A Domain-Independent Algorithm for Multi-Plan Adaptation and Merging in Least-Commitment Planners

Anthony G. Francis, Jr. and **Ashwin Ram***
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
{centaur, ashwin}@cc.gatech.edu
http://www.cc.gatech.edu/ai/ai.html

Abstract

Solving problems in many real-world domains requires integrating knowledge from several past experiences. This integration requires the dynamic retrieval of multiple experiences and the extraction of their relevant subparts. Our solution is the Multi-Plan Adaptor (MPA), a method for merging partial-order plans in the context of case-based least-commitment planning. MPA provides this ability by extracting an intermediate goal statement from a partial plan, clipping a stored plan to the intermediate goal statement, and then splicing the clipping into the original partial plan. MPA is implemented in the NICOLE multistrategy reasoning system, where it is paired with MOORE, an asynchronous, resource-bounded memory module. MOORE initially retrieves its current "best guess" but continues search, spontaneously returning a better retrieval as soon as it is found.

1. Introduction

Taking advantage of past experiences is the foundation of case-based reasoning. When confronted with a problem, a case-based reasoner recalls a past experience and adapts it to provide the solution to the new problem. Unfortunately, in many real-world domains we cannot count on a single past experience to provide the outline of a solution to our problems. For example:

- A graduate student asked to present his first paper at an overseas conference must draw on separate past experiences in preparing talks for conferences within his country and preparing his passport and flight arrangements for vacations outside of his country.
- A host planning his first large dinner party must recall both the outline of a menu as served at family gatherings and his separate experiences at preparing individual dishes for himself.
- A home hobbyist attempting his first large piece of furniture must recall both past examples of that type of furniture to provide a design and experiences with acquiring, assembling and finishing individual components.

All of these problems have something in common: every *piece* of the solution can be constructed entirely out of the agent's past experience (with suitable adaptation), but no *single* past experience suffices to solve the entire problem. For these types of problems, unless a case-based reasoning system has the ability to combine several past experiences, it will have to resort to expensive from-scratch reasoning in order to solve the problem.

Some CBR planning systems combine multiple cases during reasoning. However, they either gather all partial plans at retrieval prior to adaptation (e.g., PRODIGY/ANALOGY, Veloso 1994), or break plans into *snippets* at storage time so they can be retrieved individually (e.g., CELIA, Redmond 1990, 1992). Neither of these approaches is entirely satisfactory, for various reasons.

It is not entirely clear that all of the knowledge needed to solve a problem can be assembled at the beginning of problem solving. For example, in the furniture example, it is not entirely clear whether or not the agent needs to buy new sandpaper, and hence unclear whether the agent should recall past experiences of buying sandpaper at a hardware store. This uncertainty arises out of several concerns: the uncertainty of the world state (how much sandpaper does the agent have?), uncertainty in the effectiveness of agent actions (how much wood will a piece of sandpaper sand?) and the potential of exogenous events that can invalidate parts of the plan (if a friend drops and scars a piece of the furniture, will the agent have enough sandpaper to remove the scar, or will he need to buy more?). But it can also arise out of *the plan itself:* until the agent has decided on a design for the piece and how much wood will be involved, it is unclear precisely how much sandpaper is needed, and hence unclear whether or not a plan should be retrieved. If some amount of sandpaper is on hand, the goal of acquiring sandpaper may not even arise until late in the planning process, when it has become clear that the amount on hand is insufficient.

Precomputing case snippets also has drawbacks. While this allows us to extract subparts of a case to meet the needs of a component of a plan, these subcomponents need to be computed at storage time. Unfortunately, it is not clear that every useful breakdown of a case can be computed in advance. For example, in the foreign conference example, the two past experiences must be closely interleaved to produce a new plan. If the agent has not stored the passport experience as a separate snippet, the agent may not be

* This research was supported by the United States Air Force Laboratory Graduate Fellowship Program, by the Air Force Office of Scientific Research under Contract # F49620-94-1-0092, and by the Georgia Institute of Technology.

able to extract that particular piece of a case if it is retrieved — if the agent was able to retrieve it at all.

