Localized Search for Multiagent Planning

Amy L. Lansky

Sterling Federal Systems
NASA Ames Research Center (AI Research Branch)
MS 244-17, Moffett Field, CA 94035 U.S.A

Abstract

This paper describes the localized search mechanism of the GEMPLAN multiagent planner. Both formal complexity results and empirical results are provided, demonstrating the benefits of localized search, A localized domain description is one that decomposes domain activities and requirements into a set of regions. This description is used to infer how domain requirements are semantic ally localized and, as a result, to enable the decomposition of the planning search space into a set of spaces, one for each domain region. Benefits of localization include a smaller and cheaper overall search space as well as heuristic guidance in controlling search. Such benefits are critical if current planning technologies and other types of reasoning are to be scaled up to large, complex domains.

1 Introduction

The focus of this paper is the use of *locality* — the inherent structural qualities of a domain — to control the explosive cost of planning and other forms of reasoning. The use of localized reasoning, while quite intuitive and natural, has not been a fundamental aspect of most AI systems. A localized domain description is one that is explicitly decomposed into a set of regions. Each region may be viewed as a subset of potential domain activity with an associated set of requirements or *"constraints"* that pertain only to the activities within that region. We refer to this delineation of constraint applicability as *constraint localization.* (Throughout this paper we use the term "constraint" very broadly to refer to *any* type of goal or domain property that the planner must fulfill.)

Localized planning is the process of creating a valid domain plan by searching a set of smaller, regional planning search spaces rather than a single large "global* space. Each GEMPLAN search space may be visualized as a plan-construction search tree, where each tree node is associated with a region plan and each arc is associated with a plan modification that transforms the preceding plan into a new plan in order to satisfy some region constraint. Localized search is a powerful technique for reducing the size and cost of the entire planning search space. Although this paper discusses localized search as applied to planning, the technique is generally applicable to any form of decomposable reasoning search.

This research has been made possible in part by the National Science Foundation, under Grant IRI-8715972.

In addition to GEMPLAN, a scheduler, an abduction-based planner, and an image understanding system have already incorporated aspects of GEMPLAN's localized search method.

The GEMPLAN domain representation allows for a myriad of decompositional strategies, including the use of regions that overlap, are disjoint, are organized hierarchically, or form any combination thereof. The allowance for overlap, in particular, renders our localized search method quite useful for realistic domains. Criteria for decomposition are usually suggested by the innate characteristics of a domain - e.g., its physical structure, its behavioral processes, its functional elements, and its abstraction hierarchies. A good decomposition typically reflects several such criteria. Consider, for example, a building-construction domain. Viewed globally, the domain may be described by a set of constraints, some of which describe the structure and requirements for a specific building, some that encode the characteristics of contractors and physical resources, and those that describe construction "rules of thumb.*' Clearly, most of these constraints apply only to a subset of the construction activities to be planned. One way to decompose the domain is according to the physical structure of the building - e.g., to utilize separate regions to model each room. Other regions could model the individual building processes - e.g., the contractors. In most cases, physically-motivated regions and process-based regions will overlap. Figure 1 depicts a possible decomposition for a small construction domain.

The primary goal of localization is to cluster activities into regions so that constraints are applied as narrowly as possible. The actual decomposition chosen will determine the exact scope of applicability of domain constraints - i.e., each region's constraints are assumed to apply only to the activities within that region. However, different localization decompositions will incur different planning costs* As we will show, there is a trade-off between increased decomposition and the resulting increased costs necessary for coping with regional overlap. While most of our empirical tests have utilized user-provided decompositions, we are currently developing a technique for automatically learning good decomposition strategies. This work bears some similarity to Knoblock' learning of domain abstractions [4]; however, Knoblock's abstractions must be strictly hierarchical.

