Scope and Abstraction: Two Criteria for Localized Planning

Amy L. Lansky
Lise C. Getoor

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

Abstract

Localization is a general-purpose representational technique for partitioning a problem into subproblems. A localized problem-solver searches several smaller search spaces, one for each subproblem. Unlike most methods of partitioning, however, localization allows for subproblems that overlap – i.e. multiple search spaces may be involved in constructing shared pieces of the overall plan. In this paper we focus on two criteria for forming localizations: *scope* and *abstraction*. We describe a method for automatically generating such localizations and provide empirical results that contrast their use in an office-building construction domain.

1 Introduction

Over the years, many researchers have focused on the use of *abstraction* to reduce search costs for planning and other types of problem-solving [1, 3, 4, 6, 13, 14]. Abstraction techniques restructure a problem and the problem-solving process into a set of "abstraction levels." At the top level of abstraction, a problem is described at its most coarse-grained level of detail. Each successive level is made more concrete by incrementally adding information to the problem description. The use of abstraction can improve problem-solving performance if the solution found for an abstract level serves as a good starting point for problem-solving at the next level of detail. Thus, abstraction may be viewed as a heuristic for ordering which pieces of a problem are solved first, and which later.

Our work also focuses on partitioning a problem to improve problem-solving (in our case, planning) performance. Our technique, *localization*, restructures a planning problem by partitioning its components into *regions*. Semantically, each region defines a "region of interaction" consisting of a subset of the overall problem requirements and action-type descriptions that are related in some way. A problem's regional structure then forms a basis for partitioning the overall planning space into a set of smaller spaces, one for each region. Each space is focused on constructing a plan fragment that contains region actions and satisfies region requirements.

For example, consider the localization depicted in Figure 1 for a building construction domain. This partitioning is based on mixed criteria: level of detail, contractor agents, and building-structure. Each box represents a region and contains the names of action types within that region's frame of reference. Some boxes also contain other boxes – those of their subregions. Each region would be associated with the requirements pertaining to its action types or the action types of its subregions. For example, problem solving for `painter` would focus strictly on painting activities and requirements. In contrast, region `roomA` would focus on the decomposition of `high-level` activities into lower-level `electrician` and `painter` activities for `roomA`.

There are at least two ways in which localization differs from traditional uses of abstraction for planning. First, localization more strictly partitions the frame of reference for problem solving; region requirements are satisfied only with respect to a region plan fragment, not the entire global plan. In contrast, most abstraction-based frameworks conduct reasoning globally at the lowest level of detail.

Second, localized search can accommodate complex region structures. Instead of being confined to partitionings composed of "levels," the regions comprising a localization may take on nearly any configuration – disjoint, hierarchical, overlapping. Indeed, since regions may overlap, localized problem-solving typically flows back and forth among the regional search spaces (usually, to cope with region interactions) rather than searching a sequence of problem levels. The localized search algorithm maintains consistency between these search spaces and the plan fragments that they generate, and also guarantees that all regional requirements are ultimately satisfied [9, 10].

In contrast, a traditional abstraction-based planner can revisit a planning level only via backtracking. Because of this, abstraction researchers have focused on devising partitioning techniques that guarantee minimal interaction between levels (thus minimizing backtracking). Unfortunately, however, real-world problems do not often lend themselves to neat partitioning; as a result, levels tend to collapse.