What is needed is a method that allows the merging of arbitrary numbers of plans at any point during the adaptation process, and which allows the dynamic extraction of relevant case subparts. Our solution is the Multi-Plan Adaptor (MPA), a novel method for merging partial-order plans in the context of case-based least-commitment planning. The MPA algorithm builds on the Systematic Plan Adaptor (SPA; Hanks & Weld 1994), a case-based least-commitment planner that annotates partially ordered plans with dependency structures to allow later refitting and adaptation. SPA shows significant improvements over generative planning, but can only adapt one plan at a time and hence for the types of problems described above must resort to significant amounts of from-scratch planning.

MPA overcomes this limitation of SPA by extracting an *intermediate goal statement* from a partial plan, *clipping* a stored plan to the intermediate goal statement, and then *splicing* the clipping into the original partial plan. Depending on the size of the plans spliced and the retrieval algorithms used, MPA can produce significant speedups over SPA.

Because the cost of retrieval can potentially outweigh the benefits of adaptation in an interleaved system, developing heuristics for deciding when to retrieve is a challenging problem. Our current implementation of MPA in the multistrategy reasoning system NICOLE uses an asynchronous, resource-bounded memory module called MOORE that initially retrieves its current "best guess" but continues to search, spontaneously returning a better retrieval as soon as it is found.

In this paper, we briefly review least-commitment planning and how it can be used for adaptation. We then present the MPA algorithm, describe its foundations in and extensions of the SPA algorithm, and then detail how MPA can be used to address the limitations of existing multi-plan systems. We then discuss how MPA can be integrated into various control regimes, including systematic, pure case-based, and interleaved regimes, and describe our implementation of interleaved MPA in the NICOLE system. We conclude the paper by reviewing other case-based planning work and then outline our contributions in the appendix.

2. Least Commitment Planning and Systematic Plan Adaptation

2.1. Least Commitment Planning: SNLP

Least-commitment planning departs from traditional planning systems by delaying decisions about step orderings and bindings as much as possible to prevent backtracking (Weld 1994). Depending on the domain and search strategy, least-commitment planning can lead to substantial improvements over traditional totally ordered plans (Barret & Weld 1994; but for an alternative view see Minton et al 1994, Veloso & Blythe 1994, Veloso & Stone 1995).

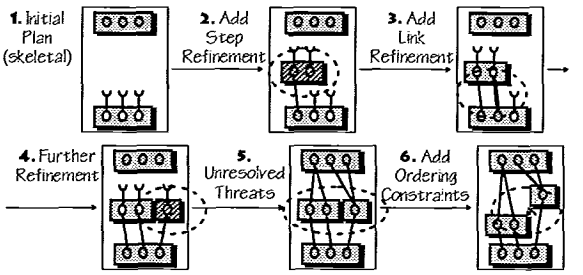

Figure 1. Refinement of Partial Plans

Least-commitment planners solve problems by successive *refinement* of a partial plan derived from the initial and goal conditions of the problem (Figure 1). Plans are represented as sets of steps, causal links between steps, variable bindings and ordering constraints. Beginning with a skeletal partial plan based on the initial and goal conditions of the problem, a least-commitment planner attempts to refine the plan by adding steps, links and constraints that eliminate open conditions or resolve threats.

An *open condition* in a partially ordered plan occurs when a plan step has a precondition that has no causal link to an effect in a prior step in the plan that establishes that condition. Open conditions can be resolved by adding a new step that establishes the desired effect and linking the precondition to it, or by linking the precondition directly to an effect already in the plan if one exists.

A *threat* in a partially ordered plan occurs when the condition established by the producing step in a causal link may be clobbered by the effects of another step before it is used by the consuming step in the link. Threats may be resolved by adding *ordering constraints* that move the threatening step before or after the steps in the causal link, or by adding *binding constraints* that ensure that the effects of the other step cannot unify with the step of the causal link.

Given an arbitrary sequence of decisions to add steps and bindings, there is no guarantee that the partial plan produced can be successfully refined into a correct solution. An *analytical failure* occurs when an incomplete partial plan cannot be further refined by adding new steps, links or constraints.

SNLP (McAllester & Rosenblitt 1991) is a complete, consistent and systematic partial order planner that uses a STRIPS-like notation to represent steps in its partial plans. Problem solving in SNLP begins with a list of initial conditions and goal conditions, which SNLP transforms into a skeletal plan with a dummy initial step whose postconditions establish the initial conditions and a dummy final step whose preconditions match the goal conditions. As outlined above, SNLP operates by repeated refinement in which open conditions are resolved and threats are eliminated.

