COLLAGE: A Diversified Constraint-Based Planning Architecture

Amy L. Lansky
Andrew G. Philpot

Sterling Software/NASA Ames Research Center
Artificial Intelligence Research Branch
MS 269-2, Moffett Field, CA 94035-1000
LANSKY@PTOLEMY.ARC.NASA.GOV PHILPOT@PTOLEMY.ARC.NASA.GOV

Abstract

This paper describes the architecture of COLLAGE, a planner that provides generality and flexibility in several dimensions of planning: it includes a range of plan-construction methods, utilizes a flexible localized search technique, and is currently being extended to allow plan construction to be performed at varying times relative to plan execution. Our paper begins with a characterization of planning in terms of a five-dimensional spectrum of planning features. We then describe how traditional planning and more recent planning methods fit within this characterization. The rest of the paper focuses on COLLAGE and how it can embody several points within the planning spectrum.

1 Introduction

For many years, the term "planning" has been equated with the particular representation and reasoning mechanisms used by "traditional" planning systems such as STRIPS [4], SIPE [15], and NONLIN [14]. Such systems define domain requirements in terms of state predicates (that describe possible world states) and action descriptions (that define actions in terms of their state-based preconditions and effects). Problem instances are described in terms of initial and goal states. The task of a planner is then to come up with a set of actions (either totally ordered or partially ordered, depending on the system) all of whose executions are valid with respect to action preconditions and effects and all of which transform an initial state into a desired goal state.

Motivated by tractability problems inherent in this formulation of planning [2] as well as by a desire to tackle domain features not adequately addressed by it, researchers have recently focused on a variety of new approaches to planning. Examples include work on dynamic and unpredictable domains, methods for dealing with concurrency and other aspects of multiagent activity, the reuse of previously generated plans, and attempts to close the gap between high-level reasoning and the low-level control required by physical agents.

In our work, we have found it useful to take a broadened view of planning that accomodates both traditional and recent approaches – i.e. planning is *any* activity that generates a plan that fulfills stated purposes and satisfies domain requirements. Indeed, all existing methods of planning can be characterized as instantiated points within a five-dimensional feature space that embodies the variety of ways domains can be encoded and plans can be generated:[1]

1. **Domain and Problem Representation**
   This dimension defines *how domain and problem information is represented.* Among the possibilities are: (1) traditional state-based description; (2) action-based description, where domain requirements are described in terms of "behavioral" constraints on actions rather than in terms of state; and (3) functional input/output requirements. This dimension must also address a variety of semantic issues. For example, what forms of parallelism are allowed? Can effects depend on whether or not actions occur in parallel? Does the representation make use of metric-time? Does it assume discrete or continuous actions and time? Is probabilistic information or uncertainty about domain state or actions allowed? Are mechanisms (such as localization) provided to cope with the frame problem or other problems related to scope of effect?

2. **Plan Representation**
   This dimension defines *how plans are represented.* Among the possibilities are partially or totally ordered sets of actions, reactive control rules, procedures, code, or neural nets. Other aspects of this dimension include the possible use of metric time-stamping information, interval-based temporal relations, the allowance for varying "levels" of activity, and the integration of justification structures and other kinds of auxiliary information within the plan.

3. **Plan Construction**
   This dimension defines *how plans are constructed.* Possibilities include use of the traditional algorithms based on the modal truth criterion, the utilization of user-defined procedures or previously generated plans that are combined in a variety of ways,

[1] Each of these dimensional descriptions should not presumed to be exhaustive in any way.

plan transformation or compilation, forward projection, action decomposition techniques, temporal and causal reasoning, and neural-net reinforcement methods.

4. **Control**
This dimension defines *how plan-construction is controlled.* Options include search-based, reactive, or blackboard-based control, or combinations of these. In some cases, there are multiple control-spaces. Some control schemes are flexible and context-sensitive, while others are rigid.

5. **Time of Plan Construction**
This dimension deals with *when plan construction methods are applied relative to plan execution* and is thus linked to plan-construction control. The spectrum of possibilities range from pure advance pre-planning to pure run-time reactive-planning. Intermediate points along this spectrum include reactive systems that allow for some advance reasoning and pre-planning systems that allow for some reactive plan modification.

Although the number of possible dimensional combinations is enormous, each one may represent a valid and useful approach to planning, depending on the target domain. For example, in unpredictable or quickly changing environments, it is important to allow for uncertainty in the domain and plan representation; plan construction is most profitably performed dynamically in response to the changing environment. In contrast, more stable domains that require intricate forms of coordination warrant more thorough advance reasoning.

