Practical Planning in COLLAGE

Amy L. Lansky **Lise C. Getoor**

RECOM/NASA Ames Research Center
Artificial Intelligence Research Branch
MS 269-2, Moffett Field, CA 94035-1000
(415) 604-4431
LANSKY@PTOLEMY.ARC.NASA.GOV
GETOOR@PTOLEMY.ARC.NASA.GOV

Abstract

COLLAGE is a domain-independent planner that differs from traditional planners in two key ways: (1) it uses a diverse and extendible set of *action-based* plan constructions methods; and (2) it utilizes a technique called *localization* to partition the planning problem into smaller (and potentially interacting) subproblems. In this paper, we describe these features and how they are exploited in two different real-world planning domains. The first domain, building construction planning, has served as a framework for us to investigate how localization can improve scalability. The second domain, data analysis planning, benefits from COLLAGE's unique plan construction and representation methods. In addition, we describe extensions that were necessary to tackle these real-world problems.

1 Introduction

This paper describes COLLAGE, a nontraditional action-based planner [7, 8], and two real-world domains to which it has been applied: office building construction planning and Earth science data analysis planning. We discuss how each of these applications benefit from planning features that are unique to COLLAGE and also describe extensions that were required in order to handle them. In addition we present several empirical results, including a comparison of two planners on the same task. Finally, we identify some opportunities for learning in COLLAGE and discuss how domain knowledge acquisition can be facilitated.

COLLAGE differs from traditional planners in two key respects. First, its method of plan construction is *action-based* rather than state-based. This approach to planning is focussed strictly on actions and action interrelationships instead of satisfying state-based goals and preconditions. Moreover, rather than using a single plan construction technique (based on the modal truth criterion [1]), COLLAGE's plan construction techniques are diverse; each of COLLAGE's action-based constraint forms is associated with its own constraint satisfaction (i.e. plan construction) method. COLLAGE's second key feature is its use of *localized* planning, a method of partitioning a problem and the problem-solving process, in order to improve plan-construction efficiency.

Both of these unique features are exploited by the practical planning problems that we have been working on. For instance, in the office building construction domain, we have explored how localization and action-based planning affect problem scaling and efficiency. First we built an office building generator that can output problems of varying size and complexity. We then measured the performance of both localized and non-localized domain specifications on a suite of office building structures. Since SIPE [11] has also been run on one of the same test suites, we were able to compare its performance with that of COLLAGE.

We have also been exploring how localizations for the office building domain can be generated automatically. This is an interesting opportunity for learning in planning. In this paper we describe various criteria we have used for generating localizations and present some empirical results regarding their performance (see also [9]). We are hoping to extend these techniques so that we can find an optimal localization for a given problem class.

Our second domain application area, data analysis planning, benefits primarily from COLLAGE's expressive action-based domain representation and planning methods. This domain focuses on helping Earth scientists to plan data-retrieval, preparation, and analysis tasks; thus in this domain COLLAGE can be viewed as a data analysis assistant [6, 7]. Our work in this domain has raised some interesting issues about user/planner interaction, and has also forced us to rethink the boundary between planning and execution in our system. In addition, it has pushed us to extend our representation in several interesting ways. For example, we have extended our system to handle more complex action parameter types and new techniques for imposing and propagating parameter binding constraints. We also introduced ways of conditionalizing the plan-construction process and added a static knowledge base which can interact with the plan construction methods.