2.2. Systematic Plan Adaptation: SPA

The Systematic Plan Adaptor algorithm (Hanks & Weld 1994) is an algorithm for case-based planning that incorporates SNLP into its adaptation mechanism. SPA

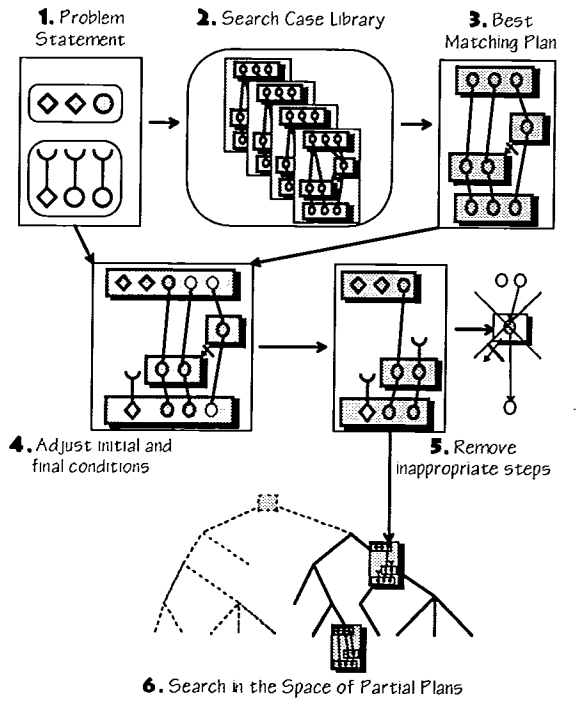

Figure 2. Fitting of Partial Plans

provides vast speedups over SNLP's performance by retrieving single plans which it fits and adapts, yet it maintains SNLP's properties of soundness, completeness, and systematicity.

SPA is based on three key ideas: annotate partial plans with *reasons* for decisions, add a *retraction mechanism* to remove decisions, and add a *fitting mechanism* to fit previous plans to current situations. Reason structures take a middle ground between a derivational trace of a planner's activity (e.g., Ihrig & Kambhampati 1994a, Ram & Cox 1994, Veloso 1994) and an unannotated plan. Like the partial-order plan representation itself, which records necessary orderings and interactions between steps without specifying a precise ordering, reasons record the necessary dependencies between refinements without specifying the precise ordering in which the planner made the choices.

The reason structures are first used to *fit* a retrieved plan to the new situation (Figure 2). The initial and goal conditions of a prior plan may not match the current situation, and it may contain steps and links that are not relevant to solving the new problem. To remedy this situation, SPA adjusts the initial and final conditions of the retrieved plan to match the current problem, and then uses the reasons to recursively eliminate steps that attempt to resolve deleted goal conditions and links that depend on deleted initial conditions.

Reasons are also used to select refinements for *retraction*. Once a plan has been retrieved, it may contain steps and constraints that would lead to analytical failures in the current situation. Reasons allow SPA to select past refinement decisions which are safe to retract (decisions upon which no other decisions depend) and to eliminate them.

2.3. Limits of SPA: Achieving Systematicity and Completeness

One limitation of SPA is that it is a single-plan adaptor; even if a new problem could be solved by merging several plans, SPA must choose only one and adapt it to fit. This limitation arises from SPA's attempt to maintain *systematicity* and *completeness*. A systematic planner never repeats its work by considering a partial plan more than once; a complete planner is guaranteed to find a solution if it exists.

SPA's generative predecessor SNLP achieves both of these properties through the clever design of its refinement algorithm. When SNLP removes a plan from the search frontier, it selects a refinement to apply and then replaces the plan with all possible instantiations of the refinement. Since this process consumes refinements and refinements are never retracted, duplicate partial plans are never generated; since all possible refinements are generated, the planner is guaranteed to not miss a solution.

In an adaptive planner like SPA, the picture is complicated by the presence of retraction operators. SPA ensures that the refinement and retraction operators never undo each other's work by marking each plan in the frontier with a direction. SPA begins adaptation by placing the retrieved plan onto its search frontier twice, marked once for refinement and once for retraction. The plan marked for refinement is treated precisely in the same way as in SNLP: all of its possible children are computed and placed on the frontier.

The copy of the plan marked for retraction, however, is treated differently. First, a refinement in the plan is chosen and retracted. This retracted plan is placed on the frontier and is marked for further retraction. Then, all of the *alternative* decisions that the planner might have made in place of the retracted refinement are generated, and these siblings are placed on the frontier marked for refinement. This retraction-refinement process achieves systematicity by ensuring that the only decisions being retracted are ones that were part of the original partial plan in the first place; it achieves completeness by ensuring that all of the alternative decisions are considered.

