This removes the strong distinctions between state-space and plan-space planners, and allows planners that use multiple refinements. In [7], we describe a planner called UCP that allows arbitrary interleaving of refinements, and show that it leads to performance improvements over pure state-space and plan-space planners.

2.4 Classification of Partial Plans

The plan representation discussed in this section is fairly general to allow many varieties of plans. In the following

[2] In [4], Ginsberg argues for an elimination of this separation saying that a ''good'' plan will have a large candidate set most of which will be able to achieve the goals of the problem.

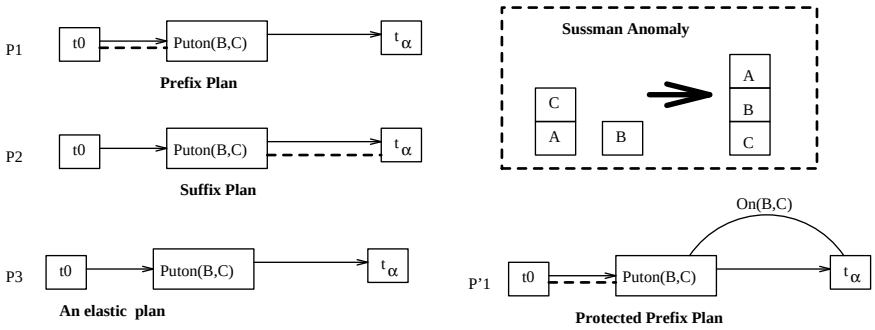

Figure 2: The Sussman Anomaly problem, and a variety of plans for solving the subgoal $On(B, C)$.

we identify subclasses of partial plans which have interesting properties from subgoal interaction point of view. The subclasses will be identified in terms of the syntactic restrictions on the plan constraints (see Figure 2).

A plan P for achieving a goal g from the initial state I is called **prefix plan** if all the steps of the plan, except t_∞, are all contiguous. Any feasible prefix plan will also have the property that the prefix steps can all be executed from the initial state in sequence (as otherwise, their candidate sets cannot contain any executable operator sequences). For example, a prefix plan for the subgoal $On(B, C)$ in the sussman anomaly problem is P_1: $t_0 * Puton(B, C) \prec t_\infty$. Planners that do forward search in the space of states produce feasible prefix plans.

Similarly, a plan is said to be a **suffix plan** if the steps of the plan except t_0 are contiguous to each other. Any feasible suffix plan will have the property that the result of regressing the goal state through the plan suffix is a feasible state. A suffix plan for $On(B, C)$ is P_2: $t_0 \prec Puton(B, C) * t_\infty$. Suffix plans are produced by planners that do backward search in the space of states.

Planners that search in the space of plans typically generate plans in which actions are ordered only by precedence (''$\prec$'') relations. Since any arbitrary number of new steps can come in between two steps ordered by a precedence relation, we shall call such plans ''**elastic plans**.'' An elastic plan for the subgoal $On(B, C)$ in sussman anomaly is P_3: $t_0 \prec Puton(B, C) \prec t_\infty$.

A plan P for a subgoal g is called a **protected plan** if it has IPC constraints to protect the subgoal as well as every precondition of every step of P. Protection of a condition c at a step s is done by adding an IPC $s' \overset{c}{-} s$ where s' is a step in P which gives the condition c to s. For example, a protected prefix plan for $On(B, C)$ in sussman anomaly is P_1' : $t_0 * Puton(B, C) \prec t_\infty$ with the IPCs $Puton(B, C) \overset{On(B,C)}{-} t_\infty, t_0 \overset{clear(B)}{-} Puton(B, C), t_0 \overset{Clear(C)}{-} Puton(B, C)$. (Note that since the partial plan is a prefix plan, no new steps can come before $Puton(B, C)$ in the plan. Thus, the last two IPCs are redundant, as they can never be violated by new steps.)

Finally, it is useful to distinguish a fourth kind of plan that we call **blocked plan**. A blocked plan P for a goal g_1 contains contiguity constraints that set the absolute distance between every pair of steps in P except the t_0 and t_∞ steps. As an example, a blocked plan for the subgoal $On(A, B)$ in Sussman Anomaly problem will be $t_0 \prec Puton(C, Table) * Puton(A, B) \prec t_\infty$.