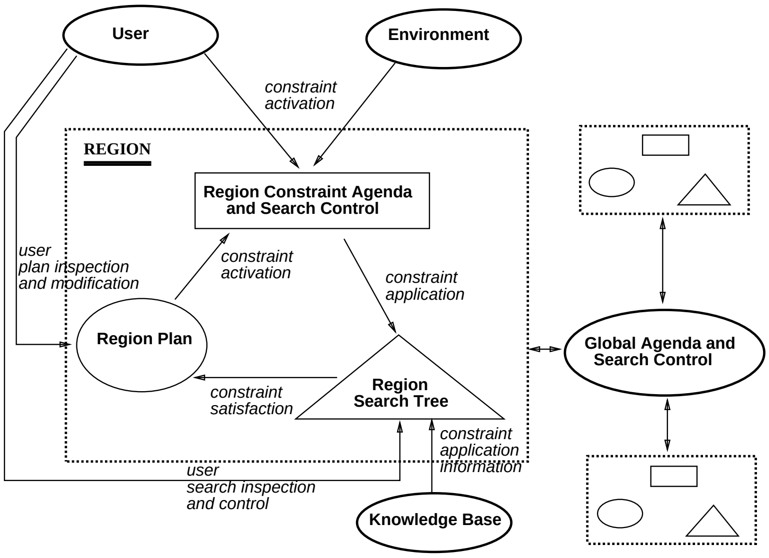

Figure 1: COLLAGE Localized Planning Architecture

2 The COLLAGE Planner

Central to the philosophy behind 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 [10], but in a much broader sense. In COLLAGE, a constraint is *any property that the planner knows how to test and make true.* COLLAGE is associated with a broad and extendible repertoire of constraint forms and constraint satisfaction algorithms.

After a target domain and problem goals and requirements have been identified, they are specified in terms of *action type descriptions* and *constraints*. Action type descriptions provide the possible kinds of actions that can be instantiated within a plan. Constraints are instances of the constraint forms provided within the COLLAGE constraint repertoire. These constraint forms include requirements for adding high level actions into the plan, for action decomposition (similar to hierarchical task networks [2, 11]), a variety of temporal and causal requirements between actions (these constraint forms may also add new actions into the plan), and binding requirements that may be imposed among action parameters.

Given a problem description, COLLAGE's task is to create a plan consisting of actions of the types provided that satisfies all problem constraints. Each COLLAGE plan consists of actions, relations between actions (forming a partial ordering), and binding requirements on action-parameters that have been imposed as a result of the planning process. Instead of backward- or forward-chaining on goals and conditions, COLLAGE planning is more properly viewed as search through a constraint satisfaction search space. Each node is associated with the plan constructed up to that point in the search and each arc with a plan construction step. Upon reaching a node, the planner chooses a relevant constraint to test. If the constraint is not satisfied, the planner will apply one of several constraint satisfaction methods (resulting in the addition of new actions, relations, and bindings anywhere they are required within the plan), yielding a new plan at the next node. The branching factors in this space are the set of applicable constraints and the set of possible constraint satisfaction methods.

Another important facet of COLLAGE is that it conducts planning in a distributed or *localized* fashion, searching a set of planning search spaces (each devoted to a portion of the overall planning problem) rather than a global space. The structure of a problem is defined by partitioning its action type descriptions and constraints into sets called *regions*. COLLAGE associates a planning space with each region that constructs a portion of the overall plan utilizing regional action types and satisfying regional constraints (see Figure 1). Planning control for each region is governed by an agenda-based mechanism. Constraints are "activated," placed on the agenda, and later handled by the search mechanism. Although COLLAGE constraints may currently only be activated by plan modifications made by the constraint algorithms, eventually we plan to extend the architecture so that constraints may be activated by the environment and the user.

The regions within a domain may be structured arbitrarily; they may be disjoint, contain subregions, or *share* subregions. The partitioning chosen for a domain is typically based on its physical structure, functional processes or 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 in the plan associated with that region and its subregions.

When regions are shared by more than one parent region, one of COLLAGE's tasks is to keep all regional planning spaces consistent. Thus, while COLLAGE allows for great flexibility in the formation of localization structures, this capability can also make localized reasoning complex. Indeed, we have found that realistic domains do tend to manifest complex regional interactions – i.e. it is fairly common for multiple regions to be involved in the construction of shared portions of the plan. In practice, however, we have found that the flexibility and savings afforded by COLLAGE's partitioning capability outweighs added consistency maintenance costs. Analytical and empirical results with localized search have demonstrated nearly universal search-cost reduction – and up to exponential savings in domains which require substantial backtracking. Previous papers have described the localized search technique, its benefits in reducing planning tractability problems, and its relationship to the technique of *abstraction* [8, 9].

3 Office Building Construction Domain

Our office building construction domain was derived from a civil engineering study using the SIPE planner [4]. For his thesis, Khartam applied SIPE to a suite of office building construction problems. We have built an office building generator, that, given a high level description of the building layout, generates the appropriate domain facts necessary to specify the building. Using this, along with a set of domain constraints that impose the same requirements used by SIPE, we have generated equivalent building plans. We have also used this domain as a testbed for our work on localization. In one case, we were interested in how localization improved the scalability of a problem. In the second study, we were interested in how localizations could be automatically generated from the domain constraints. These two studies are described below.