So-called "traditional" planners can be characterized within this five-dimensional space as follows. First, they utilize state-based domain and problem description and plan construction techniques based on the modal truth criterion. Depending on the planner, plans may be partially or totally ordered sets of actions, and may also allow for action hierarchies. Control is search-based with a variety of possible search strategies. Planning is done in advance of execution and execution failures are handled with complete replanning or plan modification based on attached justification structures [7].

At the other end of the "temporal" dimension, "reactive" planning techniques apply pre-constructed plan fragments in response to dynamically changing world states and goals. However, they differ in how plan fragments are initially constructed and represented. In systems like PRS [5], procedures are supplied by the user. In others [1, 6, 13], extensive advance reasoning about the action/state space is performed, the fruits of which are distilled into reactive rules or code.

2 The COLLAGE Architecture

COLLAGE is a descendant of the GEMPLAN planner [8, 9].[2] The design of both COLLAGE and GEMPLAN was motivated by a particular class of domains: large domains with parallel activities that require complex forms of coordination. Typical examples include building-construction planning and other forms of logistical planning. The demanding requirements of this domain class has led to an architecture that can be tuned to exhibit several points within the five-dimensional planning spectrum. The planner provides generality in the *domain and problem representation*, *plan construction*, and *control* dimensions, and is currently being extended to provide flexibility in the *time* dimension as well.

2.1 Constraint-Based Planning

Central to COLLAGE is the view of planning as "constraint satisfaction." Here we utilize the term "constraint," not in the confined sense used within the CSP literature [12], but in a much broader sense. In COLLAGE, a constraint is *any property that the planner knows how to test and make true.* The system is associated with a broad and easily extendable repertoire of constraint forms and constraint satisfaction algorithms. Domain descriptions and problem instances are defined in terms of *action descriptions* and *constraints.* *Action descriptions* provide the possible types of actions that can be instantiated within a plan and simply provide an action name with a set of parameters. For instance, *pick(block)* would define an action type, an instance of which is *pick(a)*. *Constraints* are instances of the constraint forms provided within the COLLAGE constraint repertoire. Most of the constraint forms are action-based; that is, they pertain directly to relationships between actions. These include the following:

- Action decomposition. This constraint form defines a set of possible decompositions (partially ordered sets of subactions) for a specified action type.
- A variety of temporal and causal requirements between action types. For example, in the blocks world, the constraint `enable(pick(B),put(B,_))`[3] would require each *put* action that places a block on some surface to be causally enabled by a *pick* action that first picks the block up.
- Desired patterns of behavior expressed as regular expressions over action types. For example, the pattern `(pick => put)*=>` would require events of type *pick* and *put* to alternate in time.
- CSP constraints on plan variables. These are used to restrict the bindings of action parameter values.

More traditional "STRIPS" constraints may also be utilized. These constraints pertain to the attainment and maintenance of state-based goal conditions and preconditions. The associated algorithms are essentially those used by traditional planners. However, rather than utilizing STRIPS-based action descriptors, COLLAGE represents state-based requirements in terms of constraints and auxiliary *predicate definitions.* As described above, a COLLAGE action description simply provides an action-type name and its parameters. A predicate definition

[2] COordinated Localized aLgorithms for Action Generation and Execution. Throughout the rest of this paper, we shall be referring primarily to features of the COLLAGE system. However, some of the features described may exist in GEMPLAN but are not yet implemented in COLLAGE.

[3] Capitalized tokens (or "_") represent variables. Lowercase is used for constants.

defines how world states are affected by actions. For instance, in the blocks world, the following definition of *clear(B)* is used:

```
predicate_definition(clear(B),
  [adder(pick(A),on(A,B)), adder(put(B,_),true),
   deleter(put(_,B),true), deleter(pick(B),true)])
```

A predicate definition includes a list of adder and deleter descriptions. The first parameter of `adder` or `deleter` is an action type which adds or deletes the predicate, under the condition in the second parameter. Notice how conditional effects are quite easy to describe in this context. Preconditions and goals are then described using constraints. For example, in the blocks world, we might have:

```
constraint(precondition(pick(B),clear(B))
constraint(precondition(put(_,B),clear(B))
constraint(goal(on(a,b))
```

Notice how the separation of precondition constraints from predicate definitions clearly distinguishes between necessary action preconditions and those conditions utilized only for describing conditional effects.