3 Candidate Set Based definitions of Subplan Interactions

Given two goals g_1 and g_2 to be achieved conjunctively from an initial state I, and specific subplans for achieving either of those goals, there are two scenarios in which the combinatorics of finding a plan achieving both goals is controlled:

1. We can find a plan P_1 for g_1 and a plan P_2 for g_2 independently, and then merge the plans together to produce a plan for g_1 and g_2. When subplans are mergeable this way, then we can parallelize the planning effort by working independently on the subgoals first and then working on merging the plans.
2. We can find a plan P_1 for g_1 that can be refined into a new plan for achieving both g_1 and g_2 without violating any commitments made in P_1. When serial extension is possible, we will essentially be able to work on the second subgoal without ever having to backtrack on (i.e., undoing) the refinements made in planning for the first subgoal.

We will capture these two notions from a candidate set perspective through the definitions of mergeability and serial extension below.

Definition 1 (Mergeability) *We say that a plan P_1 for achieving a goal g_1 from an initial state I is mergeable with respect to a plan P_2 for achieving goal g_2 if there is a plan P' that achieves both g_1 and g_2 (from the same initial state), and $\langle\langle P' \rangle\rangle \subseteq \langle\langle P_1 \rangle\rangle \cap \langle\langle P_2 \rangle\rangle$. (Thus syntactically, P' contains all the constraints of P_1 and P_2).*

In addition, the plans are said to be **simple mergeable** *if every step in P' is present in either P_1 or P_2 (i.e., P' does not contain any new steps), and the number of steps in P' is the sum of number of steps in P_1 and P_2. Finally, the plans are said to be* **trivial mergeable** *if P' contains no more or less constraints (steps, orderings, bindings) than P_1 and P_2.*

In general, merging two plans involves either combining steps in the plans being merged, or adding steps that are not present in either of them. Simple mergeability essentially ensures that the plans can be merged without adding any new steps, or combining existing steps (thus bounding

the amount of effort required in the merging phase). In contrast, the merging involving addition and combination of steps can be as costly as planning itself [15]. Even simple mergeability can lead to costly merging phases (as a possibly exponential number of combined linearizations of the two plans need to be considered). Trivial mergeability is the most restrictive as it requires that the merging operation only involve unioning the constraint sets of the two plans.

To illustrate these ideas, consider the blocks world situation where we have four blocks A, B, C and D all on table in the initial state. The plan $t_0 \prec Puton(A, B) \prec t_\infty$ for subgoal $On(A, B)$ is trivial mergeable with the plan $t_0 \prec Puton(C, D) \prec t_\infty$ for $On(C, D)$. In contrast, the plan $t_0 \prec Puton(A, B) \prec t_\infty$ for $On(A, B)$ is simple mergeable with the plan $t_0 \prec Puton(B, C) \prec t_\infty$ for $On(B, C)$ (but not trivial mergeable). The plan $t_0 * Puton(A, B) \prec t_\infty$ for $On(A, B)$ is not simple mergeable with the plan $t_0 \prec Puton(B, C) \prec t_\infty$ for $On(B, C)$, although it is mergeable. This is because the only way of merging these plans will be to insert additional steps giving rise to a plan such as $t_0 * Puton(A, B) \prec \underline{Puton(A, Table)} \prec Puton(B, C) \prec \underline{Puton(A, B)} \prec t_\infty$.

Finally, an example of mergeability that requires combining steps in the plans being merged is the ''one-way rocket'' problem which involves transporting objects A and B from the earth to the moon with the help of a single one-way rocket. The plan for taking object A to the moon will be $P_1 : t_0 \prec load(A) \prec Fly(Rocket) \prec unload(A) \prec t_\infty$, and the plan for taking object B to Moon will be $P_2 : t_0 \prec load(B) \prec Fly(Rocket) \prec unload(B) \prec t_\infty$, However, merging P_1 and P_2 to solve both the goals requires combining the two instances of $Fly(Rocket)$, since every complete plan can only have one instance of $Fly(Rocket)$.