The use of localized reasoning has several benefits. From a representational point of view, locality provides a solution to aspects of the frame problem; constraint localization may be viewed as a frame rule which limits interaction among domain actions and properties. Most importantly, locality provides a rationale for partitioning a potentially explosive global planning space into a set of smaller, localized planning spaces. This has three interrelated benefits: (1) The absolute *size* of the union of a set of localized planning spaces is usually smaller than that of a non-localized space; (2) At each node in the search space, expensive planning algorithms need be applied to much smaller regional plans. Thus, search *cost* is cheaper, even if search space size is unaffected; (3) Since a localized domain description provides information about how constraints and activities interact, it serves as a heuristic for constraint application. In particular, localized search enables the application of constraints only to relevant segments of the overall emerging plan and causes constraints to be applied only *when* then actually need to be. All of these factors facilitate scaling up to large domains.

The notion of localized search is related to several other research efforts. The consistency maintenance techniques used in localized search for dealing with regional overlap, for example, are similar to data consistency methods used by shared, distributed databases. Other planning researchers have looked at methods of problem decomposition in order to reduce search complexity [1, 5], but they have focused primarily on goal reduction and operator reformulation rather than search space decomposition. The goals of controlling the scope of constraint applicability and constraint triggering are shared by work in truth maintenance and constraint propagation. Our use of locality, however, generally produces a much coarser-grained connectivity than that found in a TMS or CSP net. GEMPLAN regions record constraint interrelatedness without the need for incremental update or truth checking; they are thus less expensive. Semantically, however, localization can provide significant information. For instance, even if a constraint syntactically appears relevant to a broader fragment of the overall plan, its inclusion within a region will limit its application to within that region. Localized search also imposes an explicit control over constraint application that is distinct from anything intrinsically provided by a TMS orSP net Recent work in discovering "textures'[1] in CSP networks [3] to control constraint propagation bears some similarity to localization, although CSP constraints are usually extensional and propositional, whereas GEMPLAN constraints are broader in content and usually intensional. One interesting use of locality is deKleer's work on localizing or merging clauses in a TMS [2]. In deKleer's case, merging is done to gain completeness. In localized search, partitioning is done to gain efficiency, but with no loss of completeness.

The rest of this paper is organized as follows. Section 2 provides an overview of the GEMPLAN architecture and a description of the localized search algorithm. This

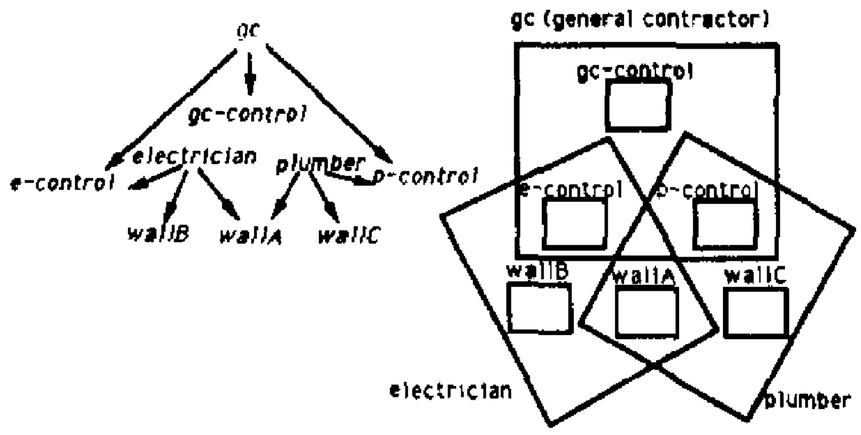

Figure 1: A Localized Construction Domain Description

discussion is made concrete in Section 3, which describes a simple planning scenario. Section 4 discusses various tradeoffs inherent in localized search, and then substantiates these intuitions with a complexity analysis. This is followed by the empirical results of Section 5.

2 The GEMPLAN Planner

GEMPLAN is a planner designed for multiagent domains that require complex coordination. It has been applied to multiagent blocks-world problems, the Tower of Hanoi, and several versions of a construction domain. The system includes an execution facility, and has the ability to apply constraints before or during execution- It may thus be viewed as a combined pre-planner/dynamic-planner. While the existing system is primarily designed for pre-planning, we have begun implementation of a next-generation GEMPLAN system, called COLLAGE, that will span the pre-planning/dynamic-planning spectrum in a more seamless fashion.

