N92-63

126951
9P

Localization vs. Abstraction: A Comparison of Two Search Reduction Techniques

Amy L. Lansky

Sterling Software
NASA Ames Research Center (AI Research Branch)
MS 269-2, Moffett Field, CA 94035
LANSKY@PTOLEMY.ARC.NASA.GOV

Abstract

There has been much recent work on the use of *abstraction* to improve planning behavior and cost. Another technique for dealing with the inherently explosive cost of planning is *localization*. This paper compares the relative strengths of localization and abstraction in reducing planning search cost. In particular, localization is shown to subsume abstraction. Localization techniques can model the various methods of abstraction that have been used, but also provide a much more flexible framework, with a broader range of benefits.

1 Introduction

Over the years, several research results have appeared on the use of *abstraction* to guide and improve planning performance [2, 3, 4, 5, 11, 12]. Abstraction techniques restructure a problem and the problem-solving process into a set of "abstraction levels." At the top level of abstraction, the problem is described and solved at the most coarse-grained level of detail. Each successive level is made more concrete than its predecessor by incrementally adding information into the problem description. The use of abstraction can benefit planning if the solution found at 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 the overall planning problem are solved first, and which later. At least two methods have been used within the planning community for creating levels of abstraction: (1) creation of more *concrete* levels of detail by incrementally decomposing abstract actions into more concrete subactions (*operator abstraction*); and (2) creation of more *abstract* levels by incrementally eliminating required action preconditions (*state abstraction*).

Recent work has also appeared on the use of domain *localization* or *decomposition* to structure a problem description and thereby guide and improve planner performance. In this case, search savings are attained via a "divide and conquer" approach to reasoning. A domain and problem description (its actions, definitions, goals, preconditions, and any other constraints or properties) are divided up into *regions*. Semantically, regions define the precise "scopes of interaction" between domain properties and actions. Each region consists of a subset of the overall set of actions and the various properties and goals that pertain to those actions.[1] The localization structure of a domain is then used to break the planning space into a set of smaller reasoning spaces (each constructing a plan for a particular region) and to determine how these spaces are searched. In [9], a localized search algorithm is described, along with analytical and empirical results that demonstrate how exponential savings in search cost can be achieved.

This paper compares the relative strengths of localization and abstraction as heuristics for reducing planning search cost. In particular, localization is shown to subsume abstraction; localization can model abstraction "levels," but also provides a more flexible framework for domain partitioning, with a broader range of planning benefits. Section 2 begins with a characterization of the planning search space and the relative search benefits achievable via localization and abstraction. Section 3 then provides a description of the localized reasoning frameworks of two planners - GEMPLAN [6, 7, 8, 9] and COLLAGE, a new system that builds upon the ideas in GEMPLAN. Analytical and empirical results that describe the cost savings attainable by utilizing lo-

[1]Problem reduction (translation of a goal into subgoals) may also be used to decompose a planning problem [1]. However, this kind of technique may more properly be viewed as a problem solving method rather than a search reduction technique, though search savings may occur as a result.

calization are also summarized, as well as tradeoffs in its use. Next, Section 4 shows how localization can encode the various commonly used methods of abstraction. Finally, Section 5 concludes with further discussion of the strengths and weaknesses of the two techniques.

2 The Planning Search Space

Consider a search space in which each node is associated with a plan and each arc is associated with a plan-construction operation that transforms a plan into a new plan (typically via the addition of actions, relations, or variable bindings). Such a tree directly reflects plan-space search, but can also be mapped onto state-space search. In the latter case, the plan P associated with a node is mapped onto the "state" reached after executing P, and each arc operation is mapped onto the action (i.e., the reasoning the planner must perform in order to add that action) that takes it from one state into the next state. Given this characterization of planning search, we can see that the cost of both state-based or plan-based search can be improved in at least three ways:

1. *Lowering operation cost* – i.e., reducing the cost of each arc or plan-construction operation. Since most planning algorithms are NP-complete in the size of the plan, reducing plan size is one way of lowering operation cost.

2. *Operation ordering* – i.e., choosing a good order in which to apply plan-construction operations. Goal-ordering is one example of this, as are other heuristics for determining how a planning space is searched. A good operation ordering may result in less backtracking, but may also improve solution quality.

3. *Reducing implicit search space size*, typically by lowering the branching factor of the search space. Of course, decreasing necessary backtracking via operation ordering may reduce how much of a space is actually searched. But limiting the applicable operations at each node absolutely reduces the total size of the space.