Next, we will consider the idea of serial extention:

Definition 2 (Serial Extension) *We say that a plan P for achieving g_1 from a given initial state I (i.e., executing P from I will get us to a state where g_1 is true) is serially extensible with respect to a second goal g_2 if $\langle\langle P \rangle\rangle \cap L(g_1 \wedge g_2) \neq \emptyset$, where $L(g_1 \wedge g_2)$ is the set of all ground operator sequences that can achieve g_2 from the initial state.*

Any plan P' whose candidate set is a subset of $\langle\langle P \rangle\rangle \cap L(g_1 \wedge g_2)$ will achieve both g_1 and g_2. Since all candidates of P' are also candidates of P, P' has all the constraints of P (plus more). Thus, we never have to backtrack over any refinements that lead to P in coming up with P'.

Continuing the three block stacking example, the plan $t_0 \prec Puton(A, B) \prec t_\infty$ for $On(A, B)$ is serially extensible with respect to the goal $On(B, C)$, but the plan $t_0 \prec Puton(B, C) * t_\infty$ for $On(B, C)$ is not serially extensible with respect to the subgoal $On(A, B)$. To see the latter, note that no solution for sussman anomaly can have $Puton(B, C)$ as the final step.

3.1 The Role of Planner vs. Partial Plans in subplan interactions

Perhaps surprisingly, our characterization of subgoal interactions shifts the attention from the type of planner to the nature of partial plans that are being merged or extended. The critical role played by the ''planner'' is in coming up with the plan for the initial subgoal. If the candidate set of that plan does contain a solution for the two goals together, then any refinement planner which uses only complete refinement operators -- be they forward state space, backward state space, plan-space or a combination thereof -- will be able to extend the plan. This distinction may seem artificial given that most traditional planners use the same refinement strategy to generate the first subplan as well as to extend it. However, the distinction becomes more useful when we consider planners which can use multiple refinements, such as UCP [7].

For example, we noted that the plan $P_1 : t_0 \prec Puton(B, C) \prec t_\infty$ for subgoal $On(B, C)$ in the sussman anomaly is serially extensible with respect to subgoal $On(A, B)$. This means that any complete refinement -- including forward and backward state space refinement strategies -- can extend this plan into a solution. To illustrate, here is a series of forward state space refinements that will convert P_1 into a solution. (i) Apply an instance of the operator $Puton(C, Table)$, to the head state giving $P_2 : t_0 * Puton(C, Table) \prec Puton(B, C) \prec t_\infty$. (ii) Apply the $Puton(B, C)$ step to the head state of P_2 giving $P_3 : t_0 * Puton(C, Table) * Puton(B, C) \prec t_\infty$ and finally (iii) Apply an instance of the step $Puton(A, B)$ to the head state of P_3 giving rise to a solution. It is *not* the case that we need a plan space refinement for this purpose.

4 Characterizing Subgoal interactions

Mergeability and serial extension are defined in terms of specific plans for individual subgoals. Since a particular subgoal may have a variety of plans, given two subgoals g_1 and g_2, some of the plans for g_1 may be mergeable with some of the plans of the second subgoal g_2, while some others may be serially extensible with respect to g_2. In order to exploit the computational advantages of mergeability and serial extension, we need to consider ''all possible'' plans of g_1 (and g_2). Since, as discussed in Section 2.4, there are in general a variety of partial plans and, depending on the refinements one uses, only a subset of these plans may actually be realized by a refinement planner, it makes more sense to qualify the claims with respect to a class of plans, as we do below:

Definition 3 (Parellelizability) *We will say that two subgoals g_1 and g_2 are parallelizable modulo a class of plans $\mathcal{P}$, if each plan $P_1 \in \mathcal{P}$ for achieving g_1 is mergeable with any plan $P_2 \in \mathcal{P}$ that achieves g_2.*

The subgoals g_1 and g_2 are said to be **simple parallelizable** *modulo the class of plans $\mathcal{P}$ if any plan of g_1 in $\mathcal{P}$ is simple mergeable with any plan of g_2 in $\mathcal{P}$ to give rise to a plan for g_1 and g_2. The subgoals are* **trivial parallelizable** *if the plans for the subgoals are trivially mergeable.*