GEMPLAN differs from standard hierarchical planners in several ways. Besides its use of localized search, GEMPLAN has the ability to satisfy a broad range of domain "constraint forms," not simply the attainment and maintenance of state conditions. Note that we use the term "constraint" in a very broad sense - i.e., to refer to any sort of property that the planner knows how to test and make true. The system includes a set of general-purpose constraint satisfaction algorithms for a set of constraint forms, which may be further augmented by user-supplied constraint-satisfaction methods. The default constraint algorithms are fully general for partially-ordered plans and perform the task of plan construction by introducing actions, action interrelationships, and variable bindings. GEMPLAN's current constraint repertoire includes: (1) The attainment and maintenance of goal conditions and preconditions - i.e., the traditional planning algorithms used by STRIPS-based planners. Actions may have conditional effects, and established conditions are protected; (2) Action decomposition. GEMPLAN allows for reasoning about actions at mixed levels of detail, rather than confining itself to reasoning "one level at a time," as do some hierarchical planners [10]; (3) A variety of temporal and causal constraints; (4) Required patterns of behavior expressed as regular expressions; (5) CSP constraints on potential values of plan variables.

More details on GEM PLAN appear elsewhere [6, 7, 8], The rest of this paper focuses on GEMPLAN's localized search mechanism, A detailed account of GEMPLAN's localized search implementation can be found in [9].

2.1 Localized Search Overview

Part of GEMPLAN's domain specification input is an explicit decomposition of domain action and constraint information into regions. GEMPLAN uses this information to create a search space for each region, rather than utilizing a single global search space. Each search space is concerned with building a plan for its region that satisfies all regional constraints. The planner may thus be viewed as a set of "mini-planners,"tied together by the structural relationships between regions.

The localized search algorithm has two basic functions: (1) controlling the flow of constraint application - in particular, shifts between regional search spaces; and (2) maintaining the consistency of the overall search space. Besides assuring that all regions are searched at least once (to satisfy goal constraints), localized search typically shifts from one regional search space to the search space for a region *R* only if previous plan modifications have impacted *R'*slan. That is, search will flow only to those regions whose plans have been affected and whose constraints truly need to be checked. Planning for a region can impact the plan for another region only if they share a common portion of the overall plan (i.e., they overlap). And because the search spaces of overlapping regions reason about shared subplans, localized search must make sure that they share a consistent view of these subplans.

2.2 Region Description

Let us assume that a domain is specified as a set of regions. Each region *R* is defined by a *region description: <action$(R), subregions(R), constraints(R),tree(R)>.* The set *adions(R)* consists of action *types,* instances of which may occur directly within *R* (but not within a subregion of *R).* The set *subregions(R)* consists of subregions belonging to *R.*For each such subregion Q, we use the notation *Q C R.* The set *constraints(R)* includes constraints that pertain to activities within *R* and its subregions. Finally, each region is associated with a plan-construction search tree *tree(R).*

Consider the construction domain of Figure 1. It has been partitioned into regions corresponding to the activities of an electrician, plumber, and general contractor. These regions are further decomposed to include subregions that contain the activities of the electrician and plumber at various walls as well as their respective "control" activities (these might include communication actions between contractors). Each wall region would be associated with constraints that are relevant only to the actions taking place at that wall; e.g., wallA might include constraints relating to coordination the plumber and electrician. The electrician, plumber, and gc region constraints, which apply to all their subregions, describe more global requirements; e.g., the gc constraints might describe how the contractors communicate instructions.