Both localization and abstraction may be viewed as problem-solving heuristics for reducing planning search cost. Alternatively, they may be viewed as ways of reformulating or recasting a planning problem so that the cost of search required to solve that problem is reduced. Abstraction techniques explicitly break the problem-solving process up into "abstraction levels." At each level, more information is added into the problem definition (e.g., actions are decomposed or preconditions are added) to create a more complex planning problem. Since abstraction levels inherently control the order in which pieces of the problem are tackled, it is a heuristic for *operation ordering*. In earlier stages of the reasoning process, only "higher" level operations, which involve high-level actions or conditions, are applied. This set is expanded as the problem and domain definition is expanded. Although abstraction also initially limits the set of applicable operations at each search node, an inherent reduction of applicable operations is not a guarantee of the abstraction technique once the domain is fully expanded. Rather, it is the job of abstraction-derivation techniques to form abstraction hierarchies that guarantee properties like *monotonicity* [4], which limit interaction between the actions and states of the various abstraction levels.

Rather than dividing a problem definition into abstraction levels, localization divides a problem according to the inherent scope of its actions, properties, and goals. A particular localization or domain decomposition provides a planner with a semantic definition of the scope of all domain actions and properties. Each region may be viewed as a "scope" of reference, with an associated set of actions, definitions, goals, etc. As a result, a localization can be used to determine which domain actions and properties interact, and which are independent. Localization then forms a valid basis for partitioning the planning search space into a set of smaller spaces (one for each region), for focusing the application of plan-construction operations to specific pieces of the plan, and for triggering those operations at appropriate times.

Moreover, unlike abstraction, localization can be used to encapsulate domain information based on any criterion, not just "abstractness." The region divisions are based on the particular qualities and scopes of the domain rather than a particular "abstraction-inducing" technique such as operator or state abstraction. Thus, abstraction-based localizations might be used, but also physically-based, process-based, or temporally-based partitionings, which may be more compelling.

Finally, and perhaps most importantly, localization allows for domain regions that overlap and interact. While it is often difficult to attain a clean partitioning into abstraction levels (often resulting in a collapse of levels or a great deal of interaction between levels), localization embraces the notion that real-world decompositions cannot be neatly decomposed and will naturally entail regional overlap. Thus, the localization technique explicitly provides methods for coping with regional interaction.

In terms of the potential search benefits described above, localization can achieve all three:

1. Plan-construction operations are applied to much smaller *regional* plans. Thus, localization *reduces operation cost.* While abstraction may provide a way of initially working on smaller plans at higher levels of abstraction, ultimately, the scope of the planning algorithms becomes the most detailed plan. Thus, abstraction does little to partition the scope of reasoning and does not inherently improve operation cost.

2. The localized search technique directs search flow so that only "relevant" operations are applied at each point in the reasoning process (i.e., those operations relevant to regions whose plans have been modified). Thus, localization controls *operation ordering.*

3. Since only region operations are applicable at each region search node, localization reduces search space size by *limiting the branching factor at each search node.*

3 Localized Representation and Reasoning

We now explain the localization technique by describing its instantiation in two localized planners, GEMPLAN and COLLAGE. In both systems, a region R is defined by a *region description*:

$<actions(R), definitions(R), constraints(R), subregions(R)>$

Each region R is associated with a search tree, $tree(R)$, whose role is to construct a plan, $plan(R)$, that satisfies all regional constraints, given available actions and definitions. Each plan is a partially ordered set of actions. The set $actions(R)$ defines a set of *action types* which are considered to belong directly to R and instances of which may occur within $plan(R)$. (Note that $plan(R)$ may also include actions belonging to subregions of R.) The set $definitions(R)$ includes any definitions pertaining to activity in $plan(R)$. The set $constraints(R)$ includes "constraints" that must be satisfied by $plan(R)$. Finally, $subregions(R)$ consists of regions belonging to R.[2]

The regions comprising a domain may take on any structural configuration – they may be disjoint, form hierarchies, or even overlap. Semantically, a particular decomposition defines the *scope* of domain properties; the scope of each definition and constraint associated with R is $plan(R)$ – which may be composed only of actions in R and its subregions. It is the role of the domain describer to ensure that these scoping semantics are correct; the planner assumes that they are. The only required criterion for domain decomposition is that each constraint and definition belong to a region that includes *at least* the entire "scope of applicability" of that definition or constraint (but possibly more).