The subgoals g_1 and g_2 are said to be **optimal parallelizable** *modulo the class of plans $\mathcal{P}$ if any optimal plan of g_1 from $\mathcal{P}$ is mergeable with any optimal plan of g_2 of $\mathcal{P}$ to give rise to an optimal plan for g_1 and g_2.*

From the complexity point of view, parellelizability allows us to use divide-and-conquer approaches for planning. If g_1 and g_2 are parallelizable, then the cost of solving

the conjunctive goal is additive in the cost of solving the individual goals, plus the cost of merging the plans. However, parallelizability does not in itself imply that actually parallelizing the goals is either efficient (since the merging phase can be costly) or desirable (since the merged plan may be in optimal). For parallelization to be a win, the cost of merging should be small. The cost depends upon the type of merging (trivial, simple or non-simple). While trivial mergeability takes constant time, and simple mergeability can be NP-hard [15], merging involving the addition and deletion of tasks can be as costly as planning itself.

Given any domain where all the actions are reversible, any pair of subgoals from that domain will be parallelizable (since we we can always ''undo'' the actions of the individual plans for both the goals and then add actions to find a correct plan for the overall problem). Of course, this not only makes the merging step costlier than the original planning problem, but also leads to very inefficient plans. Thus, for parallelizability to be desirable, we need to have the guarantee that the divide and conquer approach will find ''optimal plans'' for the conjunctive goal by starting from optimal plans for the individual goals. This leads to the notion of optimal parallelizability. The optimality and efficiency restrictions on parallelizability can of course be combined. In fact, Korf's definition of subgoal independence [11], implies optimal and trivial parallelizability of all subgoals.

Definition 4 (Serializability) *Given a class $\mathcal{P}$ of plans, we will say that g_1 is* **serializable** *with respect to g_2 modulo $\mathcal{P}$ if every plan $P_1 \in \mathcal{P}$ of g_1 is serially extensible with respect to g_2.*

Serializability does not necessarily give rise to savings in planning effort. The main reason is that while parellelizability is a commutative relation, serializability is non-commutative. It is possible for g_1 to be serializable with respect to g_2 but for g_2 not to be serializable with respect to g_1. Thus for the planner to be able to exploit serializability, it needs to work on g_1 first and then on g_2. When there are multiple subgoals, the chance of picking the correct goal order is low and thus serializability does not imply improvements in planning cost. Following Barrett and Weld [1], we thus further extend the notion of serializability to consider *trivial* and *laborious* serializability.

Definition 5 (Serialization Order [1]) *Given a set of n subgoals $g_1, g_2 \ldots g_n$, a permutation π on these subgoals is considered a serialization order (modulo a class of plans $\mathcal{P}$), if every plan for achieving $\pi[1]$ can be serially extended to $\pi[2]$ and any resulting plan can be serially extended $\pi[3]$ and so on.*

The set of subgoals are considered trivially serializable if all the permutations correspond to serialization orders, and are considered laboriously serializable if a significant number of permutations ($> \frac{1}{n}$) correspond to non-serialization orderings.

Relation Between Serializability and Parallelizability: Finally, it is instructive to note that while any form of parallelizability implies serializability, even trivial serializability does not guarantee any form of parallelizability. To see this, consider a simple domain with only two goals g_1 and g_2 and four operators defined below:

A trivially serializable but un-parallelizable domain			
Op	Prec	Add	Del
O_1	p	g_1	w
O_1'	r	g_1	q
O_2	w	g_2	p
O_2'	q	g_2	r

Suppose in a given problem, the initial state contains p, q, r and w and we want to achieve g_1 and g_2. It is easy to see that if g_1 is achieved by O_1 then we cannot achieve g_2 using O_2 and have to use O_2'. Thus not all plans of g_1 are mergeable with all plans of g_2. However, g_1 and g_2 are trivially serializable since any plan for g_1 or g_2 can be extended into a plan for both goals.