Unfortunately, the desire to maintain these properties effectively bars SPA from incorporating additional partial plans during problem solving. Because a plan contains a *set* of refinements, SPA cannot simply remove a plan from the frontier and splice a new retrieved plan into it without losing its completeness guarantee; nor can it leave both the original and adapted plans on the frontier in an attempt to maintain completeness without running the risk of re-generating the adapted plan, hence violating systematicity. To maintain both properties, all of the siblings of each refinement spliced into the plan must also be added to the frontier — a workable but potentially costly solution.

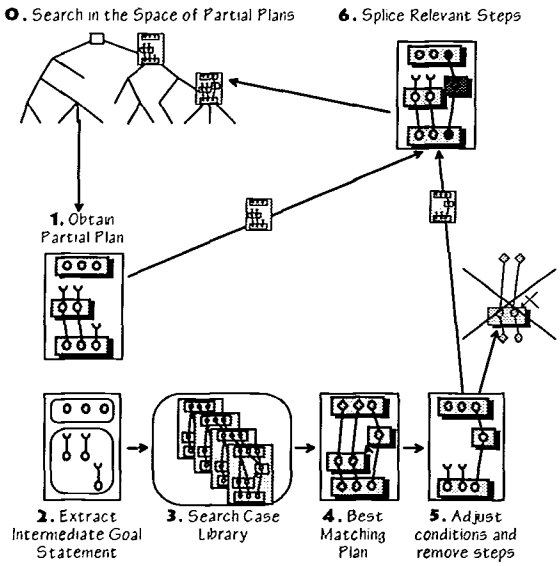

Figure 3. Overview of Multi-Plan Adaptation

3. Plan Merging

3.1. An Algorithm for Plan Merging: MPA

Because it adapts only one plan, SPA can resort to significant amounts of from-scratch planning even when the knowledge needed to complete the plan is present in the case library. To make the most effective use of the planner's past experience, we need the ability to recognize when a partial plan needs to be extended, select plans to that address the deficiency, and then extract and merge the relevant parts of the retrieved plan into the original plan.

The Multi-Plan Adaptor algorithm (MPA) resolves this problem in SPA by allowing the retrieval and merging of arbitrary numbers of cases at any point during the adaptation process. MPA further allows the dynamic extraction of relevant parts of past cases. To achieve this, the MPA algorithm extends the SPA framework in three crucial ways:

- extract *intermediate goal statements* from partial plans
- use the fitting mechanism to *clip* plans for merging
- build a *plan splicer* to merge two plans

Intermediate goal statements provide MPA with the ability to merge partial plans at any point of the adaptation process and contributes to its ability to dynamically extract the relevant subparts of retrieved cases. MPA translates the incomplete state of a partial plan into a goal statement, allowing the system to use the same retrieval mechanisms that it used to retrieve the initial plan. Once a plan has been retrieved that matches the intermediate goal statement, the relevant subparts must be extracted. The plan fitting mechanism performs this extraction dynamically, removing portions

Input: A partial plan *P*, and a case library *C*.
Output: A new partial plan *P'*.

procedure **MPA** (*P*, *C*):
1 *P'* ← **Copy-Plan**(*P*)
2. *igs* ← **GetIntermediateGoalStatement**(*P'*)
3. *plan* ← **RetrieveBestPlan** (*C*, *igs*)
4. {*clipping*, *mapping*} ← **FitPlan** (*plan*, *igs*)
5. for *cgp* in *mapping* do
6. if **Producer-Exists** (*oc-gl-pair*, *P'*)
7. then **Splice-Link** (*oc-gl-pair*, *P'*, *clipping*)
8. else **Splice-Step** (*oc-gl-pair*, *P'*, *clipping*)
9. **AddNewOpenCond-GoalPairs**(*mapping*, *P'*)
10. return *P'*

Figure 4. Outline of The MPA Algorithm

of the plan that are not causally relevant to the intermediate goal statement.