As an example, consider the small construction domain depicted in Figure 1. It has been partitioned into regions that include the activities of an electrician and plumber. These regions include subregions that contain activities at specific walls. Each *wall* region would be associated with definitions and constraints that are relevant only to the actions that can take place at that wall. In contrast, *electrician* (or *plumber*) definitions pertain to *all* activity directly within e. *trician* (*plumber*), as well as all activity at it- vall subregions. Since *wallA* is shared by the . trician and plumber, both the electrical and mbing constraints apply to the activity within ι ::*A*. The constraints directly associated with *wai.A* itself would probably include those relating to coordination of the plumber and electrician activities at that wall. The figure also shows search trees for these regions. Each tree is concerned with building a plan for its region that satisfies all regional constraints. The planning process may thus be viewed as a set of "mini-planners," tied together by the structural relationships between regions.

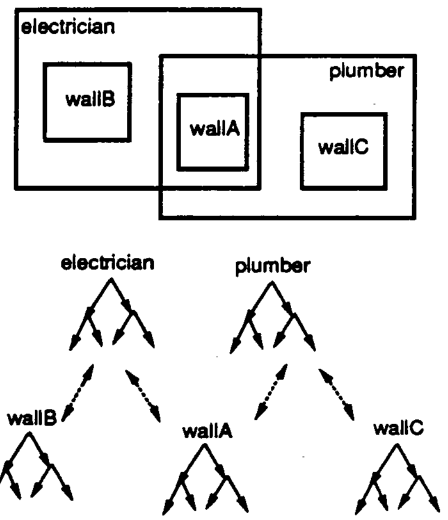

Figure 1: A Localized Construction Domain

3.1 Localizing Traditional Planning Representations

One important distinction between the planning representation of GEMPLAN and COLLAGE and that of traditional planners is the encoding of domain information in terms of "actions," "definitions," and "constraints" rather than STRIPS-like

[2]Section 3.1 describes the relationship between "actions, definitions, and constraints" and more traditional planning representations.

operator descriptions. One reason for this is that it allows domain information to be more easily localized. In a traditional planning representation language, an "action description" is bound up with action preconditions and effects. The "definition" of a particular state predicate is essentially a side-effect of the set of action descriptions within the domain and is thus "distributed" throughout the domain description. Whether or not a literal P is true at some point in the plan is determinable by examining the actions within the plan, along with their defined preconditions and effects, and seeing whether they "combine" to achieve P. The goals that a traditional planner attempts to achieve are a combination of user-provided top-level goals and subgoals that are posted to fulfill action preconditions.

In contrast, the GEMPLAN/COLLAGE framework separates the definition of actions from their preconditions and effects. *Action-type definitions* are simply descriptions of the action types themselves - an action "name" with a set of parameters. For instance, in a blocks world domain, *pick(block)* would define an action type, an instance of which is *pick(a)*. A state predicate is defined separately by an explicit *predicate definition.* In the GEMPLAN implementation of the blocks world, the following definition of *clear(B)* is used:[3]

```
strips_definition(clear(B),
                  [adder(pick(Y),on(Y,B)),
                   adder(put(B,_),true),
                   deleter(put(_,B),true),
                   deleter(pick(B),true)]).
```

A predicate definition includes a list of conditional 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. For example, an action of type *pick(Y)* adds *clear(B)* if *on(Y, B)* is necessarily true just before it, *put(B, _)* always adds *clear(B)*, and *put(_, B)* or *pick(B)* always delete *clear(B)*. Separating predicate definitions from actions descriptions allows actions and predicates to be individually localized (see Section 4). It also makes conditional effects easy to describe; for example, that an action adds a particular literal P in some contexts and another literal Q in others.

In the GEMPLAN/COLLAGE framework, action preconditions and top-level goals are also described as separate entities - they are explicitly defined as *constraints*. For example, in the blocks world domain description, we have:

[3]Capitalized tokens (or the character "_") represent variables. Lowercase is used for constants. Thus, notation of the form *pick(X)* or *pick(_)* is used to denote any *pick* action with a single parameter.

```
constraint(precondition(pick(B),clear(B))
constraint(precondition(put(X,B),clear(B))
```

Such constraints can be easily localized. Also, note how the separation of precondition constraints from predicate definitions clearly distinguishes between necessary action preconditions and those conditions utilized only for describing conditional effects.