3.1 Scaling in the Office Building Construction Domain

In this section we discuss a suite of empirical tests conducted in the office building domain. Two different localizations of the domain were used: a non-localized partitioning, *global*, and a hand-crafted localization, *user-defined.*

The graph in Figure 2 provides, for both localizations, total cpu run-time for a suite of office building problems ranging in size from one to eleven floors, with an identical floor plan on each floor. The graph also provides timing results for SIPE, which was applied to exactly the same problem suite [4]. As can be seen, the localized version performs substantially better than *global,* which in turn performs better (in the long run) than SIPE.

Perhaps the most interesting result is the performance of COLLAGE relative to that of SIPE. Although these results are admittedly for different planners implemented on different hardware, it is clear that the slope of the COLLAGE curves increase much more slowly than that of the SIPE curve. Indeed, even the *global* curve starts to smooth out. In contrast, SIPE's performance was estimated to be $O(n^{2.4})$ in the size of the building [4]. This result is the best evidence we have to date of the relative efficiency of the action-based plan construction algorithms, even without the use of localization.

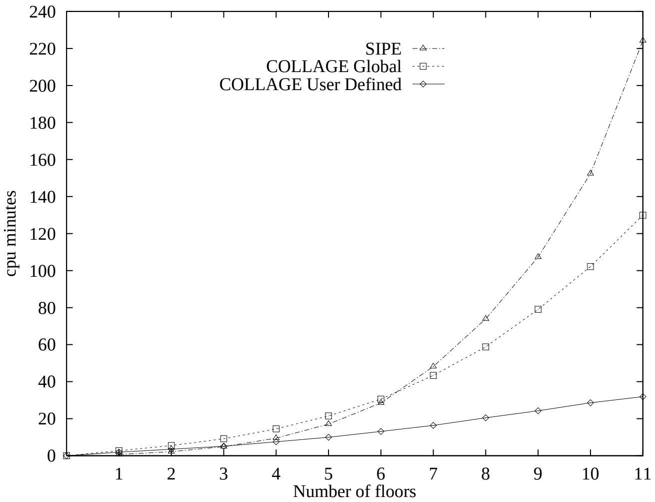

Figure 2: Total Run Times for Office Building Test Suite

3.2 Automatic Generation of Localizations

Until recently, COLLAGE users were required to describe how action type descriptions and constraints should be localized – i.e. localization was based on a user's intuition about problem structure. But coming up with an optimal localization is a tricky problem: there is an inherent tradeoff between increasing the degree of localization and the resulting overhead of maintaining consistency between regional search spaces. Thus, we have begun to explore techniques for generating localizations automatically. In particular, we developed the LOC localization generator, which creates localizations based on an initial, non-localized domain description. We have identified two domain independent criteria for forming these localizations: *abstraction* and *scope.*

Knoblock's method for generating planning *abstraction levels* for STRIPS-based frameworks is based on an analysis of problem goals and operator descriptors [3]. We were able to graft Knoblock's technique onto COLLAGE's action-based framework by recognizing that the relationship between state literals he derived was based on a type of *activation* relationship – i.e. how planning to achieve one literal could lead to (activate) planning to achieve another. Thus, to find an abstraction-based localization for COLLAGE, we focused on potential activation relationships between constraints.

First, from the syntactic description of each COLLAGE constraint, we derived the types of actions it could add into a plan. Given the activators for each constraint (another set of action types), we then derived a constraint activation graph, where an arc from constraint $C1$ to constraint $C2$ indicates that $C1$ might activate $C2$ during the planning process. We then found the strongly connected components of this graph. Each of these components was then used to form a planning region (an "abstraction level") consisting of the constraints for that component. Using region-agenda heuristics, these regions are searched in a total order consistent with the activation graph ordering. Like Knoblock's work, monotonic consideration of the regions is guaranteed – i.e. each region space will be searched only once.[1] In contrast, such monotonicity is not guaranteed for an arbitrary localization; i.e. in general, region spaces may be revisited many times depending on constraint activation behavior.