Intermediate goal statements are extracted by the inverse of the representational trick with which SNLP constructs its initial skeletal plans. Recall that SNLP builds the initial plan it considers by adding dummy initial and final steps whose post- and pre-conditions match the initial and goal conditions of the problem. As planning proceeds, open conditions in the goal statement are resolved, but new open conditions are posted as new steps are added. These open conditions themselves can be extracted from the plan to form a new goal statement. Similarly, the initial conditions of the partial plan can be extracted and used as a new set of initial conditions.[1]

Just like the original goal statement, the intermediate goal statement can be used to retrieve and fit a partial plan. However, the result of this process is not a complete fitted plan suitable for adaptation; it is a *plan clipping* that satisfies some or all of the open conditions of the partial plan from which the intermediate goal statement was derived.

To take advantage of the plan clipping for adaptation, it must be *spliced* into the original partial plan (Figure 3). Our splicing mechanism uses the intermediate goal statement to produce a mapping between the partial plan and the plan clipping, pairing open conditions from the partial plan with satisfied goal conditions from the plan clipping. The plan splicer uses this mapping to perform a guided refinement of the original partial plan, selecting goal conditions from the clipping and using the links and steps that satisfied them as templates to instantiate similar steps and links in the original plan.

[1] Unfortunately, since ordering constraints and binding constraints may be posted to the plan at any time, only the initial conditions can be guaranteed to be valid conditions for the intermediate goal statement. Conditions established by other steps of the plan may be clobbered by the addition of new steps and new ordering constraints. However, it might be possible to develop heuristics that select additional initial conditions that are likely to hold, perhaps in conjunction with more complex retrieval, fitting and splicing algorithms. In general, deciding which parts of a plan can be extracted to form a sensible and effective intermediate goal statement is a difficult and unsolved problem.

As these steps are added, new mappings are established between open conditions in the new steps and satisfied preconditions in the clipping and are added to the queue of mappings that the splicer is processing. Hence, the plan splicer performs a backwards breadth-first search through the causal structure of the plan clipping, using links and steps in the clipping to guide the instantiation of links and steps in the original plan. Figure 4 briefly outlines the MPA algorithm.

3.2. The Efficiency of Plan Splicing

Both adapting a single partial plan and adapting merged partial plans can produce significant benefits over generative problem solving. The cost of generative planning is exponential in the size of the final plan produced, whereas fitting a plan is a linear operation in the size of the plan. Hence, the potential exists for substantial improvement through retrieval and adaptation if an appropriate past plan exists, especially for large plans. In certain domains, SPA has demonstrated significant improvements over generative planning. However, if large gaps exist in the retrieved partial plan, SPA must resort to adaptation, which, like generative problem solving, has an exponential cost in the number of steps that need to be added.

While this amount of adaptation may be a significant improvement over complete from-scratch problem solving, the potential exists to reduce that even further by using MPA to clip and splice more partial plans. Fitting a clipping and splicing a clipping are linear operations in the size of the plan being spliced. Hence, the potential exists for substantial improvement through plan merging if an appropriate past plan exists, especially if the gaps in the existing plan are large. An initial implementation of MPA for a test domain indicated significant speedups (beginning at 30%) over SPA for even the smallest examples (solution size of the final plan = 5 steps).

However, both plan adaptation and plan merging require the retrieval of a past plan, and that retrieval cost may offset the benefits of adaptation or merging. For SPA, retrieval costs are incurred once, before adaptation begins, and hence it is simple to determine whether the cost of retrieval will be offset by the benefits of adaptation. For MPA, the picture is not so simple. MPA allows plan merging at any point during adaptation, and hence it is not clear from the MPA algorithm itself when retrieval will be performed, how many retrievals will be performed, and hence whether those costs will be offset by the improvements in problem solving. In order to determine this, we must embed MPA within a *control regime* that determines when and how often retrieval will be performed before we can precisely specify the benefits of multi-plan adaptation.

4. Controlling Plan Splicing

Merely having the ability to splice plans together does not allow us to take advantage of past experience. We need to decide what experiences to combine and when to combine them. Because the MPA algorithm can potentially be performed at any point during the adaptation process — using an initial skeletal plan derived from the initial and goal statement, using a fitted plan derived from retrieval, or using an adapted plan after some arbitrary amount of retraction and refinement — we have considerable flexibility in deciding what to retrieve, when to retrieve it and when to merge it.

We have considered three alternative control regimes, each of which makes different commitments about when to retrieve and when to adapt. On one end of the spectrum, *Systematic MPA* preserves SPA's property of systematicity by splicing all retrieved cases before adaptation begins. On the other, *Extreme MPA* never performs (generative) adaptation and instead uses a set of *pivotal cases* (Smyth & Keane 1995) to guarantee completeness.