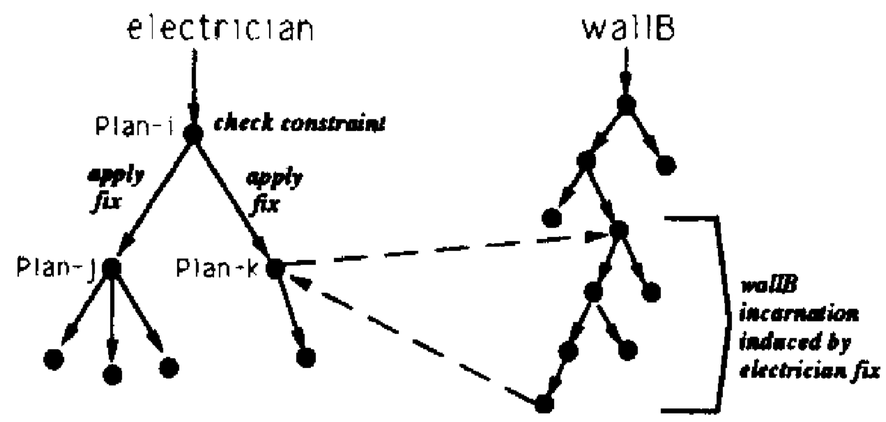

Figure 2: Region Search Trees

2.3 Plan Representation

The focus of regional search is to incrementally construct a *region plan.* Each region plan consists of a *local region plan* and a set of *sub plans* (the region plans of its subregions). For example, if R1C*R* and *R2* C -R, the region plan for *R* will include a local region plan for *R* and region plans for R_1 nd *R2.* GEMPLAN associates all plan information with the smallest region that encompasses that information. This assures that plan information is visible to all relevant regions. The region plans of *RI* and R_2will thus include all plan information that deals exclusively with *R1*and R2 respectively. The local region plan of *R* will include plan information that deals specifically with activities in *R* or that pertains to relationships among R R_1, and *R2* (and therefore cannot be associated strictly with *RI* or *R2).*

2.4 Region Search Trees

Figure 2 depicts portions of planning search trees for the electrician and wallB regions of our sample domain. Each tree reflects search through a space of "plan construction" operations - i.e., it is a plan-construction search space (rather than a state search space). 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 or "fix" 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 relevant constraints at each node.) If the chosen constraint is not satisfied by the plan associated with that node, constraint satisfaction algorithms or "fixes" must be applied (there may be several fix algorithms for each constraint, as well as many possible solutions or "fixes" per fix algorithm), resulting in a set of new region plans at the next level down in the tree. A GEMPLAN fix typically adds new actions, relations, and variable bindings to a region plan, and may also generate new subregions. Deletion of plan information can only be accomplished by backtracking (this restriction will be lifted in the COLLAGE system). GEMPLAN uses, by default, a depth-first search strategy. It tries constraints in the order supplied by the domain specification, fixes in the order supplied by GEMPLAN's internal constraint mechanism, and backtracks only if all fixes fail for a particular constraint. However, since search should optimally be driven by domain-dependent information and

the structure of the growing plan, GEMPLAN allows for flexible tuning of tree search. The order in which constraints and fixes are applied, as well as backtracking schemes, can be made context dependent.

2.5 Transfer Between Trees

In the current GEMPLAN system, transfer of control flow between regions occurs according to a fixed algorithm. Aside from assuring that each region is visited at least once to address goal constraints, transfer between regions generally occurs in response to information transmitted to the search mechanism by a fix. Suppose we are in *tree(R).* After applying a fix for one of R's constraints to R's region plan, the fix must return a subset of *IVs* subregions, R_1, ...,.R_m, whose plans have been affected by the fix. The GEMPLAN search algorithm will then inhibit further search within *tree(R)* until *tree(R.l)...,tree(Rrn)* are all satisfactorily searched. As depicted in Figure 2, if electrician affects the subpian for region wallB via the introduction of new actions there, search within *tree*(electrician) cannot safely proceed until wallB's tree is searched and its constraints are rechecked and satisfied.

Notice how shifts between parent and child regions induce a partitioning *on* the child's search tree. We call these search fragments *incarnations* - search within the child is reinvoked or "reincarnated" each time the child's plan is modified in some way by a parental fix. Each time a child's tree is reincarnated, all of its affected constraints must be rechecked. One restriction on GEMPLAN's search control mechanism is that all search strategies (e.g., alternative backtracking schemes) must be applied within the confines of an individual incarnation. This greatly simplifies the problem of search consistency.