Another important criterion for localization is the *scope* or relevance of constraints to various portions of a plan. Determining the precise scope of an action-based constraint is simple: the action-types relevant to a constraint can be syntactically derived from its description. Given a problem's action types and constraints, its "most localized" or most finely-grained localization consists of a region for each action-type and a region for each constraint. Each constraint-region includes, as a subregion, each of the action-type-regions it refers to. This forms a two-tiered localization structure with action-type regions below a layer of constraint-regions. However, this structure also usually manifests a great deal of overlap or interaction between constraint-regions, since many constraints may refer to the same action-types.

Starting with this "most localized" partitioning, scope-based localizations may be created by merging and restructuring regions in a variety of ways. For example, if constraints $C1$ and $C2$ are relevant to exactly the same set of action types, they should be placed within the same region: i.e. the $C1$ region should be merged with the $C2$ region. Alternatively, if $C2$'s action types are a subset of $C1$'s action types, the $C2$ region should be made a subregion of the $C1$ region. This methodology is utilized by Loc, described below.

3.2.1 Loc

Loc is a general framework for generating localizations from a problem description. The input to Loc is the full set of problem action types and constraints. From this information, Loc first generates the "most localized" partition: for each action A there is a region R_A containing only the action type description for A, and for each constraint C, there is a region R_C which contains C and the subregions $\{R_{A_i} ... R_{A_j}\}$, where $\{A_i ... A_j\}$ is the set of action types relevant to C. Loc then applies a suite of transforms that iteratively modify this localization in a variety of ways. Currently, there are three basic Loc transforms:

- $Subsume(R_i, R_j)$: Parent region R_i "absorbs" or subsumes child region R_j.
- $Merge(\{R_1 ... R_n\})$: Replace $R_1 ... R_n$ with a region R that contains the union of all the constraints, action types, and subregions of $\{R_1 ... R_n\}$.
- $CreateHierarchy(R_i, R_j)$: Make R_j a subregion of R_i.

[1] This is true because the above construction assures that each region's constraints cannot activate any constraints associated with a previous region in the total order.

To create an abstraction-based localization, we use the method described earlier to find the set of activation-based clusterings of constraints. Loc then applies a *Merge* transform for each cluster, yielding a region containing the constraints for each abstraction level.

Generating interesting scope-based localizations was a bit more challenging. We identified several scope-based uses of the three basic transforms. For example: (1) Subsume regions which provide no additional localizing effect. For instance, if a region R contains only action types (and no constraints) and has only one parent region P, we can perform $Subsume(P, R)$ without any loss of locality; (2) Merge children or parent regions with identical scoping functionality; and (3) Create appropriate scoping hierarchies when possible. Notice that these transforms do not alter the portion of the plan each constraint is applied to; they simply remove unnecessary regions and regional interactions.

3.2.2 Empirical Results

Using Loc, we generated three localizations of the office building construction domain described earlier. The first, *scoped*, is the result of applying the scope-based transforms to the "most localized" localization. The second, *abstracted*, is the result of applying the abstraction-based merges to the constraint regions in the "most localized" localization. Finally, a third localization, *abstracted-scoped*, results from applying the scope-based transforms to *abstracted*. This localization combines the two criteria of scope and abstraction within a single localization. It has a similar structure *abstracted*, but removes some unnecessary regions and regional overlap.

Figure 3 shows total run-time for all three test scenarios. One obvious conclusion is that *scoped* performs best. We believe this can be attributed to several factors. First, the scope-based localization has much smaller regions. Since expensive operations such as temporal closure are only performed on a regional basis in Collage, much less work is done by *scoped*. Second, scope-based transforms remove unnecessary regions and overlap. As a result, scope-based localizations tend to have lower consistency maintenance costs. However, in this case, the savings of *abstracted-scoped* over *abstracted* is actually negligible. This is because both of the abstraction-based localizations suffer from unnecessary regional collapse. In particular, both of the abstraction-based localizations contain an extremely large region due to potential constraint-activation relationships. Thus, although the abstraction-based localizations do result in a monotonic reasoning space that searches each region only once [3, 9], they do so at a price. In contrast, *scoped* takes advantage of Collage's highly flexible localizing abilities – i.e. its ability to allow for a variety of localization structures and to flow back and forth among regional search spaces while still assuring consistency. It thus benefits from plan-space partitioning without paying the price of unnecessary regional collapse.