Both Systematic and Extreme MPA make extreme commitments: either integrate all knowledge before adaptation begins, or never adapt and rely solely on past experience. An alternative approach is to allow plan splicing at any point during adaptation. In the middle stands *Interactive MPA* (MPA-I), a system that can potentially attempt a retrieval at any time, either with the initial skeletal plan or with partial plans produced as a result of adaptation. Since the results of splicing cause large jumps in the search space, the system deliberately departs from the systematicity of SNLP and SPA in an attempt to solve the problem with less search.

However, allowing arbitrary plan retrieval and plan splicing is not without cost. Performing a full search of the system's case library at every step of the problem space could be computationally prohibitive. The costs of searching the case library at every step of the problem space could outweigh the benefits of reduced search, especially if the system enters a "slump" — an adaptation episode which begins and ends with the application of relevant clippings, but which goes through a series of intermediate plans for which the system cannot match any existing plans in its case library. Clearly, it is worthwhile to retrieve and apply clippings at the beginning and end of a slump, but a full search of the case library at each intermediate step could cost more than the benefits that the initial and final retrievals provide. This is the *swamping utility problem* — the benefits of case retrieval can be outweighed by the costs of that retrieval, leading to an overall degradation in performance as a result of case learning (Francis & Ram 1995).

Developing heuristics for deciding when and when not to retrieve is a challenging open problem. To solve this problem, we have implemented a multistrategy reasoning system called NICOLE which pairs MPA with an *asynchronous,* resource-bounded memory module called MOORE. In NICOLE, MPA and MOORE share a central blackboard. When MPA requests a partial plan, MOORE returns its current best guess. However, the retrieval request remains active, and as the MPA adapts

the plan the new partial plans it constructs provide additional cues for MOORE to attempt further retrievals. When MOORE finds a new past case whose degree of match exceeds a certain threshold, it signals MPA, which splices it into the appropriate partial plan.

5. Related Work

There are wide bodies of work on both least-commitment planning and case-based reasoning. The most relevant example of that work to this research is of course SPA, upon which MPA builds. Other similar plan reuse systems include PRIAR (Kambhampati & Hendler 1992) an SPA-like system based on NONLIN, and XII (Golden et al 1994), an SPA-like system that plans with incomplete information. Hanks and Weld (1995) discuss these and other plan reuse systems from the perspective of the SPA framework.

MPA's plan splicing mechanism is in many ways similar to DERSNLP (Ihrig & Kambhampati 1994), a derivational analogy system built on top of SNLP that uses *eager replay* to guide a partial order planner. While DERSNLP's eager replay mechanism is in some ways similar to a limiting case of Systematic MPA in which a single plan is retrieved and spliced into a skeletal plan derived from an initial problem statement, DERSNLP goes beyond SPA's reason mechanism and includes a full derivational trace of problem solving in its cases. While DERSNLP and its extension DERSNLP-EBL focus on when it is profitable to retrieve a partial plan, unlike MPA-I they do not provide the capability of interrupting adaptation as a result of an asynchronous memory retrieval, nor do they provide the ability to integrate the results of multiple plans.

Combining multiple plans in case-based reasoning is not a new idea. The PRODIGY/ANALOGY system (Veloso 1994) can retrieve and merge the results of an arbitrary number of totally ordered plans during the derivational analogy process. However, because PRODIGY/ANALOGY manipulates and stores totally ordered plans, it runs into significant issues on deciding how to interleave steps (Veloso 1994, p124-127), an issue MPA avoids because of its least-commitment heritage. Furthermore, PRODIGY/ANALOGY deliberately limits its capability to retrieve and combine cases on the fly in an attempt to reduce retrieval costs.

The JULIA system (Hinrichs 1992) also has the ability to combine pieces of several past cases, but this is largely a domain-dependent algorithm for merging declarative structures, rather than a domain independent planning system. The CELIA system (Redmond 1990, 1992) stores cases as separate *snippets,* case subcomponents organized around a single goal or set of conjunctive goals. Snippets provide CELIA with the ability to retrieve and identify relevant subparts of a past case based on the system's current goals. Note that while snippets are superficially similar to plan clippings, plan clippings are constructed dynamically during problem solving, whereas snippets need to be computed and stored in advance.