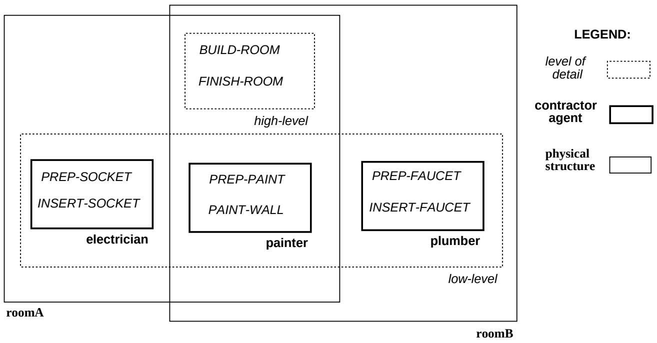


Figure 1: A Construction Problem Localization

As our sample domain illustrates, many criteria can be used to form a localization. Indeed, one criteria *is* abstraction; a region could be formed to correspond to each abstraction level. Localized search would then proceed from one region-level to the next, as in traditional abstraction-based planning. One focus of this paper is to show how Knoblock's method for generating abstraction-based partitionings [3] can be used to automatically generate abstraction-based localizations, even for non-STRIPS-based planning frameworks. As in Knoblock's work, our method yields a partitioning that guarantees monotonicity in the planning process: if a solution exists for a particular region (level), it can be found by refining a solution found at the preceding region (level), without disturbing established requirements. A region partitioning based on this form of abstraction will thus yield a search process that visits each region space only once (except for backtracking).

Another interesting criterion for localization is *scope* – i.e. the relevance of problem requirements to specific portions of the plan. Scope-based partitionings typically correspond to the natural structural characteristics of a problem. For example, when planning how to construct a building, some problem requirements will be relevant only to plumbing activities and others only to electrical activities. Alternatively, requirements might be clustered on the basis of building structure – e.g., rooms and floors (again, see Figure 1).

Localizations based on scope usually do not guarantee monotonic consideration of region requirements. Region search spaces may be visited more than once, even without backtracking. For example, in the scenario depicted in Figure 1, we might begin by constructing a high-level plan in region `high-level`. These high-level activities might then be further refined on a room-by-room basis. The planner might start by planning electrical and painting activities for `roomA`, then plan out plumbing activities for `roomB`, and finally return to painting activities for `roomB` (thereby causing a return to the `painter` search space). Although they may not guarantee monotonicity, scope-based localizations can provide powerful search reduction benefits, often surpassing those obtained using abstraction.

The primary goal of this paper is to demonstrate how localization can improve problem solving and, in particular, how scope and abstraction serve as criteria for localization. We begin in Sections 2 and 3 by describing the localized planning architecture of the COLLAGE planner [7, 9] and the kinds of search improvements that localization can provide. Next, Sections 4 and 5 discuss how scope and abstraction can be used as criteria for generating localizations. In particular, we describe how the LOC localization generator was used to generate scope and abstraction-based localizations for COLLAGE. Finally, in Section 6, we provide empirical results that contrast the utility of a variety of localizations for an office-building planning domain. We also contrast our results with those obtained using SIPE-2 on the same domain [5, 14].

2 Localized Planning

COLLAGE differs from traditional planners in two key ways: its use of localization to partition the planning search space and its use of non-traditional plan construction methods. In this section we provide a short description of both. For a more thorough description see [9].

Intrinsic to COLLAGE is the notion of planning as constraint satisfaction. Each COLLAGE problem description includes a set of action types and a set of constraints. The planner's task is to create a plan consisting of actions of the types provided that satisfies all problem constraints. Each plan (or plan fragment) 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.[1]

In contrast to STRIPS-based action descriptors [2], a COLLAGE action-type description simply provides an action name and the types of its parameters. For instance,

```
:action-type
  (insert-socket ?s_socket ?w_wall ?l_loc)
```

defines an `insert-socket` action type. A specific instance of this type might be `(insert-socket socket1 wall2 loc3)`.

Rather than being defined in terms of state-based goals and preconditions, COLLAGE problem requirements are defined in terms of *action-based constraints*. Such constraints focus on desired forms of action instantiation, action relationships, and action-parameter binding requirements. For example, consider the constraint:

[1] A CSP-network [11] on action-parameter-variables is embedded in the plan and maintained as part of the planning process.

```
(before
  (paint-wall ?p_paint ?w_wall)
  (insert-socket ?s_socket ?w_wall ?l_loc))
```

This requires that each `insert-socket` action be temporally preceded by a `paint-wall` action at the same wall; i.e. a wall must be painted before sockets are finally inserted. In a STRIPS-based framework, this requirement would be stated in terms of a state-based precondition for `insert-socket` actions. COLLAGE includes an extensive library of action-based constraint forms that can be utilized. The advantages of action-based representation and its relationship to state-based representation are discussed at length elsewhere [7, 9].