Given a framework of actions, definitions, and constraints, planning may be viewed as "constraint satisfaction" rather than backwards and/or forwards chaining on state-based goals and conditions. In GEMPLAN and COLLAGE, a "constraint" is simply any property that the planner knows how to test and make true. The standard STRIPS-based algorithms form only a subset of the possible methods of plan construction in GEMPLAN and COLLAGE - many other kinds of constraint forms and satisfaction algorithms are provided by the two systems. Any of these constraint forms may be used to encode domain properties, and all constraints are appropriately scoped by the localization structure of a domain. Thus, in many ways, both systems may be viewed as general constraint-based reasoners rather than strictly as planners.

3.2 Localized Search

Once a domain has been localized, its regional structure guides how localized search is performed. As described earlier, each *tree*(R) is concerned with constructing a *plan*(R) that satisfies *constraints*(R) given *actions*(R), *definitions*(R), and the actions and definitions of all subregions (and subsubregions, etc.) of R. Each tree node is associated with the region plan constructed up to that point in the search, and each tree arc is associated with a plan modification that transforms a region plan into a new region plan. Upon reaching a node, the planner must choose which region constraint to check next. (Thus, an implicit branching factor in the search space is the set of all region constraints at each node.) If the chosen constraint is not satisfied by the plan at that node, constraint satisfaction algorithms must be applied, resulting in a set of new region plans at the next level down in the tree. A constraint satisfaction algorithm typically adds new actions, relations, and variable bindings to a region plan, and may also generate new subregions. For example, in order to satisfy a precondition constraint, one option is to add an action and appropriate relations that establish that action as an "adder" of the precondition.

Because it is partitioned into regional search trees, localized search is more complicated than the traditional global search utilized by most planners. The localized search algorithm described in [9] has two basic functions: (1) *global correctness*: making sure that all constraints that need to be checked are checked and that appropriate shifts occur between between regional search spaces; and (2) *global consistency*: making sure that all of the plan fragments being constructed (especially those shared by more than one super-region plan) are consistent with each other. This second function is much like that of a distributed database and is ensured by updating all relevant plans for ancestor regions of R, each time search exits from $tree(R)$. Global correctness is ensured, first, by making sure that all regions are searched at least once, and second, by making sure that search eventually occurs for a region R whenever R's plan has been affected by some previous plan modification. GEMPLAN uses a fixed strategy for controlling search flow and consistency maintenance, but COLLAGE allows for more flexible approaches.

In some senses, localized search control may be viewed as a TMS-like strategy for maintaining constraint satisfaction – only "affected" constraints need to be rechecked. Unlike a true TMS, however (which also tries to capture "what affects what"), domain localization is a broad-brush heuristic strategy that need not be accompanied by perpetual and expensive reasoning to update those dependencies. The domain decomposition provides a "cut" at defining scope and interactions; the planner uses it, but never needs to verify it or update it. In this respect, localization provides the same level of heuristic information as abstraction, providing a "useful" partitioning of domain information. However, localization can encapsulate information based on many, perhaps mixed, criteria. Some regions may capture physical structures, others may represent processes, and others may represent abstraction hierarchies within these or overlaid with these. For instance, the construction domain of Figure 1 includes regions that are physically-based (the walls) as well as those that represent contractor "processes." One might view localization as having the ability to capture both "horizontal" as well as "vertical" decomposition.

3.3 Localized Search Benefits and Tradeoffs

In [9] a detailed complexity analysis is provided that highlights the potential benefits and tradeoffs of localized search. That paper also provides some initial empirical results that support this theoretical analysis. This section summarizes these benefits and tradeoffs.

Since the cost of localized search for a particular domain is very dependent on the particular constraints, structure, and problem specification for that domain, the "general" complexity analysis described in [9] was performed on a somewhat idealized domain scenario. The search cost of a global, non-localized domain was compared with that of the same domain, partitioned into a set of m subregions each of which overlaps by some factor k with another region g. An original set of n_c constraints was partitioned among these $m+1$ regions. Table 1 summarizes provides the results of this analysis. Complexity results were calculated assuming that all constraints were either constant, linear, quadratic, or exponential in cost relative to the size of the plan. The table also compares the cost of best-case or worst-case search. Best-case measures the cost of one path through the search space (no backtracking), and worst-case measures the cost of the entire potential space. The term s is the size of the final plan. The term n_f is the number of potential fixes for each constraint. Finally, C is the cost of maintaining consistency, which is assumed to be $O(m^2k)$.