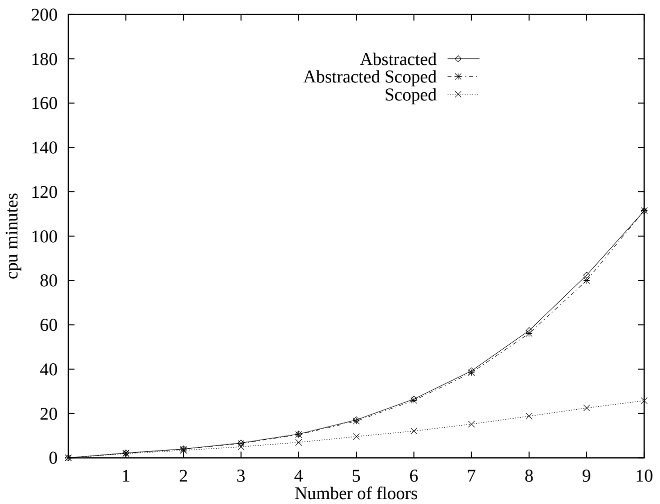

Figure 3: Total Run Times for Office Building Test Suite

Future work on Loc will focus on several areas. One will be feeding information about the actual planning process and the form of the final plan back into Loc, enabling it to improve a localization by analyzing planning behavior. For example, currently Loc applies scope-based transforms solely on the basis of the problem description – i.e. the *potential* scope of constraints with respect to the various action type descriptions. However, during planning, specific constraints may actually be applied to much narrower portions of the plan. For example, action types (and regions) might be profitably "split up" according to different parameter-value instantiations, especially if constraints tend to be reapplied in various different, narrow contexts (e.g., the same electrical constraint might be applied to distinct clusters of activity on different floors). We would like to extend Loc to include such splitting transforms, thereby allowing such constraints to be applied in narrower contexts.

We also hope to extend our experiments to enable testing on a broader set of problems and on problems with more complex constraints. In particular, we have already extended our office building domain to incorporate metric-time constraints. If resource constraints are also added, this test suite would become a scheduling problem, which would display much more backtracking behavior. This would enable us to test the effects of localization on even more challenging problems.

4 The Data Analysis Domain

An important new domain we are working on focuses on helping Earth scientists perform data analysis. Though we have worked on this domain for over a year, only recently have we come to view it as an instance of a much broader class of potential planning applications: helping humans to navigate through seas of software- and data-selection possibilities.

For example, one of the data preparation tasks we have tackled is *data registration*, a highly knowledge-intensive portion of the data analysis process. The development and validation of Earth-system models require several data inputs that may be gleaned from a mixture of remotely-sensed images (taken by satellite instruments) and ground sources (e.g., meteorological readings, soil maps, and vegetation surveys). After data sets are retrieved and before they can be used, they must all be *registered* so that they lie within the same coordinate projection system and scale – i.e., all coordinate values must accurately correspond to one another. Unfortunately, the scientist's task of selecting suitable data and acceptably registering them is more difficult than it might seem. This process is often a burdensome and tedious portion of the scientific cycle that can consume over half of a scientist's time.

One reason is that heterogeneous data types are often not directly comparable. For example, sparse temperature data collected by weather stations is usually not directly correlatable to satellite image data. Thus, all data sets for a particular application must be registered to some common *base map*. First, a target coordinate system and scale is chosen. This target system is typically one that is similar to a majority of the data sets to be registered and that meets scientific and data-related constraints. Next, a base map of the study area is chosen that conforms to the target system. Then, all data sets are registered to this map.

Depending upon the selected base map and the original form of a data set, required preprocessing steps may include geometric corrections, projection and scale transforms, radiometric corrections, atmospheric corrections, data restoration, interpolation, image enhancement, and ground control point selection (points that are used to achieve a correspondence between a data set and base map). Each step is typically composed of several substeps, and for each of these there may be a variety of algorithms, programs, and computational platforms to choose from. All of these choices must meet a variety of constraints that encode dependencies among the registration steps. If poor choices are made, the registration process may introduce unacceptable distortions into the data. In some cases, registration may be impossible.