Each constraint C is associated with three types of mechanisms that it inherits from its constraint form: a *check method*, a set of *fix methods*, and an *activator* (along with an *initial activation setting*). The role of a check for C is to test whether or not C is satisfied by a plan. Whenever a check is applied, it returns a list of *bugs* – descriptions of the ways in which C is violated. The fix methods for C implement the various possible plan "repairs" that will satisfy these bugs. For example, the check for our sample constraint would return all `insert-socket` actions that have no appropriate preceding `paint-wall` action. For each such action, the fixes would either find an existing `paint-wall` action for the same wall and make sure it precedes the `insert-socket` action, or alternatively, would create a new such action.

The purpose of C's activator (and initial activation setting) is to indicate when C may be violated. Some constraints (i.e., COLLAGE's analogue to "goals") are initially active when planning begins. Others can be violated only by the addition of actions into the plan. Thus, C's activator is a set of action types. If any instance of one of these types is added into the plan, C will be "triggered" or activated for consideration by the planner. For instance, the constraint above is activated by the addition of `insert-socket` actions.

In addition to describing action types and constraints, a COLLAGE problem description also specifies how these types and constraints should be partitioned into regions. This partitioning may take on nearly any desired structure; the only restriction is that region-subregion relationships form a DAG. Once specified, COLLAGE utilizes the regions and their constraints to drive planning search. Instead of backward- or forward-chaining on goals and preconditions, as a traditional planner would, COLLAGE searches a localized constraint satisfaction search space (see Figure 2). The planning space for each region R is focused on constructing a region plan fragment, $Plan_R$, that contains actions of the types in R or any of R's descendant regions. At the end of planning, $Plan_R$ must satisfy all of R's constraints.

Search control for each region is governed by an agenda-based mechanism: constraints may be activated by the addition of actions into the plan, placed on a *constraint agenda*, and later handled by the region search mechanism. A *region agenda* is used to regulate control flow between regions. It keeps track of which regions have been activated for further planning (i.e. which regions have active constraints). Agendas may also be associated with heuristics that determine the order in which activated constraints and regions are chosen.

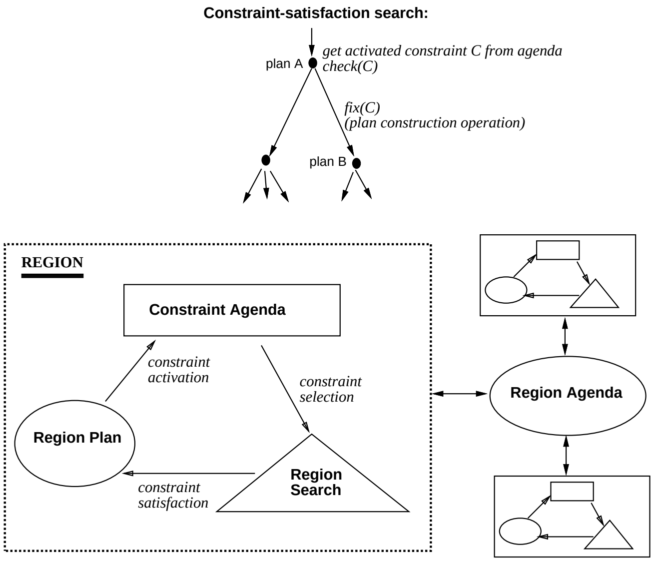


Figure 2: Localized Search Architecture

Since regions may share subregions (and thus, region plans may share subregion plans), plan changes made by a fix in one region may activate constraints in other regions. As a result, planning will typically flow back and forth between regions, necessitating careful consistency maintenance among the region planning spaces. COLLAGE's consistency maintenance algorithms are described in [9]. Analytical and empirical results for localized search [9, 10] have demonstrated that significant search-cost reduction can be obtained, and up to exponential savings in domains that require substantial backtracking. However, because of the need for consistency maintenance, localized reasoning can be complex. There is a tradeoff between focusing the application of constraints as narrowly as possible and the cost of keeping all of the region spaces consistent. A good localization balances both factors.