5 Relation to existing work

Readers familiar with previous efforts on characterization of subgoal interaction will note that our notions of serializability and parallelizability are defined modulo a class of plans. In this section, we will explain how our characterization subsumes the existing work by identifying the specific classes of plans over which the existing characterizations of subgoal interactions are implicitly based.

5.1 Korf's Subgoal Interactions

Korf [11] defines two subgoals to be serializable if there exists an ordering among the subgoals such that they can be planned for sequentially, such that once the first goal is achieved, the agent never passes through a state where the first goal is violated.[3]

The Sussman anomaly has non-serializable subgoals according to this definition. For example, suppose we work on $On(B, C)$ first and then $On(A, B)$. A state in which $On(B, C)$ is true is: S: $On(B, C) \wedge On(C, A)$. However, we cannot go from S to any goal state without violating $On(B, C)$.[4]

The following proposition shows that Korf's definition can be seen as a special case of our subgoal serializability for the class of protected prefix plans.

Proposition 1 (Korf's Serializability) *Two subgoals g_1 and g_2 are Korf-Serializable, if they are serializable with respect to the class of protected prefix plans.*

[3] Joslin and Roach [6] give a similar analysis of subgoal interactions in terms of the state space graph. In particular, they consider the state transition graph of the domain, and identify each subgoal with a subgraph of the transition graph (where all the states in that subgraph satisfy that goal). These subgraphs in general may contain several connected components. The set of subgoals is said to be nonlinear if any of the subgraphs corresponding to the subgoals have a connected component that does not contain a goal state. The idea is that if the planner finds itself in a component of the sub-graph of the goal, it cannot achieve the second goal without undoing the first.

[4] There is of course another state $On(B, C) \wedge On(C, Table) \wedge On(A, Table)$ from which we can reach a solution state without violating $On(B, C)$. However, serializability requires that this be true of *every* state that has $On(B, C)$ true.

The qualification about ''protected plans'' is needed as Korf requires that the goal g_1 remains achieved while P_1 is being extended. The ''prefix plan'' qualification is needed since Korf's analysis assumes that the search is being done in the space of world states and that the plan for the first goal takes us to a completely specified world state. Indeed, in sussman anomaly, the plan $t_0 \prec Puton(B,C) \prec t_\infty$, with the IPC $Puton(B,C) \overset{On(B,C)}{-} t_\infty$ is serially extensible with respect to $On(B,C)$; although the plan $t_0 * Puton(B,C) \prec t_\infty$, with the same IPC is not.

5.2 Barrett and Weld's Serializability

Barett and Weld [1] extended Korf's [11] subgoal interaction analysis to plan space planners, and showed that problems like sussman anomaly are serializable for the partial order planner SNLP. Although their analysis concentrated on specific planning algorithms, it can also be understood in terms of the class of plans with respect to which serializability is being defined. Specifically, we have:

Proposition 2 (Barrett and Weld's Serializability) *Two subgoals g_1 and g_2 are serializable by Barrett and Weld's definition if they are serializable with respect to the class of protected elastic plans.*

Since prefix plans have smaller candidate sets than elastic plans with the same set of steps, the latter naturally have higher likelihood of being serially extensible with the second goal. Indeed, the sussman anomaly problem is ''serializable'' for the class of elastic plans.

5.3 Relaxing the protected plan requirement

We saw that both Korf's and Barrett & Weld's notions of serializability implicitly involve protected plans, which post IPCs to protect the establishment of all the preconditions in the plan. Relaxing the requirement for protected plans leads to classes of problems that may not be serializable for the class of protected plans, but are serializable for the class of unprotected plans. This should not be surprising, given the discussion above, since everything else being equal, plans with IPCs have higher commitment (and thus smaller candidate sets) than plans without IPCs.