We have found that Collage's action-based constraints are a natural fit to the requirements of the data analysis domain. In particular, data analysis requirements are most naturally expressed in terms of action decomposition, temporal/causal relationships between actions, and variable binding requirements. Our work in this domain also motivated us to extend COLLAGE in several ways. Viewing Collage as a data analysis "assistant" has led us to pay more careful attention to the facilities available for user interaction. In particular, we have found that these interactions are necessary at both planning time and at run-time. We have also recognized that some limited execution must be performed at planning-time and some planning must be conducted at run-time. We have thus designed extensions to our architecture to allow for *flexi-time* planning – i.e. activation of constraints at run-time as well as during preplanning.

This domain has also required us to extend our representation language. We have added a static knowledge base that can control portions of the planning process. We have also added several new action parameter types and a variety of flavors of binding constraints for them. These extensions are described in the following sections.

4.1 Planning, Execution, and User Interaction

As we began to write the domain constraints for this application and deepen our understanding of the role of our planner vis-a-vis the user, we saw the line blurring between traditional notions of planning and execution. Much of the data analysis process must be planned in advance; for example, scientists would be loathe to perform tedious manually-intensive transforms unless they have created a data analysis plan that they are fairly sure will succeed. However, some forms of execution must take place during the planning process. For example, sometimes, derived information about data sets must be acquired during the planning process in order to enable reasoning about which algorithms are most appropriate to use. Another example is the need for planning-time interactions with the user in order to facilitate the best algorithm choices.

Even allowing for some forms of planning-time "execution," however, some parts of the plan must be filled out or modified during actual data processing. For example, the ground-control-point selection process is often iterative – points must sometimes be added or deleted in order to yield the best registered image. These changes can't be determined until execution time, when an actual transform matrix is built and tried. Similarly, the most appropriate image enhancements for a data set often can't be determined until execution time, when the scientist can dynamically visualize those enhancements.

Thus, it is clear that the spectrum of planning behavior required by this domain does not match classical search-based preplanning nor common notions of "reactive" planning. Instead, the desired planning behavior can be viewed as a *dialogue* between the planner and the user, who are involved in a collaborative effort. The process must be able to flow between classical deliberative reasoning, more dynamic forms of user-interaction and control over the planning process, and dynamic plan modification in response to the execution environment or user-directives.

For this reason, we have designed COLLAGE to enable a more fluid form of reasoning that we call "flexi-time" planning. The system already allows for some forms of actions (e.g., choices, information retrievals, interactions with the user) to be performed at planning time. We plan to extend the primarily preplanning framework of COLLAGE so that constraints can be triggered at any time relative to execution. The COLLAGE constraint-triggering mechanism was intentionally designed to enable this kind of extension.

4.2 Domain Knowledge Base

From the start, it was clear that the data analysis planning process required extremely large amounts of domain specific knowledge. For example, it requires information about Earth projection systems, constraints on usage of specific data types, formats, projections, and scales, and information about available data transform algorithms. We also recognized that much of this information is relevant and understandable to the scientist and could be considered distinct from the planning engine and domain constraint specification. Thus, we decided to build a separate knowledge base consisting of static domain- and problem-specific factual information and to keep it separate from the domain constraints that drive the planning process. The knowledge base also includes domain-specific functions and problem-specific data selection and registration goals. The planner uses this knowledge by conditioning the constraint-satisfaction process on knowledge-base contents and by using the domain-specific facts and functions within it to impose constraints on plan variables. For example, consider the temporal constraint below.

```
(tempbefore
 :condition ((test (requires-registration ?prob)))
 :actions ((choose-coordsys ?coordsys ?prob)
           (do-registration ?data ?coordsys ?prob))
 :binding-req (suitable-target-system
                              ?data ?coordsys))
```

This constraint requires that each `do-registration` action be preceded by some `choose-coordsys` action that chooses a common target system. This constraint is additionally conditionalized so that it will only be applied if the data sets associated with `?prob` really do need to be transformed (this would be unnecessary if the data sets are already within the same coordinate system). An additional binding constraint is imposed on the action parameter variables to ensure that the target coordinate system chosen is suitable for the data being registered. Both `requires-registration` and `suitable-target-system` are defined by information (functions or facts) in the knowledge base.