Clippings are similar to macro operators (Fikes et al. 1972) in that they use past experience to combine several problem solving steps into a single structure that can be applied as a unit, allowing the system to make large jumps in the problem space and avoid unnecessary search. However, macro operators differ from clippings in two important ways. First, macro operators are traditionally precomputed at storage time, whereas clippings are computed dynamically; second, macro operators are fixed sequences of operators, whereas clippings are partially ordered sets of operators that may be resolved in a wide variety of ways in the final plan.

Kambhampati & Chen (1993) built and compared several systems that retrieve partially ordered case-like "macro operators". They demonstrated that least-commitment planners could take greater advantage of past experience than totally-ordered planners because of their ability to efficiently interleave new steps into these "macro-operators" during planning. While this work focuses primarily on interleaving new steps into single past plans, the explanations the authors advance for the efficiency gains they detected could be extended to suggest that least-commitment planners would be superior to totally-ordered planners when interleaving multiple plans. The completed MPA-I system should provide us a testbed with which we can empirically evaluate this hypothesis.

6. Conclusion

We have presented the Multi-Plan Adaptor, an algorithm that allows a case-based least-commitment planner to take advantage of the benefits of several past experiences. MPA provides the ability to retrieve and merge dynamically selected case components at any point during the adaptation process by extracting an intermediate goal statements from a partial plan, using the intermediate goal statement to retrieve and clip a past plan to the partial plan, and then splicing the clipping into the original partial plan.

Multi-plan adaptation has the potential for substantial speedup over single-plan adaptation, but in order for those benefits to be realized MPA must be embedded within a control regime that decides when the system attempts a retrieval, when the system merges, and when the system resorts to adaptation. We have used the NICOLE multistrategy reasoning system to implement an interactive algorithm called MPA-I which allows the retrieval of past cases at any point during adaptation.

To cope with the potentially swamping cost of retrieval at every adaptation step, MPA-I is currently paired with an asychronous, resource-bounded memory module called MOORE that retrieves a "best guess" and then continues to monitor the progress of adaptation, returning a new or better retrieval as soon as it is found.

Appendix: Describing the Contributions

A. Reasoning Framework:

1. What is the reasoning framework?

The reasoning framework is a case-based reasoning system layered on top of a least-commitment planner. This framework can be embedded in several control regimes; an interleaved regime based on an asynchronous memory module is implemented.

2. *What benefits can be gained that motivated using adaptation? reuse?*
   Adaptation/reuse of past problem solving cases can eliminate vast amounts of search and cause significant speedups in problem solving.
3. *What are the specific benefits and limitations of your approach?*
   Our approach can provide improvement over from-scratch problem solving and single-case adaptation. However, it is currently limited to merging problem-solving cases or other cases with a causal structure, and like all planning and learning algorithms it is sensitive to domain characteristics.
4. *What invariants does it guarantee?*
   When the domain affords the use of past problem solving experiences and relevant experiences are available, this method can reduce the number of states visited in the search space.
5. *What are the roles of adaptation, knowledge and reuse in your approach?*

A. Knowledge:

1. *What does it encode?* Plans
2. *How is it represented?* Partial-order plans with a STRIPS-like notation.
3. *How is it used?* To provide the framework for new plans.
4. *How is it acquired?* Through generative or case-based problem solving.

B. Adaptation:

1. *What is adapted?* Plans (past problem solving cases).
2. *Why is it adapted?* To avoid generative search.
3. *What properties of the knowledge representation does adaptation require or exploit?* Plans must be annotated with reason data structures that allow retraction of decisions (for adaptation) and plan fitting (for adaptation and merging).
4. *How are the benefits measured?* In the number of states in the search space avoided and/or problems solving time.
5. *What is gained?* Past plans provide an outline of a solution to the new problem.

C. Reuse:

1. *What is reused?* Plans (past problem-solving cases).
2. *Why is it reused?* To guide the problem solver and avoid generative search.
3. *What properties does the reuse process place on the adaptation process?* The adaptation process must be capable of being guided by clipped partial plans.
4. *How are the benefits measured?* In the number of states in the search space avoided and/or problems solving time.
5. *What is gained?* Past plans provide an outline of a solution to the new problem.