Once specified, COLLAGE utilizes domain constraints to drive the planning process. Instead of backwards- or forwards-chaining on goals and conditions, COLLAGE planning is more properly viewed as search through a constraint satisfaction search space (see Figure 1). Each node in the space is associated with a plan constructed up to that point in the search. Each plan is represented as a partially ordered set of actions in which both potential as well as required forms of parallelism between actions are expressible. Upon reaching a node, the planner chooses a relevant constraint to test. If the constraint is not satisfied, the planner will apply an appropriate constraint satisfaction method (resulting in the possible addition of new actions, relations, and bindings), yielding a new plan at the next node in the search space. The branching factors in this space are the set of applicable constraints as well as the set of possible plan repairs to satisfy a constraint.

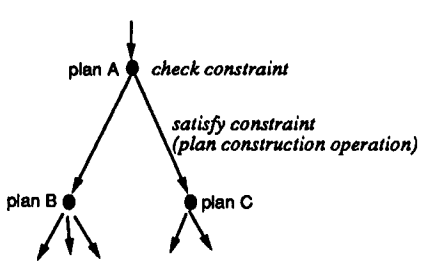

Figure 1: Constraint Satisfaction Search

Traditional planning may be seen as a specialization of COLLAGE's constraint-satisfaction view of planning. In traditional frameworks, the only kinds of "constraints" are goals and action preconditions. Likewise, the only methods of "constraint satisfaction" employed by traditional planners are those based on the modal truth criterion. Some planners have expanded upon this approach by including causal reasoning and action hierarchies [3, 15]. COLLAGE takes this one step further by allowing for *any* kind of constraint form and accompanying algorithms that are supplied to it. Each of the constraint satisfaction algorithms in COLLAGE is associated with its own structures and machinery that manipulate the underlying COLLAGE plan representation. This framework can be easily extended to include new constraint forms.

COLLAGE's diversified constraint-based view of planning allows the most natural and efficient forms of constraints to be used for expressing and fulfilling domain requirements. Though STRIPS-based domain representation is expressive, its associated algorithms can be prohibitive. It is often more natural and efficient to express requirements in other ways. Complex domains, in particular, are often restricted enough to enable more specialized and efficient methods of plan-construction. For example, action-based constraints can be quite efficient and are especially useful for expressing the coordination requirements of multiagent domains [8].[4]

The flexibility of the current COLLAGE constraint set also allows for various kinds of reasoning to be emulated. For example, we have recently extended the constraint mechanism to allow for conditionalization; i.e., constraint application can be conditionalized upon plan and domain features. In addition, decomposition constraints can be internally conditionalized, so that actions are expanded in different ways depending on the particular situation. Given these new capabilities (and the ability to appy constraints flexibly in time), decomposition constraints could be tuned to mimic reactive procedural reasoning (different "procedures" could be stored as alternative, conditionalized decompositions that are applied reactively).

2.2 Localized Reasoning and Agenda-Based Control

Using cheaper constraint forms is one way of mitigating planning cost. Another is provided by COLLAGE's use of *localized* reasoning, wherein COLLAGE searches a set of planning spaces (each devoted to a portion of the overall planning problem) rather than a global space. Intuitively, these planning spaces correspond to loosely interacting subproblems of an overall planning problem. The COLLAGE planning spaces are determined by structural information supplied within a domain and problem specification: action descriptions and constraints are partitioned into sets called *regions*. COLLAGE associates a planning space with each region that focuses on constructing a portion of the overall plan that utilizes regional action types and satisfies regional constraints (see Figure 2). Planning control for each region is governed by an agenda-based mechanism: constraints are "activated," placed on the agenda, and later handled by the regional search mechanism. COLLAGE allows for a variety of sources of constraint activation, including plan modifications made by the constraint algorithms, the search tree, the environment, and the user. A global

[4] The *enable* constraint described earlier between *pick* and *put* actions in the blocks world is a typical example of this. It would replace the use of the *holding* predicate in a more traditional formulation.