Note that the sussman anomaly problem is indeed serializable for the class of un-protected prefix plans. In particular, the plan $t_0 * Puton(B,C) \prec t_\infty$ is serially extensible to the solution plan $t_0 * Puton(B,C) * Puton(B,Table) * Puton(C,Table) * Puton(B,C) * Puton(A,B) \prec t_\infty$ (which happens to be a non-minimal solution).[5]

We note that the protected plans made by a causal link planner such as SNLP are more constrained than the unprotected plans made by non-causal link planners such as TWEAK [2] and UA [13], or planners that use disjunctive protections (e.g. multi-contributor causal links) such as MP and MP-I [9]. Thus, there may be domains with subgoals that are not serializable with respect to SNLP but are serializable with respect to these latter planners. ART-MD-RD, first described in [9], and shown below, is one such domain:

ART-MD-RD domain from [9]			
Op	Prec	Add	Del
A_i (i even)	I_i, he	G_i, hf	$\{I_j \mid j < i\} \cup \{he\}$
A_i (i odd)	I_i, hf	G_i, he	$\{I_j \mid j < i\} \cup \{hf\}$

To see this, consider a problem where the initial state contains $I_1, I_2, \ldots I_n$ and he and we want to achieve two subgoals g_1 and g_2. If we consider the class of protected elastic plans, we are forced to decide which step gives the condition he to the step A_2 achieving g_2, and since some of the possibilities, such as the initial state, eventually become infeasible in the presence of the second subgoal g_1, the problem will not be trivially serializable for this class. This difficulty goes away when we restrict our attention to the class of unprotected elastic plans.

In [14], Veloso and Blythe also provide a range of domains where protection commitments become the critical issues with respect to serializability. In particular, they compare SNLP with a state space means ends analysis planner which doesn't use any protection and conclude that in these domains, the protections can hurt the performance of SNLP.

Although Veloso and Blythe's observations have been made in the context of a state-space vs. plan-space planner comparison, they can be generalized by noting that the critical issue once again is the commitment inherent in the plan for the first subgoal, rather than the planner being used to extend it. In particular, similar performance tradeoffs would be observed if the planners being compared were both state-space or both plan-space planners (e.g., one being TWEAK [2] and the other being SNLP), as long as one planner uses protections and the other doesn't.

6 Factors Influencing the Selection of a Refinement Planner

One of the prime motivations for understanding subgoal interactions in refinement planning is to see if such an understanding would help us in selecting the right type of refinement planner (including multi-method refinement planners such as UCP [7]) for a given domain. In this section, we will use our analysis of subgoal interactions to explore the various factors that affect this decision.

Based on our characterization of subgoal interactions, a general method for selecting a refinement planner involves (a) delineating the class of plans with respect to which most of the goals of the domain are trivially serializable, and (b) finding a refinement planner that is capable of generating exactly that class of plans. However, ''a'' and ''b'' are not completely independent. Since the serializability and parallelizability notions are defined in terms of a given class of plans (see Sections 2.4 and 4), and we have not put any restrictions on the legal classes of plans, it is theoretically possible to find a sufficiently small class of

[5] On the other hand, the one way rocket problem is not serializable even without the protection restriction; the critical issue for this problem is the prefix plan requirement. Since by Korf's original definition, both these problems are non-serializable, and thus indistinguishable, we note that by considering serializability in terms of classes of plans, we can make finer-grained distinctions among different problems.

plans with respect to which any given set of subgoals are trivially serializable. This is not useful in and of itself if there is no (domain-independent) refinement planner that can generate exactly that class of plans.

Consider, for example, the artificial domain shown below (originally described as the $D^*S^1C^2$ domain by Barrett and Weld [1]):

$D^*S^1C^1$ domain of Barrett and Weld			
Op	Prec	Add	Del
A_i^1	I_i	G_i	G^*
A_i^2	I_i	G_i	

All problems have the initial state where all I_i and G^* are true. Whenever the goals of the problem include both G_i ($i \leq n$) and G^*, the problems will not be trivially serializable for the class of elastic plans, since some of the elastic plans for the G_i subgoals will contain the steps A_i^1 which delete G^*. The problems will however be serializable for the subclass of elastic plans that do not contain any of the steps A_i^1 steps. Unfortunately, this latter fact is of little help in exploiting the subgoal interactions in the domain since elastic plans are generally generated by plan space refinements, and domain independent plan-space refinement will not be able to avoid generating the plans with A_i^1 steps. The domain is also trivially serializable with respect to the class of feasible suffix plans (see Section 2.4), which can be produced by BSS refinements. This leads to the following guideline:

Guideline 1 *Given a domain, we should try to select a subclass of plans with respect to which the subgoals are trivially serializable, such that there is a domain independent way of generating that subclass of plans.*

6.1 Least committed plans and Serializability

Often, we have more than one subclass of plans that satisfy the requirements of the guideline 1. One feature that affects the selection of subclasses in such cases is the commitment level inherent in the various subplan classes. The conventional wisdom is that least committed plans lead to more efficient planning, and we will evaluate this idea here.