Keeping the knowledge base distinct from the domain constraint specification and planning engine has several features that enhance utility. First, planning functionality can be increased by extending the knowledge base rather than by extending the underlying domain constraint specification. This enables the same domain specification to be used in numerous contexts with different knowledge bases. Second, the knowledge base can be represented in a form amenable to viewing and extension by the user. This feature is critical since we, the developers of COLLAGE, cannot possibly gather all domain-relevant information for this application. New data bases and algorithms are always being developed within the scientific community. To be truly useful, the system must be easily extendible. Thus, we hope to facilitate domain knowledge capture by making incremental knowledge addition performable by the users.

4.3 Extended Variable Facility

Another COLLAGE extension for the data analysis domain was the ability to use structured variables (i.e. variables composed of heterogeneous subcomponents). Structured variables are critical for this domain since each data set is typically associated with a variety of pieces of information about its features.

We also extended COLLAGE so that a variety of new binding requirements could be imposed on variables and their subparts. COLLAGE's variable-binding-requirement propagation facility is now quite extensive. Each COLLAGE binding requirement must be a unary or binary relation between action parameter variables. Whenever a new binding requirement is added into a region plan, COLLAGE maintains node and arc consistency using a simple forward propagation algorithm. The binding requirements within a plan form a network, much like a CSP-network [10]. However, new variables may be added into the network as actions are added into the plan, and rather than using a single network for the entire plan, a set of regional sub-nets are used, one for each region. Currently, the following kinds of relations are handled:

- Arbitrary boolean relations between variables from enumerable domains.
- Linear relations of the form ($Y\ op\ m*X+b$), where op is one of $\{<,>,\leq,\geq,=\}$, m and b are constants, and X and Y numeric variables.[2]
- Equality relations between variables from non-numeric, non-enumerable types.

COLLAGE also supports an n-ary binding requirement of the following form: `(require (f ?x1...?xn))`. A variable `?xi` that takes part in a `require` requirement can be of any type; `f` must be a boolean relation or function. If such a requirement is imposed, COLLAGE will keep track of the binding status of `?x1...?xn`. When the last of `?x1...?xn` is bound to a unique value, `f` will be tested as part of the binding propagation process.

Conclusion

Plans can be generated using many different methods; ideally the method utilized should depend on the nature of the target domain. In our work, realistic domains have benefited from the non-traditional features of COLLAGE. We have found that COLLAGE's action-based planning methods are particularly well suited to domains such as building construction and data analysis, which require coordination of many activities or tasks. Likewise, localization is a tool that can help with the problem of scaling the planning process for large domains. In this paper, we presented empirical results showing the benefit of both an action-based representation and localized reasoning for large problems. In addition, we described some of extensions that were required for these domains and identified key areas for learning and knowledge acquisition in COLLAGE.

[2]Using this facility, COLLAGE has been used for metric temporal reasoning. In particular, action types can be declared as "metric" and associated with a timestamp parameter. If two metric actions are temporally related as a part of the planning process, COLLAGE will automatically impose an appropriate linear relation between their timestamp parameters.

References

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

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

[3] Knoblock, C.A. "Learning Abstraction Hierarchies for Problem Solving," in *Seventh International Workshop on Machine Learning*, pp. 923-928 (1990).

[4] Khartam, N.A. *Investigating the Utility of Artificial Intelligence Techniques for Automatic Generation of Construction Project Plans*, Doctoral Dissertation, Stanford University Department of Civil Engineering (1989).

[5] Lansky, A.L. "Action-Based Planning," *Proceedings of the Second International Conference on AI Planning Systems*, Chicago, Illinois (1994).

[6] 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). Also in IEEE Expert, Volume 9, Number 1, pp. 21-27 (1994).

[7] Lansky, A.L. "A Data Analysis Assistant," *Proceedings of the 1994 AAAI Spring Symposium on Software Agents*, Stanford, California (1993).

[8] Lansky, A. L. "Localized Planning with Diverse Plan Construction Methods," NASA Technical Note FIA-TR-94-05, NASA Ames Research Center (1994). Submitted to *Artificial Intelligence Journal, Special Issue on Planning.*

[9] Lansky, A.L. and L. C. Getoor, "Scope and Abstraction: Two Criteria for Localized Planning," *Proceedings of the Workshop on Theory Reformulation and Abstraction* (1994).

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

[11] Wilkins, D.E. *Practical Planning*, Morgan Kaufmann Publishers (1988).