These results show that localized search is nearly always better than non-localized search – in most cases much better. The only exceptions are constant-complexity best-case search (when there is no reduction in the amount of the space searched *nor* in constraint algorithm cost) or when the cost of consistency maintenance overshadows the cost of the search. The amount by which localized search wins over non-localized search is proportional to m (the amount of decomposition), but inversely proportional to mk (the amount of overlap). Thus, increased decomposition is always worthwhile, except for the cost of increased overlap. The gains of localized search become exponential as the complexity of the constraint algorithms increases and the amount of the space actually searched increases. These gains come from three sources, which correspond directly to the three factors described in Section 2:

1. The *cost* of each arc – i.e., *operation cost*. Even if the absolute size of the non-localized and localized search spaces are the same, expensive constraint algorithms are applied to much smaller plans in the localized case.

2. The *search heuristics* provided by localization – i.e., *operation ordering*. Because of the semantic information provided by a localized domain description, the most relevant constraints tend to be applied at the right time, enabling a reduction in the amount of the search space that actually needs to be searched.

3. The *size* of the search space – i.e., *branching factor reduction*. This is because only regional constraints are relevant at each node.

complexity of c(i) and f(i)	*Non-Localized (best-case)*	*Localized (best-case)*	*Non-Localized (worst-case)*	*Localized (worst-case)*
constant (b)	$2bs$	$2b(s+mk)+C$	$b(n_c n_f)^s$	$b(m(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (\frac{n_c n_f}{m+1})^{mk}) + C$
linear (ib)	bs^2	$b(\frac{s^2}{m} + (mk)^2) + C$	$bs(n_c n_f)^s$	$b(s(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (\frac{n_c n_f}{m+1})^{mk}) + C$
quadratic (i^2)	$\frac{2}{3}s^3$	$\frac{2}{3}(\frac{s^3}{m^2} + (mk)^3) + C$	$s^2(n_c n_f)^s$	$\frac{s^2}{m}(\frac{n_c n_f}{m+1})^{\frac{s}{m}} + (mk)^2(\frac{n_c n_f}{m+1})^{mk} + C$
exponential (b^i)	$2b^s$	$2(mb^{\frac{s}{m}} + b^{mk}) + C$	$(bn_c n_f)^s$	$m(\frac{bn_c n_f}{m+1})^{\frac{s}{m}} + (\frac{bn_c n_f}{m+1})^{mk} + C$

Table 1: Complexity Results

Empirical tests were also carried out which bolster these results. In [9], several decompositions of a building-construction domain were compared, as well as the effects of varying the size of the actual building plan constructed. In this domain, constraint cost was fairly low (close to linear for most constraints) and there was no backtracking. Even so, the search cost of the best localization was less than 50% of the non-localized domain configuration. The results also show that increased localization provides increased benefit, except for the added expense due to increased regional overlap. However, as also predicted by the complexity results, the detrimental effects of increased overlap become overshadowed as plan size and search space size increases.

One of the focuses of the COLLAGE project is to flesh out our understanding of localized search by performing many more controlled experiments. The new COLLAGE search control architecture features a constraint-activation and consistency-activation agenda mechanism that allows for various aspects of the search strategy to be easily modified and reconfigured. Using this architecture, we plan to test a suite of search strategies over a suite of problem types that vary in the amount of backtracking required, constraint algorithm difficulty, as well as domain localization structure and problem size. Finally, we also hope to come up with a *localization learning* approach that automatically discovers domain-dependent and domain-independent localization heuristics.

4 Modeling Abstraction With Localization

In order to model traditional planning-based abstraction methods in a localized framework, we must create "levels" of reasoning, representing incrementally more detailed descriptions of the domain. Referring to the characterization of a region description in Section 3, we can see that this can be achieved by incrementally adding regions and/or subregion links, constraints, action types, and definitions. The addition of this information can be done by a special search step that introduces the next "level" of reasoning. Both GEMPLAN and COLLAGE already incrementally add regions during planning, and both systems access relevant domain information in such a way that makes incremental addition of other types of information trivial to accommodate.[4] In addition, the incremental addition of subregion containment relationships adds an interesting "twist" to the types of abstraction levels attainable; a region may be initially visible to some super-regions and then incrementally become a subregion of additional regions, resulting in "mix-and-match" levels of abstraction.