B. Task:

1. *What is the task? The domain?*
   The task is problem solving. The algorithm is domain-independent.
2. *What are the inputs?*
   Problems specified in terms of initial and final states.
3. *What are the outputs?*
   Solutions to the problems in the form of partial-order plan representations.
4. *What are the constraints on the outputs?*
   A correct solution must be found if one exists within the system's search-depth limits.
5. *Are there characteristics of the domain that the method requires, relies on, or exploits in some way?*
   The method does not rely on any domain characteristics.

C. Evaluation:

1. *What hypotheses were explored?*
   For a certain class of domains and problems, multi-plan adaptation will be more effective at reducing search than single-plan adaptation.
2. *What type of evaluation?*
   Test cases were used to evaluate the algorithms to verify that they provided improvements. For these problems, we empirically tested problem solving methods (generative, single-plan, and multi-plan) against problem-solving time. This work is preliminary; a more complete evaluation is in progress.
3. *What comparisons were made with other methods?*
   MPA was tested against a generative planner (SNLP) and a case-based reasoner (SPA) and showed improvements over both on the same problems.
4. *How does the evaluation validate or illuminate your theory of adaptation of knowledge for reuse?*
   Our research provides an initial indication that multi-plan adaptation can be more effective than planning from scratch.
5. *What are the primary contributions of your research?*
   Our research contributes an algorithm for multi-plan adaptation for case-based least-commitment planning and offers initial indications that multi-plan adaptation may be an efficient adaptation technique

References

Barret, A. & Weld, D. (1994). Partial Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, 67(1), 71-112.

Cox, M. (1993). Introspective Multistrategy Learning. *Cognitive Science Technical Report #2*, Atlanta: Georgia Institute of Technology, College of Computing.

Fikes, Hart and Nilsson. (1972). STRIPS. *Artificial Intelligence*, pp 189-208, 1972.

Francis, A. and Ram, A. (1995). A Comparative Utility Analysis of Case-Based Reasoning and Control-Rule Learning Systems. In *Proceedings, ECML-95*. Springer-Verlag.

Golden, K., Etzioni, O., & Weld, D. (1994). Omnipotence Without Omniscience: Sensor Management in Planning. In *Proceedings of AAAI-94*. AAAI Press/MIT Press.

Hanks, S., & Weld, D. (1992). Systematic Adaptation for Case-Based Planning. In *Proceedings of the First International Conference on AI Planning Systems*, 96-105. Morgan Kaufmann.

Hanks, S., & Weld, D. (1995). A Domain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research* 2, p319-360

Hinrichs, T.R. (1992). *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence Erlbaum.

Ihrig, L. & Kambhampati, S. (1994). Derivation Replay for Partial-Order Planning. In *Proceedings of AAAI-94*. AAAI Press/MIT Press.

Kambhampati, S. & Chen, J. (1993). Relative Utility of EBG based Plan Reuse in Partial Ordering vs. Total Ordering Planning. In *Proceedings of AAAI-93*. AAAI Press/MIT Press.

Kambhampati, S. & Hendler, J. (1992). A Validation Structure Based Theory of Plan Modification and Reuse. *Artificial Intelligence, 55*, 193-258.

Kolodner, J.L. (1993). *Case-based Reasoning*. Morgan Kaufmann, 1993.

McAllester, D., & Rosenblitt, D. (1991). Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634-639. AAAI Press/MIT Press.

Minton, S., Bresina, J., Drummond, M. (1994). Total Order and Partial-Order Planning: A Comparative Analysis. *Journal of Artificial Intelligence Research* 2, p319-360.

Redmond, M. (1992). Learning by Observing and Understanding Expert Problem Solving. *Georgia Institute of Technology, College of Computing Technical Report no. GIT-CC-92/43*. Atlanta, Georgia.

Redmond, M. (1990). Distributed Cases for Case-Based Reasoning: Facilitating Use of Multiple Cases. In *Proceedings of AAAI-90*. AAAI Press/MIT Press.

Smyth, B. & Keane, M. (1995). Remembering to Forget: A Competence-Preserving Deletion Policy for CBR Systems. In *Proceedings of IJCAI-95*.

Weld, D. (1994). An Introduction to Least-Commitment Planning. *AI Magazine, (15). 4*, pages 27-61.

Veloso, M. (1995). *Planning and Learning by Analogical Reasoning*. Springer-Verlag.

Veloso, M. & Blythe, J. (1994). Linkability: Examining Causal Link Commitments in Partial-Order Planning. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175.

Veloso, M. & Stone, P. (1995). FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research* 3, p25-52.