3 Improving Search Costs

Nearly all AI problem-solving systems conduct search with respect to some object of interest – be it a plan, schedule, or model. Typically, each search node is associated with a current value for the object and each arc is associated with an operation that transforms the object in some way. For example, in a planning search space, each search space node is associated with a plan and each arc is associated with a plan construction operation that adds new actions, relations, or variable bindings into the plan. Given this kind of framework, planning search costs can be improved in at least three ways:

1. *Reducing the cost of each arc operation.* One way to do this is to tune algorithms to problem requirements. Another way is to reduce the size of the plan being considered at each node.

2. *Using search heuristics.* Good search heuristics can result in less backtracking and may also improve solution quality.

3. *Reducing the size of the implicit search space*, typically by removing redundant or irrelevant nodes and arcs. One way of doing this is to eliminate from consideration those operations that are irrelevant to the plan at a particular node, or those that would otherwise be considered elsewhere in the search space, to the same effect.

Localization may be viewed as a technique for tackling aspects of all three kinds of improvements. By definition, localized search applies each planning operation (constraint fix) only to a region plan – the portion of the overall plan that the constraint is "relevant to." Since the cost of plan-construction operations is often related to plan size (e.g., the cost of temporal closure and CSP binding-propagation), applying operations to smaller plan fragments can reduce planning cost. In our experience with COLLAGE, this type of savings alone has yielded significant gains.

One of the major contributions of abstraction is that it is a good search heuristic – indeed, one that guarantees monotonic consideration of domain requirements. Although localization does not, in general, provide such guarantees, it too serves as an excellent search heuristic. One reason is that domain constraints are tackled on a region by region basis: by default, COLLAGE addresses all activated constraints within a region before moving to another region's search space. By design, the constraints within the same region tend to be the ones "most related" to one another (they constrain each other most tightly). Thus, localization can lead to a constraint ordering that tends to resolve conflicts as rapidly as possible. Since localized search proceeds according to how constraints activate one another during planning, it also serves to focus planning on the most relevant constraints and plan fragments at each point in the search process.

Localization can also be viewed as a way of reducing the size of the otherwise global space, since only regional constraints are considered at each search node. Constraints associated with other regions are "less relevant," so removing them from consideration eliminates redundancy within the search space [12]; other regions' constraints can be tackled later with (usually) no substantial effect on the form of the final plan.

4 Localization Criteria

Until recently, COLLAGE users were required to describe how action types and constraints should be partitioned – i.e. localization was based on a user's intuition about problem structure. We are now exploring how localizations can be generated automatically using LOC, a localization generator developed for COLLAGE. For the remainder of this paper, we will focus on two criteria we have examined: abstraction and scope.

4.1 Abstraction-Based Localization

Knoblock's method for generating planning abstraction levels for STRIPS-based frameworks is based on an analysis of problem goals and operator descriptors [3]. A partially ordered graph of state literals is derived, wherein an arc from literal $L1$ to literal $L2$ indicates that $L1$ must be at the same or higher level than $L2$. The strongly connected components of this graph form the abstraction levels. A total ordering of these components determines the order in which these levels are planned, each focused on achieving the literals comprising its level. The significance of Knoblock's technique lies in the fact that plan operators introduced to achieve literals at each level are guaranteed not to conflict with operators introduced at higher (preceding) levels.

How can this method be adapted to COLLAGE's action-based planning framework (or to other types of problem solving)? Looking at the graph construction process in more detail, one notices that it is based on an *activation* relationship between literals: i.e. how achieving a literal $L1$ can activate or lead to achieving $L2$. The result is a partitioning in which literals on a particular level cannot lead to ("activate") planning for literals on preceding levels.

Knoblock's method can be adapted to COLLAGE by making the analogies in the table below. In Knoblock's framework, each plan-construction operation is focused on achieving a particular literal; in COLLAGE, each operation is focused on satisfying a constraint. Just as achieving literal $L1$ may lead to achieving literal $L2$, satisfying constraint $C1$ may lead to satisfying $C2$.