2.6 Global Correctness and Consistency

Two important challenges of localized search are making sure that all regional constraints are checked when they need to be (global correctness) and assuring that all regional search tree information is consistent (global consistency). This would be fairly straightforward if domain structure were strictly hierarchical. However, since we allow for regional overlap, added effort is required. For example, if a fix in tree(electrician) affects region wallA's plan, it is not enough to simply recheck wallA's constraints and return to tree(electrician). Region plumber's representation of wallA's subplan must also be updated within tree(plumber), and search must also eventually occur within tree(plumber) to recheck its constraints. We call this overall process *completion.*

Let us consider completion in more detail. When search within an incarnation for region J? is terminated, all changes for that region must be reflected within the tree of every ancestor region *A* of *R.* This will involve update of subplans and other information relating to *R* at relevant nodes in *tree(A).* Assuming that the final plan for R's *preceding* incarnation was *Rplan,* the nodes to be updated will be precisely those that have information about *Rplan,* Next, some method must be used to ensure that A's search space is eventually reincarnated. This could be done via an auxiliary facility that keeps track of which regions must be checked. We are taking this approach in COLLAGE; the facility will fully control region incarnation and will also allow for parallel incarnation of regions.

In the current GEMPLAN system, the DAG formed by the inclusion relation C must be completed by a spanning tree. This tree is then used as a guideline for controlling regional transfer. Each region, except some designated "top" region must have a parent via the C relation or be artificially "assigned" a parent via the spanning tree. For example, gc, electrician, and plumber are not subregions of any enclosing region. Although they do not logically belong to another region as far as constraint applicability, we must make sure that each has a "parent" for search control purposes. We denote this pseudo-parental relationship by C_P and choose gc as the "top" region, with electrician C_p gc and plumber C_p gc. Search initiates at the "top" region and transfers from parent to child only if C or C_P holds and if the child's region needs to be checked.

3 Example

We now clarify the preceding discussion with a simple planning scenario. Let us assume that the electrician, plumber, and wallA regions are associated with the following constraints:[1]

ELECTRICIAN CONSTRAINTS:
(1) action(install-socket(wallA,locAl))
(2) *decompose(install-socket(W,L)*,
{V.electprep(L) => W.insertsocket(L)})

PLUMBER CONSTRAINTS:
(1) action(install-pipe(wallA,locAl)
(2) decompose(install-pipe(W,L),
{W.plmnbprep(L) = W.insertpipe(L)})

VALLA CONSTRAINTS:
(1) (forall L)
[(forall prep:{electprep(L).plumbprepCL)})
pattern((prep)*=>)]
(2) f*cfs(*[[electprep,insert socket],
[plumbprep,insertpipej])

The first electrician constraint requires a socket to be installed in wallA, An action constraint simply results in the addition of an action to the plan.[2] The second constraint requires that each install-socket(W,L) action be decomposed into an electprep action followed

[1] This informal description is intended for didactic purposes only. A full GEMPLAN domain description requires specification of regional action types and domain structure, allows for specification and instantiation of region types, may include many more types of constraints, and may include various search heuristics. Capitalized tokens denote variables. Each variable has a scope limited to the constraint in which it is found. An action of form X,Y denotes an action Y occurring at location I.

[2] One way to introduce actions into the plan might have been via STRIPS-based goal conditions or preconditions. In this domain, it is more convenient to add the actions explicitly, since only one type of action is available for attaining the desired effect.