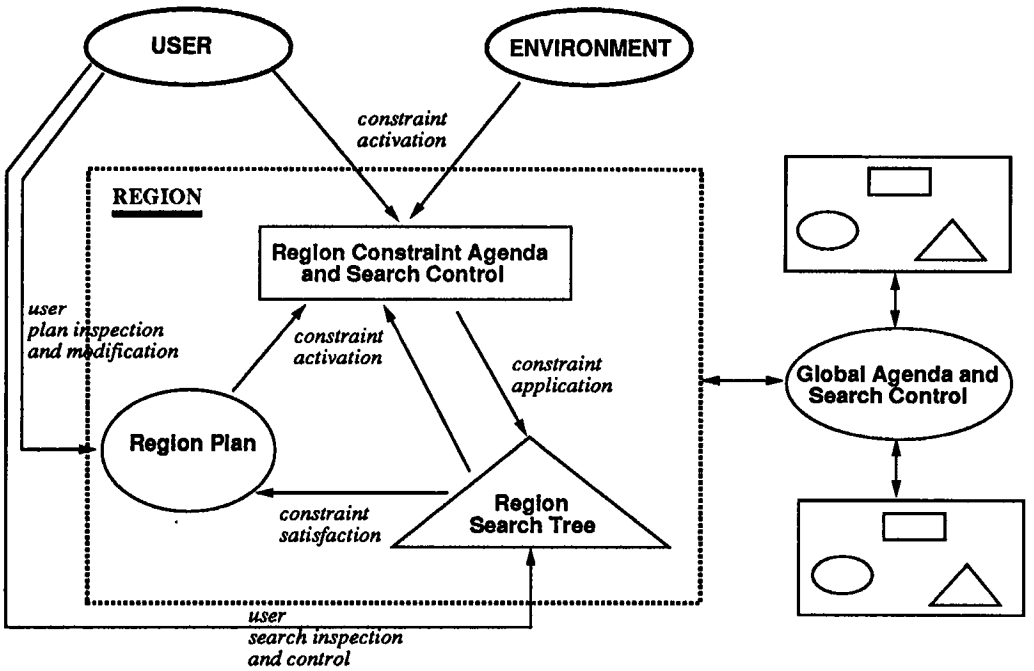

Figure 2: COLLAGE Localized Planning Architecture

agenda is used to keep track of which regions have been activated and require further planning.

COLLAGE regions may be structured arbitrarily; they may be disjoint, contain subregions, or *share* subregions (creating overlapping structures). The partitioning chosen for a domain is based on its natural characteristics – e.g., its physical structure, functional agents, temporal clusters, or levels of abstraction. However, the ultimate criterion for localization is the *scope* of domain constraints – i.e., *the constraints associated with a region are assumed to be relevant only to the actions within that region and its subregions.* As a consequence of localization, expensive constraints (e.g., STRIPS-based constraints) are applied only to the portion of the plan that includes actions relevant to those constraints.

Analytical and empirical results with localized search have demonstrated nearly universal search-cost reduction – and exponential savings in domains which require substantial backtracking and/or expensive constraint algorithms. However localized reasoning is also complex. In realistic domains, regions and their associated plan fragments will overlap (i.e., multiple regions may be involved in the construction of shared portions of the plan), resulting in the need for careful consistency maintenance among regional planning spaces. Previous papers have described the localized search technique, its benefits in reducing planning tractability problems, and its relationship to the technique of *abstraction* [9, 10]. We are currently working on automatic generation of domain localizations and experimentation with a variety of localization and search-control strategies.

2.3 Flexi-time Constraint Satisfaction

Because of the coordination-intensive nature of our target domain class, it is important to do most planning in advance of execution. For instance, the general contractor at a building site normally plans most of the construction process in advance – a large structure cannot be built "reactively," without any pre-planning coordination. However, complex, real-world domains also require run-time plan modification. This kind of reasoning can take at least two forms: (1) Some constraints cannot usefully be applied until run-time. Such constraints should be deferred until they are truly applicable or satisfiable. A typical example is a run-time dispatch constraint that controls access to resources. (2) Unanticipated situations resulting from run-time errors, user intervention, an incomplete domain theory, or environmental factors may trigger the need to make plan repairs.

In order to meet these requirements, we have designed the COLLAGE architecture to blend pre-planning search-based reasoning with more dynamic forms of reasoning.[5] We term this fusion of pre-planning with run-time reasoning *flexi-time constraint satisfaction.* The intuition is that a constraint should, in priniciple, be applicable at any time. During pre-planning, constraints are triggered primarily by plan modifications made by the planner. However, constraints could also be triggered and applied in response to the run-time environment or the user. For example, the environment may cause unexpected modifications to the plan, or the user may decide to alter the domain and problem specification.

Unfortunately, backtracking in a plan-space search framework (where the order in which actions are added into the plan has no relationship with the order in which they are executed) becomes problematic once

[5]Since our target domain class is not highly dynamic, we are not focusing on issues such as time pressures in run-time reasoning. Instead, our goal is to maintain plan correctness while providing flexibility in plan construction and repair.

plan-execution has begun. How can one backtrack over a node if portions of the plan associated with that node have already been executed? We believe that the best solution is to conduct run-time reasoning much the same way a human would. That is, once execution has begun, a plan is "patched" in response to the situation. Information gleaned from a record of the prior search space may be useful, but backtracking into the prior search space is not. A similar tactic is taken in recent work on plan reuse and modification [7].