Although given two plans with the same set of steps, the plan with lesser commitment is always more likely to be serially extensible with respect to another goal than a more committed one, this dominance does not directly translate to subgoal serializability, which is defined in terms of all plans of a specific class.[6] In particular, as we saw above, the domain $D^*S^1C^2$ is not trivially serializable for the class of elastic plans, but is trivially serializable for the class of feasible suffix plans (even though the latter are more constrained).

The intuition about least commitment being more conducive to efficient planning is true in general however when the domain contains a set of goals, not all of which are serializable with respect to any one subclass of plans. To see this, consider the domain below which is similar to the domain $D^*S^1C^2$ except for the augmented delete lists of the actions.

Variant of $D^*S^1C^1$ domain			
Op	Prec	Add	Del
A_i^1	I_i	G_i	G^*, I_{i-1}
A_i^2	I_i	G_i	I_{i-1}

These delete lists ensure that every solution for a problem involving the goals G_i and G_j ($i < j$) will have the action achieving G_i before the action achieving G_j. Now, in this situation, if the problem contains more than one of the G_i subgoals, then it will not be trivially serializable with respect to the class of suffix plans, whereas any problem that does not contain G^* will be trivially serializable for the class of elastic plans. If we have to pick a plan for the first subgoal without knowing what the next subgoal is going to be, we are still better off in general picking a less constrained partial plan. The empirical results of Barrett and Weld [1] in domains that are laboriously serializable for all their planners, do support this view to some extent.

Although less constrained plans are more likely to be serially extensible, more constrained plans do have their advantages. They are typically easier to ''handle'' in terms of consistency and terminations checks [10]. Given a domain containing subgoals that are trivially serializable for two classes of plans $\mathcal{P}_1$ and $\mathcal{P}_2$, it can be more efficient to do planning with the more constrained class of plans. The foregoing discussion can be summarized as follows:

Guideline 2 *If the set of subgoals in a domain are trivially serializable with respect to two different subclasses of plans, choose the subclass containing more constrained plans.*

Guideline 3 *If the set of subgoals in a domain are not trivially serializable with respect to any one subclass of plans, and no subclass is a clear winner in terms of percentage of problems that will be trivially serializable with respect to it, choose the subclass containing the less constrained plans.*

6.2 A Preliminary Evaluation

Although the guidelines 1-3 above are still very high level and need to be fleshed out further, they do extend our understanding about the selection of refinement planners. For example, they tell us that given a planning domain containing arbitrary subgoals, a reasonable way of improving average case performance would be to consider the most constrained class of subplans with respect to which the maximum number of subgoals are trivially serializable, and use a refinement planner, which only produces plans in this class. Given a domain, where the individual subgoals are all serializable by Korf's definition, but the subplans for these subgoals can contain many potential interactions, the set of goals will be trivially serializable for both the class of elastic plans and the class of blocked plans. However, the latter are more restrictive class of plans, and thus using a refinement strategy that produces them can lead to improved planning performance.

[6] If we are considering case-based planning, rather than generative planning, then generating and storing partial plans with fewer constraints is more likely to be a win, as they can be reused and extended in more novel situations; see [5].