	STRIPS	*COLLAGE*
Planning step	achieve literal	satisfy constraint
How a planning step leads to other planning steps	by introducing more literals to achieve	via constraint activation
Abstraction unit	level = set of literals	region = set of action types and constraints

Thus, to find an abstraction-based localization for COLLAGE, one must focus on the activation relationships between constraints. From the syntactic description of each COLLAGE constraint, we derive the types of actions it can add into a plan. Given the activators for each constraint (another set of action types), we then derive a constraint activation graph, where an arc from constraint $C1$ to constraint $C2$ indicates that $C1$ might activate $C2$. We then find the strongly connected components of this graph. Each of these components is used to form a planning region consisting of the constraints for that component. Using region-agenda heuristics, these regions are searched in an order consistent with the activation graph.

4.2 Scope-Based Localization

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. A problem's "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 usually manifests a great deal of overlap, since many constraints may refer to the same action-types.

Starting with this "most localized" partitioning, scope-based localizations can 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, 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. Notice that these transformations do not enlarge $C1$'s and $C2$'s scope of application with respect to the emerging plan – they merely optimize a localization by reducing the number of unnecessary regions and the need for consistency maintenance.

As another example, consider the following, more heuristic, scenario. Suppose that $C1$ is relevant to action types $\{A1, A2, A3, A4, A5\}$ and $C2$ is relevant to $\{A2, A3, A4, A5, A6\}$. The scope of these two constraints are similar, but not identical. However, it might be worthwhile to merge the two constraint regions together; the scope of application of both $C1$ and $C2$ would be slightly increased, but the amount of regional interaction would be reduced. Alternatively, suppose that $C2$ is relevant to $\{A3, A4, A5, A6\}$. In this case, we might decide to make the $C2$ region a subregion of the $C1$ region. This would reduce region interaction and would increase the scope of $C1$ (to include $A6$), but would not increase the scope of $C2$.

Often, scope-based and abstraction-based localizations are similar. After all, if two constraints are relevant to the same action types, they usually bear some activation relationship to one another. However, these two criteria are *not* identical. For example, suppose that $C1$ is relevant to $\{A1, A2, A3\}$, has activator set $\{A1, A2\}$, and can add actions of types $\{A1, A3\}$. Similarly, suppose that $C2$ is relevant to $\{A2, A3, A4\}$, has activator set $\{A3, A4\}$, and can add actions of types $\{A2, A4\}$. Clearly, the scope of these two constraints is different. Yet since they can activate one another (via $A2$ and $A3$), they would be placed in the same region in an abstraction-based localization.

How can scope be used in a STRIPS-based framework? Using our analogy (constraint≈literal, action≈operator), we see that the scope of a literal is the set of operators relevant to it. This set can be derived from the problem description: for each predicate P, find those operators that "add," "delete" or have a literal of type P as a precondition. A STRIPS-based planner localized according to scope would test the modal truth criterion for P only with respect to the plan operators that are "relevant" to P. COLLAGE's predecessor system, GEMPLAN [8, 10] localizes state-based reasoning in just this way.

5 LOC Localization Generator

LOC's input is a set of problem action types and constraints. From this information, LOC can generate a variety of localizations. Each localization is composed of a set of regions, $\{R_1 ... R_n\}$, where each region R_i consists of a set of constraints, a set of action types, and a set of subregions: $R_i = (\{C_{i_1} ... C_{i_m}\}, \{A_{i_1} ... A_{i_p}\}, \{R_{i_1} ... R_{i_r}\})$.

The first localization generated by LOC is the the most localized one: for each action A there is a region $R_A = (\{A\})$, and for each constraint C, there is a region $R_C = (\{C\}, \{R_{A_i} ... R_{A_j}\})$, 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. R_j is removed and R_i contains the union of the constraints, action types, and subregions of both regions.
- $Merge(\{R_1 ... R_n\})$: This transform replaces $R_1 ... R_n$ with a region R that contains the union of all the constraints, action types, and subregions of $\{R_1 ... R_n\}$.
- $CreateHierarchy(R_i, R_j)$: This transform makes R_j a subregion of R_i.

To create an abstraction-based localization, we use the method described in Section 4.1 to find the set of activation-based clusterings of constraints. Starting with the "most-localized" localization, LOC applies a *Merge* transform to each cluster of corresponding constraint regions, yielding a set of regions corresponding to the set of abstraction levels.