4.1 Operator Abstraction

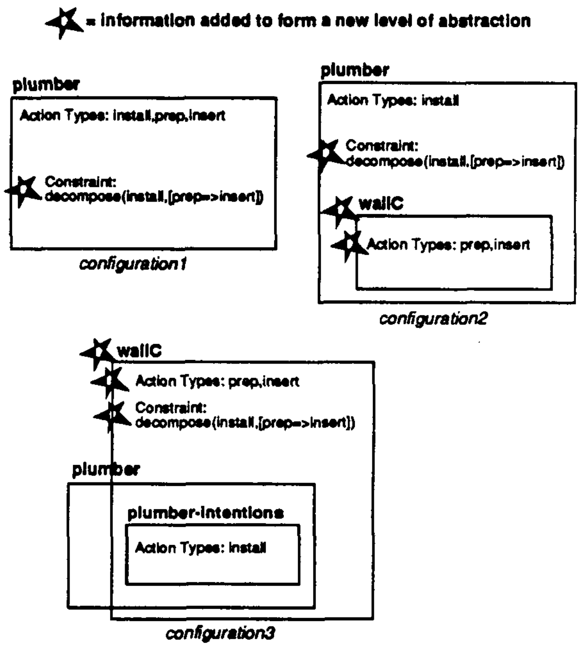
= information added to form a new level of abstraction
plumber
Action Types: install,prep,insert
Constraint: decompose(install,[prep=>insert])
configuration1
plumber
Action Types: install
Constraint: decompose(install,[prep=>insert])
wallC
Action Types: prep,insert
configuration2
wallC
Action Types: prep,insert
Constraint: decompose(install,[prep=>insert])
plumber
plumber-intentions
Action Types: install
configuration3

Figure 2: Operator Abstraction

One of the constraint forms available in GEMPLAN and COLLAGE is the *decompose* constraint, which requires that actions of a specified type be [conditionally] decomposed into one of a set of possible patterns of interrelated subactions. Operator abstraction can be modeled in a localized framework by incrementally adding such action de-

[4] All domain information accessed by the plan-construction operations is represented and accessed in plan-relative fashion. As a result, new constraints, actions, regions, and definitions can be "added" to a plan and thus become newly accessible to the reasoning mechanism.

composition constraints and, optionally, incrementally adding regions which contain the subactions at the "next level down." Different degrees of interaction between "levels" may be achieved, depending on the localization configuration used.

Figure 2 provides three sample configurations for modeling operator abstraction in the construction domain of Figure 1.[5] In all three cases, an *install* action of the plumber is decomposed into two subactions at a lower level of detail, *prep* and *insert*. In `configuration1`, no wall subregion exists. Instead, operator abstraction is achieved by simply adding a decompose constraint to *plumber*. In `configuration2`, the subaction types *prep* and *insert* and a subregion *wallC* containing them are also added, thus creating a new level that includes new action types and a new subregion. In `configuration3`, region *wallC* contains the decomposition constraint and subactions, but overlaps with *plumber* only at the point of the higher level action *install*. Note how, in `configuration2`, the lower-level actions in *wallC* become subject to *plumber*'s constraints, introducing potential interaction between "abstraction levels." In `configuration3`, this interaction does not exist except at region *plumber-intentions*. If only region *plumber* adds *install* actions, no planning interaction will occur once region *wallC* is added (i.e., there will be no need to recheck the constraints in *plumber*), thereby guaranteeing monotonicity in the reasoning process. For more discussion of monotonicity and related properties, see Section 5. Also note how, in general, constraints may refer to actions at mixed levels of detail. Unlike many hierarchical planners, GEMPLAN and COLLAGE allow both actions and their subactions to be present within a plan simultaneously.

4.2 State Abstraction

A localized framework can also model state abstraction in several ways, depending on the desired effect. In Figure 3, three possible configurations are given in which various preconditions and definitions affecting the *install* action and its subactions are incrementally added. `Configuration1` illustrates how action preconditions (or top-level goals) can be added on a per-action basis, by simply incrementally adding precondition (or goal) constraints. If we wished a specific predicate to be completely unavailable until a certain "abstraction level" (achieving a "partitioned hierarchy" [4]), is predicate definition and all precondition or goal constraints that utilize that predicate would not be added until that "level" in the reasoning process is reached. Another option is to incrementally add available "lower level" action types that have been defined to be adders or deleters of a predicate. In this way, action effects (rather than just preconditions) may be incrementally added to the frame of reasoning. In `configuration2`, a literal *clean-site* cannot be "deleted" until the lower-level action type *prep* is added to the domain. This next level also includes a new precondition for *visual-ok* (*unused-site*), as well as a new definition for *unused-site* based on the lower level actions *prep* and *insert*.