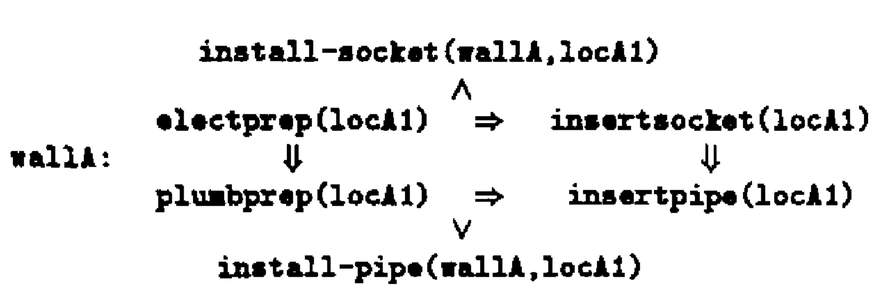

Figure 3: A Construction Plan

by an insert socket action at wall W, location L, The plumber constraints are similar. In this case, one pipe is to be installed in wall A. The two wallA constraints pertain to the coordination of the electrician and plumber actions at that wall. The first constraint states that all electprep and plumbprep actions at the same location must follow a certain pattern - they must be totally ordered by the temporal relation =>. The second constraint additionally requires that the electrician and plumber have access to the same locations of wallA on a first-come-first-serve basis. The constraint provides a set of action pairs and has the following semantics: any required execution ordering of the first actions in each pair (in this case, required orderings between prep actions) will determine the ultimate ordering of the second actions in each pair (the ordering of insertsocket and insert pipe actions). Since a total ordering is forced on all prep actions at the same location, this will force electricians and plumbers to insert their devices in common locations on a first-come-first-serve basis.

We now describe the construction of the plan in Figure 3. Reasoning begins at the "top" region gc, which, in this case, has no constraints of its own, but is responsible for invocation of the electrician and plumber search trees. In this case, electrician is invoked first and its first constraint adds the specified install-socket action into the electrician plan. Constraint 2 then decomposes this action into electprep and insertsocket actions at wallA, causing changes in the wallA subplan of electrician. Before search continues within tree(electrician), search within tree(wallA) must occur. Both wallA constraints are checked, but both are satisfied at this point. The newly completed incarnation of wallA therefore does not add any new information to the subplan for wallA associated with electrician, but the process of completion causes the new version of the wallA plan (that includes the changes made by electrician) to be inserted appropriately into nodes of iree(plumber).

All electrician constraints are now satisfied. Search returns to gc, which invokes search in tree(plumber). The plumber constraints cause the addition of the install-pipe action and its decomposition into the appropriate subactions at wallA. Search must now occur for the affected wallA region. This causes the actions electprep(locAI) and plumbprep(locAI) to be forced into some total order (electprep(locAI) => plumbprep(locAI) is chosen) and then, as a result of the second wallA constraint, a similar ordering is imposed on insertsocket(locAI) and insertpipe(locAI). The new wallA plan is appropriately inserted into both

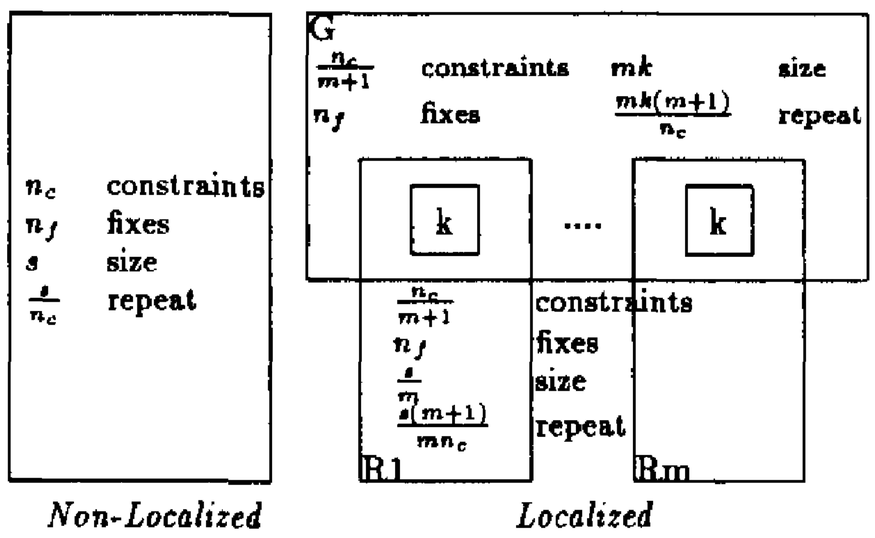

Figure 4: Non-localized and Localized Domains

tree(electrician) and tree(plumber) (due to completion). All plumber constraints are now satisfied and search returns to gc. The constraints within electrician are then rechecked (due to the changes at wallA), but they are still satisfied. Search then terminates successfully.