Generating interesting scope-based localizations is a bit more challenging. We have identified eight distinct scope-based uses of the three basic transforms, four of which are described below:

- *Remove regions which serve no purpose:* If a child region R_c has only one parent R_p and either: R_c has no constraints; or R_p has only one child (R_c) and no action types of its own; then $Subsume(R_p, R_c)$.

- *Merge children regions with identical scoping functionality:* If $\{R_1...R_n\}$ have exactly exactly the same parent regions and have no constraints of their own, then $Merge(\{R_1...R_n\})$.
- *Merge parent regions with identical scoping functionality:* If $\{R_1...Rn\}$ have exactly the same subregions and no action types of their own, then $Merge(\{R_1...R_n\})$.
- *Scoping hierarchy:* If the subregions of R_c form a subset of the subregions of R_p and R_c has no action types of its own, $CreateHierarchy(R_p, R_c)$.

The transforms above do not alter the scope of constraint application; they merely optimize a partitioning by removing unnecessary regions. The other four transforms are heuristic versions of the tranforms above. They have more relaxed application requirements and tend to remove excessive overlap at the expense of increasing constraint scope. We are currently developing evaluation criteria for controlling their use.

6 Experimental Results

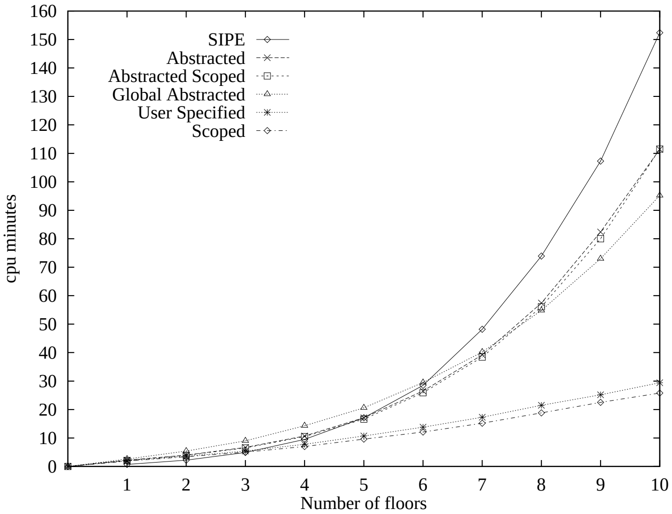
In order to compare the effects of our localization criteria on planning performance, we conducted a suite of experiments in an office-building construction domain that was also the subject of a study using SIPE-2 [5]. We developed three localizations that make use of the abstraction-based and non-heuristic scope-based transforms described in Section 5. The first, *scoped*, is the result of applying the scope-based transforms to the "most localized" localization, L. The second, *abstracted*, is the result of applying the abstraction-based merges to L. In this domain, its structure remains identical to L, except for one very large region, resulting from the need to merge several constraints that potentially activate one another. 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 contains the same large region as *abstracted*, but removes some unnecessary regions and regional overlap.

Each of these localizations was applied to the same suite of problems, for buildings ranging in size from one to ten floors with identical floor plans on each floor. We also conducted tests with a hand-crafted localization, *user-specified*, and with a global localization, *global-abstracted*. The latter consists of a single region containing all action types and constraints, and utilizes a constraint-ordering heuristic to simulate the effect of imposing abstraction levels on the search process. That is, the constraints are applied in an order that conforms with the constraint clumps used in *abstracted*; however, they are applied within a global search context rather than a localized search context.

The graph below shows total run-time for all five test scenarios as well as for SIPE-2 on the same domain. One obvious conclusion is that *scoped* performs best. The most important factor is that this localization manifests less unnecessary regional collapse than *abstracted* and *abstracted-scoped*. In particular, due to potential constraint activation relationships, the abstraction-based localizations contain a very large region containing all actions involved in constructing the office-building frame. Since temporal closure is one of the expensive operations in this domain, closure over large region plans can be quite costly. An important reason why *scoped* (and *user-specified*) performs so well is that it generates fewer closure relations (though the skeletal temporal structure of the plans generated by all of the localizations was the same). Since localized planning generates only locally-derivable closure relations, localizations with smaller regions have cheaper closure costs.