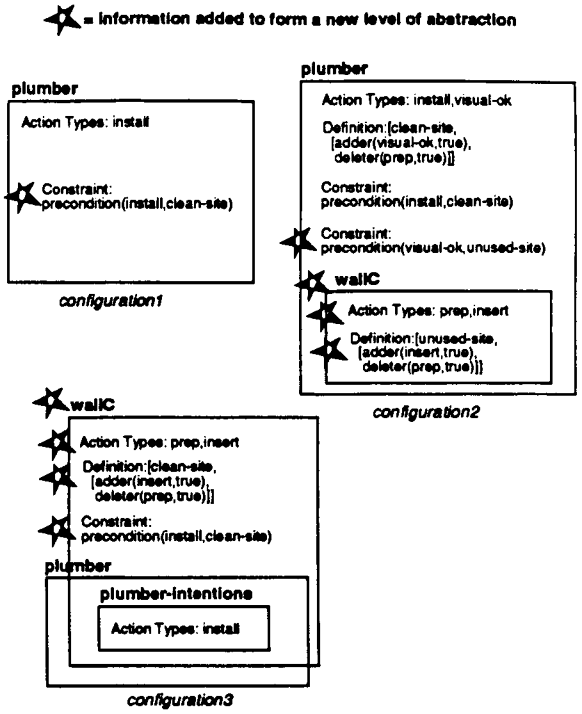
= information added to form a new level of abstraction
plumber
Action Types: install
Constraint: precondition(install,clean-site)
configuration1
plumber
Action Types: install,visual-ok
Definition:[clean-site, [adder(visual-ok,true), deleter(prep,true)]]
Constraint: precondition(install,clean-site)
Constraint: precondition(visual-ok,unused-site)
wallC
Action Types: prep,insert
Definition:[unused-site, [adder(insert,true), deleter(prep,true)]]
configuration2
wallC
Action Types: prep,insert
Definition:[clean-site, [adder(insert,true), deleter(prep,true)]]
Constraint: precondition(install,clean-site)
plumber
plumber-intentions
Action Types: install
configuration3

Figure 3: State Abstraction

One can achieve a strict, noninteracting partition of predicates and actions into levels (i.e. *monotonicity*), by utilizing the strategy depicted in `configuration3`. Here, the new region *wallC* is added which contains a new precondition constraint, actions, and definitions at the next level down. In this case, region *wallC* overlaps with region *plumber* rather than being strictly contained within it. Thus, if we adhere to a regimen in which only region *plumber* adds actions of type *install* (and only region *wallC* can add actions of type *prep* and *insert*), a strict separation of effect would be achieved – changes within region *wallC* would not trigger search within *plumber*, thereby guaranteeing monotonicity.

[5] The constraint syntax used in Figures 2 and 3 has been simplified for illustrative purposes.

5 Discussion

The primary point of this paper has been to show that localization is more general than abstraction - it can capture the same kind of heuristic information, but can also express other forms of encapsulation, with potentially greater benefits. This section discusses the impact of localization on such properties as *monotonicity* and tries to shed some light on other plusses and minuses of localization and abstraction.

In [4], several properties are described that provide a useful basis for the formation of abstraction hierarchies. These include the *upward solution property*, *monotonicity*, and *ordered monotonicity*. By constructing abstraction hierarchies in a way that ensures these properties, guarantees can be made about the completeness of an abstracted search space and the amount of backtracking that will be necessary. In particular, the upward solution property guarantees that decomposing the problem into abstraction levels will not remove completeness from the search space. Monotonicity properties additionally remove the need to backtrack into higher levels of the reasoning space.

If localization is used to represent abstraction, what effect does this have on these properties? Does strictly controlling the ordering of constraint application remove possible solutions? Once actions, constraints, and regions are added into the domain specification, will backtracking to a point in the reasoning space *before* this addition be required? These are precisely the kinds of questions that localization addresses. A localization structure captures the defined semantics of interaction between actions and constraints. If two constraints do not apply to the same pieces of the growing plan, they do not interact and their relative constraint ordering does not make a difference. Likewise, if actions, constraints, or regions incrementally added to the planning problem do not cause triggering of previously defined constraints, a pure refinement strategy is possible - no backtracking will be necessary. And even if the regional configuration of a localization does not by itself *guarantee* independence, search heuristics (that encode knowledge about such guarantees) can be used to block unnecessary backtracking or constraint rechecking.