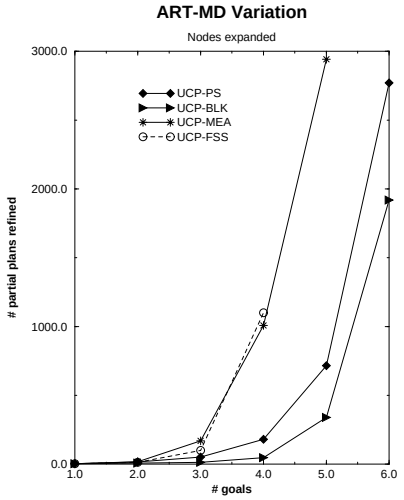

Figure 3: Results showing that the right class of subplans for a given domain may have intermediate level of commitment

We tested this hypothesis in a variation of Barrett and Weld's D^1S^2 domain shown below:

Variant of D^1S^2 domain			
Op	Prec	Add	Del
A_i (i odd)	I_i	M_i, he	hf
A_i (i even)	I_i	M_i, hf	he
B_i (i odd)	M_i, he	G_i	he
B_i (i even)	M_i, hf	G_i	hf

The domain contains a set of goals of the form g_i which can be achieved by actions B_i. Each B_i in turn needs the condition M_i given by action A_i. A_i also provides he or hf conditions to B_i, and B_i. Because he and hf conditions are deleted by many steps, the subplans for individual top-level goals will have many interactions, even though the overall plans are all serially extensible. We tested this domain on several instantiations of UCP [7] that generate different subclasses of plans, including prefix plans (UCP-FSS), protected elastic plans (UCP-PSS) and blocked plans (UCP-BLK). The results are shown in the plot in Figure 3. Our results show that blocking of steps of a top-level goal in a serializable domain improves performance both over plan-space (PS) refinements alone or over state-space refinements alone. The former is because of the plan handling cost while the latter is because the domain is not trivially serializable for prefix or suffix plans.

7 Summary

In this paper, we addressed the issue of subgoal interactions by concentrating on the patterns in which subplans for conjunctive goals may be combined to form complete plans in refinement planning. We started with a generalized representation and semantics for partial plans, and based the characterizations of subplan interactions in terms of two simple notions -- mergeability and serial extensibility of subplans for subgoals. The concepts of independence and serializability of subgoals are derived by generalizing mergeability and serial extensibility over classes of partial plans. We then showed that existing characterizations of serializability differ only in terms of the specific class of partial plans that they implicitly address. Finally, we used our analysis of subgoal interactions to generate some preliminary guidelines for selecting a refinement planner given a domain. In future, we hope to flesh out these guidelines further.

References

[1] A. Barrett and D. Weld. Partial Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, Vol. 67, No. 1, 1994.

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333--377, 1987.

[3] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Readings in Planning*. Morgan Kaufmann, 1990.

[4] M. Ginsberg. Approximate Planning. *Artificial Intelligence*, special issue on Planning and Scheduling. Vol. 76. 1995.

[5] L. Ihrig and S. Kambhampati. On the Relative Utility of Plan-space vs. State-space planning in a case-based framework ASU CSE TR 94-006; Dec 1994. (Submitted for publication)

[6] D. Joslin and J. Roach. A Theoretical Analysis of Conjunctive Goal Problems. Research Note, *Artificial Intelligence*, Vol. 41, 1989/90.

[7] S. Kambhampati and B. Srivastava. Universal Classical Planner: An algorithm for unifying state space and plan space approaches. In Current Trends in AI Planning: EWSP 95, IOS Press, 1995.

[8] S. Kambhampati. Refinement search as a unifying framework for analyzing planning algorithms. In *Proc. KR-94*, May 1994.

[9] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. *Artificial Intelligence*, Vol. 69, 1994. pp. 235-278.

[10] S. Kambhampati, C. Knoblock and Q. Yang. Planning as Refinement Search: A Unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* special issue on Planning and Scheduling. Vol. 76. 1995.

[11] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, Vol. 33, 1987.

[12] D. McAllester and D. Rosenblitt. Systematic Nonlinear Planning. In *Proc. 9th AAAI*, 1991.

[13] S. Minton, J. Bresina and M. Drummond. Total Order and Partial Order Planning: a comparative analysis. Journal of Artificial Intelligence Research 2 (1994) 227-262.

[14] M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. *Proceedings of AIPS-94*, 1994.

[15] Q. Yang, D. Nau and J. Hendler. Merging separately generated plans with restricted interactions. Computational Intelligence, 8(2):648-676, February 1992