4 Complexity Analysis

It is clear that no general complexity result can be given for localized search - the size and complexity of the planning search trees for a particular problem will depend on the structure of the domain, the constraints associated within each region, the complexity of their satisfaction algorithms, the domain search heuristics, and the specific domain problem being solved. In order to provide some theoretical estimate of the benefits of localized search, however, we present a complexity analysis that compares search of a non-localized domain with search of a simple localized version of the same domain. We provide best- and worst-case search costs, assuming that constraint algorithms are either all constant, linear, quadratic, or exponential in cost (obviously, most domains will have a mixture of these).

Although our analysis is quite idealized, it is corroborated by the empirical results of Section 5 and jibes with our basic intuitions about the tradeoffs inherent in localized search. In particular, it is clear that the completion costs necessary for dealing with overlap are nontrivial. These include search-tree update of ancestral regions to maintain consistency, and possibly wasteful rechecking of affected regional search trees (for example, the rechecking of the electrician constraints at the end of the preceding scenario). The problem of needless rechecking of regional trees can be mitigated by maintaining tighter relevancy hooks between actions and constraints (which GEMPLAN optionally provides) to inhibit unnecessary constraint checking. However, the problem is impossible to avoid completely. On the other hand, needless rechecking would be worse in global search,

Our analysis is based on the non-localized and localized domain configurations depicted in Figure 4. For both domains, we assume a total of n_e constraints, n_f possible fixes for each constraint, and a final plan consisting of s actions. The cost of checking any constraint on a plan of size j is $c(j)$ and the cost of fixing a plan

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

of size *j* is /(j). For the localized case, we assume the domain consists of *m* regions $R_1...R_m$ and a region G. The actions in the final plan are divided equally among the regions $R_1...R_m$, so that each builds a plan of size $\frac{s}{m}$. Each of the *R1....Rm* regions also contains a subregion consisting of *k* actions that overlaps with region *G.* Thus, G's region plan consists of *mk* actions. The n_c constraints of the original problem are evenly distributed among G, R_1, *...Rm so* that each region is associated with $\frac{n_c}{m+1}$ constraints. (Note that the "k" regions have no local constraints associated with them.)

Let us now consider the cost of a generic region search tree. For a region i, we assume there are n_{Ci} constraints, that each constraint has n_j fixes, and that the final size of the region plan is s_i. Because a constraint fix may violate previously satisfied constraints, constraints may need to be repeatedly checked and fixed. We call the number of times the search must cycle through the constraints the search *"repeat factor"* Assuming that a region *i* has a repeat factor *of* r_i, its tree depth is $r_i n_{c_i}$, with average depth to adding an action being $\frac{r_i n_{c_i}}{s_i}$. (Thus, we assume that at most one action is added per fix. In most domains, many actions may be added per fix.) We then assume an implicit search space that alternately branches due to choice of a constraint (the costs *c(j)* accumulated due to constraint testing) and choice of a fix (the costs f(j) accumulated due to constraint fixing). By '"best-case search" we mean depth-first search without backtracking - i.e., the cost of one path from the root to the leaves of the search space. The cost of best-case search for region i is

$$\sum_{0 \le j \le s_i} \frac{r_i n_{c_i}}{s_i}(c(j)+f(j)).$$

In contrast, worst-case search cost measures the cost of searching the entire space:

$$\sum_{1 \le j \le r_i n_{c_i}} n_{c_i}^j n_f^{j-1} c((j-1)\ div \frac{r_i n_{c_i}}{s_i}) + n_{c_i}^j n_f^j f((j-1)\ div \frac{r_i n_{c_i}}{s_i}).$$

To compare the complexity of these formulae for the non-localized and localized cases, *we must* set the repeat factor r_i for each region. For this analysis, we let r_i be $\frac{s_i}{n_{c_i}}$ – that is, we assume that exactly one action is added per fix, that the size of the plan is larger than the number of constraints, and that the depth of the region tree is equal to the number of actions in the region plan. The complexities of all cases are summarized in Table 1. For all of the localized search cases, we must also add to the total cost of the search trees an additional completion cost *C,* Assuming that completion occurs each time an action is added within a region of overlap and that the cost of each completion operation is a function of the plan data structures that, must be updated, *C* is $0(rn^2k)$,