Although *scoped* performs best, notice that *global-abstracted* outperforms both of the abstraction-based localizations (as well as SIPE-2) in the long run. Although *global-abstracted* performs completely global planning operations (whereas *abstracted* and *abstracted* are at least somewhat localized), in the end the latter two are unduly weighed down by the need to perform consistency maintenance. Thus, they poorly balance the increased-partitioning vs. consistency-maintenance tradeoff.

It is also interesting to note that our hand-crafted localization for this domain, *user-specified*, is nearly identical to *scoped*. Indeed, *scoped* surpasses the performance of *user-specified*. These facts support the viability of Loc as an automatic localization generator. Also of import is the performance of Collage relative to that of SIPE-2. Although these results are admittedly for entirely different planners implemented on different hardware, it is clear that the slope of the Collage curves (even *global-abstracted*) increase much more slowly than that of the SIPE-2 curve. This result is the best proof we have to date of the relative efficiency and scalability of action-based (in contrast to STRIPS-based) planning, even without the use of localization.



7 Conclusions

This paper described localization and localized search: techniques for partitioning a planning problem into subproblems and searching a set of smaller, though possibly interacting, subproblem spaces. Since localized search can cope with complex, overlapping localization structures, it provides a powerful mechanism for coping with real-world planning problems. And since it is a generally applicable technique, we believe localization has great potential for other types of problem-solving as well.

A key focus of this paper was on the kinds of search cost improvements that localization can obtain. In particular, we concentrated on two criteria for localization: abstraction and scope. Abstraction-based localizations are derived from constraint-activation information. Scope-based localizations are based on the relevance of constraints to specific pieces of the plan. We described how Loc, an automatic localization generator, can be used to generate localizations based on both criteria. In empirical tests we showed that scope-based localizations attained the best results.

Future work with localization and Loc will focus on feeding information about the 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 transforms solely on the basis of the problem description – e.g. the *potential* scope (or activation) of constraints. However, during planning, specific constraints may actually be applied to much narrower portions of the plan. Similarly, different applications of the same constraint may be focused on different portions of the plan (e.g. the same electrical constraint might be applied to activities on different floors). We would like to extend Loc to include splitting transforms, thereby allowing constraints to be applied in narrower contexts. One excellent candidate for focussing the splitting process will be the actual parameter bindings assigned to plan variables. Another motivation for utilizing feedback is to guide the application of heuristic scope-based transforms. For example, if consistency maintenance costs are particularly high for certain regions, they would be good candidates for merging.

Acknowledgments

Thanks to John Allen, Rich Keller, Rich Levinson, Barney Pell and the members of the FIA-SPOT reading group for useful comments and stimulating discussions.

References

[1] Christensen, J. "A Hierarchical Planner that Generates Its Own Hierarchies," in *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI90)*, Boston, Massachusetts, pp. 1004–1009 (1990).

[2] 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).

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

[4] Knoblock, C.A., Tenenberg, J.D., and Q. Yang. "A Spectrum of Abstraction Hierarchies for Planning," *Proceedings of the 1990 Workshop on Automatic Generation of Approximations and Abstractions*, Boston, Massachusetts, pp.24–35 (1990).

[5] 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).

[6] Korf, R.E. "Planning as Search: A Quantitative Approach," *Artificial Intelligence (33,1)*, pp. 65-88 (1987).

[7] Lansky, A.L. "Action-Based Planning," in *Proceedings of AIPS-94, the Second International Conference on AI Planning Systems*, Chicago, Illinois (June 1994).

[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 Planning with Diverse Plan Construction Methods," NASA Technical Note FIA-TR-94-05, NASA Ames Research Center (1994). Under review for *Artificial Intelligence Journal.*

[10] 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).

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

[12] Kambhampati, S. "On the Utility of Systematicity: Understanding Tradeoffs Between Redundancy and Commitment in Partial-Ordering Planning," *Proceedings of the 1993 AAAI Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, pp. 67–72 (1993).

[13] Sacerdoti, E. "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence, 5*, pp. 115–135 (1974).

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