Our initial approach to flexi-time reasoning in COLLAGE will be based on the creation of a "reconstituted" search framework each time plan modifications are made at run-time. Reconstitution results in a new search tree node that embodies the new state of the plan, domain specification, and environment. The node may encompass: (1) new user directives, in the form of modifications to the domain and problem specification, and (2) user-specified or other externally-motivated plan modifications. Once reconstituted, constraint-satisfaction search may proceed much as during pre-planning search. That is, constraints that have been triggered as a result of the new plan changes must be checked. However, one other class of constraints must also be tackled – those that may have become violated due to the plan changes, even though they were not explicited triggered by them. Towards this end, we will be extending the COLLAGE plan-storage structure to incorporate an explicit "justification" for each plan action, relation, and binding in order to serve as a framework for tracking and correcting constraint violations.

2.4 User Participation in Planning

In our experience with coordination-intensive domains, we have come to recognize the importance of *user-planner integration.* If users have deep knowledge of a domain and a vested interest in the form of the final plan, they will not willingly utilize a planner unless it allows for their direct input into the planning process. Unfortunately, the planning community has largely ignored this problem. Our attempt to deal with user-planner integration has resulted in the development of COLLIE, the **Collage Interface Environment**. Its design has been motivated by the needs of a NASA domain – the planning of data-selection and data-preparation steps performed by earth scientists [11]. A COLLIE user can visualize the growing plan, inspect features of each action, relation, and binding, and understand the relationship between plan structure and domain constraints. Tools are provided for viewing a graphical representation of the domain structure, visualizing the localized search process, and editing the domain specification. Tracing and stepping options are provided for monitoring planning and execution. Ultimately, the user will be allowed to modify the plan itself (possibly triggering constraints) and will be allowed to interact more directly with the constraint activation and search control mechanism.

2.5 Conclusions

This paper has described the architecture of COLLAGE, a planner that incorporates a repertoire of constraint forms for describing domain requirements, corresponding constraint-satisfaction algorithms for constructing plans, and a flexible localized, agenda-based search framework. Together, these capabilities enable flexibility in several planning dimensions: methods for domain and problem description, techniques of plan construction, and control over the plan construction process itself.

References

[1] Bresina, J. and Drummond, M. "Integrating Planning and Reaction," *Proceedings of the 1990 Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, Stanford University (1990).

[2] Chapman, D. "Planning for Conjunctive Goals," *Artificial Intelligence*, Volume 32, pp. 333-377 (1987).

[3] Currie, K. and A. Tate, "O-Plan: The Open Planning Architecture," *Artificial Intelligence*, Volume 52, pp. 49-86 (1991).

[4] Fikes, R.E, Hart, P.E., and Nilsson, N.J. "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Volume 3, Number 4, pp. 251-288, (1972).

[5] Georgeff, M.P. and A.L. Lansky, "Reactive Reasoning and Planning," *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington (July 1987).

[6] Kaelbling, L. "An Architecture for Intelligent Reactive Systems," in *Reasoning About Actions and Plans*, M.Georgeff and A.Lansky (editors), Morgan Kaufmann (1987).

[7] Kambhampati, S. and J. Hendler "A Validation-Structure-Baesd Theory of Plan Modification and Reuse," *Artificial Intelligence*, Volume 55, Numbers 2-3, pp. 193–259 (1992).

[8] Lansky, A.L. "Localized Event-Based Reasoning for Multiagent Domains," *Computational Intelligence Journal, Special Issue on Planning (4,4)* (1988).

[9] Lansky, A.L. "Localized Search for Multiagent Domains," *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, pp. 252–258 (1991).

[10] Lansky, A.L. "Localization vs. Abstraction: A Comparison of Two Search Reduction Techniques," *Proceedings of the AAAI-92 Workshop on Approximation and Abstraction of Computational Theories.*

[11] Lansky, A.L. and A.G. Philpot, "AI-Based Planning for Data Analysis Tasks," *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications (CAIA-93)*, Orlando, Florida (1993).

[12] Mackworth, A. K. "Consistency in Networks of Relations," *Artificial Intelligence*, Volume 8, pp. 99-118 (1977).

[13] Schoppers, M. "Universal Plans for Reactive Robots in Unpredictable Environments," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan, Italy, pp. 1039-1046 (1987).

[14] Tate, A. "Goal Structure, Holding Periods, and 'Clouds'," in *Reasoning About Actions and Plans*, M.Georgeff and A.Lansky (editors), Morgan Kaufman Publishers, pp. 267-277 (1987).

[15] Wilkins, D. *Practical Planning*, Morgan Kaufmann Publishers, San Mateo, California (1989).