In sum, if localization is used to capture exactly and *only* the forms of abstraction available in the various systems outlined in [4], then localization will manifest the same properties as those systems. Guarantees about such things as monotonicity are a function of the abstractions or localizations chosen for a specific domain. The techniques used by Knoblock [3] to learn abstractions that guarantee ordered monotonicity, or those used by Christensen [2], could also be used within a localization framework.

But an advantage of using a localized framework is that it can be used to capture *much more* than abstraction. Depending on the domain, physically- or process-based localizations might reap even greater search benefits than abstraction-based localizations. Even though "levels" of reasoning can be modelled, they form only a small portion of the structuring capabilities of localization. While properties such as ordered monotonicity may be useful, they come at a price. Since monotonicity requires noninteraction between levels, it may result in a collapse of the hierarchy. Indeed, this might be fairly common, since real-world problems rarely lend themselves to pure refinement strategies. In a localized framework, there is no need to collapse levels or regions into each other if they are not strictly independent. A localization need not be organized hierarchically and does not necessarily have to engender separate planning "phases." Interactions are handled as a basic mechanism of the search process which directs the flow of reasoning, without necessarily invoking backtracking into a "previous level" of reasoning. Finally, in a localized framework, actions, definitions, constraints, and regions may be incrementally added in flexible ways. "Levels of detail" may be mixed among constraints. The addition of subregion relationships can incrementally and selectively increase the scope of constraints.

Of course, just as for abstraction, the trick is to find a *good* localization that reaps as many search benefits as possible. As discussed earlier, a research focus in COLLAGE is automatically learning such localizations. The key is to find a decomposition that balances decomposition and interaction. Increased decomposition results in finer-tuned localization of constraints, but also results in increased regional overlap and accompanying increases in consistency maintenance costs and potential "thrashing" between regional search spaces. The tradeoff between locality and overlap mirrors the abstraction tradeoff between increasing the number of abstraction levels and increasing the amount of interaction between levels.

Admittedly, the cost of dealing with regional overlap and the complexity of localized search is a limitation of the localization technique. Because abstraction simply partitions the search into ever-growing levels of detail, it can still use global search methods. The management of regional search in a localized framework requires more work. Other problem reformulation techniques may also be more feasible in an abstraction-based framework, where the problem may be "reformulated" before search proceeds at each level. However, this might also be accomplished in a localized framework via incremental *modification* of domain constraints, actions, and definitions.

Finally, localization is also applicable to other kinds of tasks. Localization can be used to encapsulate any kind of domain information – not just STRIPS preconditions, goals, and action decomposition. The technique can be used by any kind of reasoning that can be cast in terms of constraints applied to a partitionable frame of reasoning. Methods based on localized search have already been incorporated into a scheduler [13], another planner that uses abduction as the primary plan-construction mechanism [10], and an image understanding framework [14]. Localization can also aid replanning and plan reuse. If certain pieces of a plan become faulty during run-time, localization provides a good first-cut at which pieces of the plan can be reused and which constraints must be rechecked. Localization-based replanning and reuse is another focus of COLLAGE.

Acknowledgments

Thanks to Steve Minton, Rich Keller, Andrew Philpot, Peter Friedland, John Allen, Smadar Kedar, Mark Drummond, and the referees who have reviewed this paper for their useful comments and encouragement.

References

[1] Bresina, J., Marsella, S. and C. Schmidt. "Predicting Subproblem Interactions," Technical Report LCSR-TR-92, LCSR, Rutgers University (February 1987).

[2] 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).

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

[6] Lansky, A.L. "A Representation of Parallel Activity Based on Events, Structure, and Causality," in *Reasoning About Actions and Plans*, M. Georgeff and A. Lansky (editors), Morgan Kaufmann, pp. 123-160 (1987).

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

[8] Lansky, A.L. "Localized Representation and Planning," in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (editors), Morgan Kaufmann (1990).

[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] Missiaen, L. "Localized Abductive Planning for Robot Assembly," in *Proceedings of the 1991 IEEE Conference on Robotics and Automation*, pp. 605–610 (April 1991).

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

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

[13] Personal communication with M. Zweben about the NASA Ames AI Research Branch scheduling project.

[14] Personal communication with Framentec Applied Artificial Intelligence Group, Cedex 16, 92084 Paris La Defense, France.