As can be seen, 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 of search space size nor constraint algorithm cost) or when the cost of completion overshadows the cost of the search. The amount by which localized search wins over non-localized search is proportional to the amount by which s dominates both $\frac{s}{m}$ (the plan size of each of the subregions R_1.-R_m) and *mk* (the plan size of *G).* Thus, increased decomposition is always worthwhile, except for the cost of increased overlap (reflected in the size of *mk* and the cost of completion C). The gains of localized search become exponential as the complexity of the constraint algorithms increases and the bushiness of the search space increases. These gains come from three sources:

1. The *size* of the search space. This size reduction become more significant as the bushiness of the search space (the amount of backtracking) increases.
2. The *cost* at each node. Even if the absolute size of the non-localized and localized search spaces are the same, expensive constraint algorithms need be applied to much smaller plans *in* the localized case.
3. The *search heuristics* provided by localization. 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 search space bushiness.

5 Empirical Results

All of our empirical experiences with GEMPLAN bear out the efficacy of localized search. Our largest application thus far is a building-construction domain which requires both resource allocation and temporal coordination. The application was used to test and compare a variety of localization configurations. In our study, all of the localized configurations were superior to the non-localized configuration. Even though planning for this domain manifested best-case search in both the localized and non-localized cases (there was no backtracking), GEMPLAN attains speedups of greater than 50% using the best decomposition. This speedup can be attributed to a reduction in the cost of the constraint algorithms rather than a decrease in search space size. Exponential speedups can be expected for domains with bushier search spaces and more complex constraint algorithms.

Test Case (49 actions)	*Number of regions*	*Overlap* <count,size>	*Weighted Sum* <total,overlap>	*Timing* total=constraints+search
non-localized	1	<0,0>	<8624,0>	110.08=77.38+32.70
localized(1)	24	<28,134>	<1544,104>	82.55=51.55+31.00
localized(2)	16	<5,32>	<2894,88>	76.55=58.40+18.15
localized(3)	19	<18,76>	<1834,88>	61.24=46.31+14.92
(97 actions)				
non-localized	1	<0,0>	<27354,0>	877.80=628.13+249.67
localized(1)	37	<52,236>	<4736,104>	503.02=354.65+148.37
localized(2)	28	<5,32>	<10602,88>	718.90=559.55+159.35
localized(3)	31	<35,102>	<5468,88>	435.30=345.27+90.03

Table 2: Empirical Results

Table 2 provides a variety of statistics about the tested decompositions (the system is written in Prolog and runs on a SPARC workstation). The "number of regions" column gives the total number of regions that have at least one constraint and one action in the final plan (this number corresponds to the m + 1 factor in the complexity analysis). The "overlap" column gives two figures: an overlap *count* and an overlap size. The overlap count is the sum, over all regions of overlap, of how many parents each region of overlap has. To obtain overlap size, we additionally multiply the count for each region of overlap by the number of actions it contains. Overlap count and size provide good estimates of completion cost and are related to the *mk* factor in the complexity analsysis. The "weighted sum" column gives a pair of numbers: *total* and *overlap.* The total weighted sum is the sum, over each constraint check and fix that was performed, of the product of plan size and a constraint-difficulty weighting (ranging from 1 to 4). It thus gives a good measure of the constraint checking and fixing cost. The overlap weighted sum is the same sum over constraints belonging to regions of overlap. Finally, the "timing" column gives three numbers: the total CPU time in seconds, which is a *sum of the* time spent dealing with constraint application and the remaining time, spent mostly in search.

The table provides results for the creation of a 49-action construction plan and for a 97-action construction plan. Results are included for four domain decompositional strategies, with the 97-action plan simply having more walls, contractors, etc. The first strategy is non-localized. The localized(l) configuration is highly decomposed but also has significant amounts of overlap between regions. The localized(2) case has less localization and much less overlap. Case localized(3) has an intermediate level of both localization and overlap, and attains the best results in both cases.

As with our formal analytical results, these results show that increased localization provides increased benefit, except for the added expense due to regional overlap. However, notice that, in the 97-action case, localized(l) is faster than localized(2). This shows that, as plan size increases, the cost of dealing with overlap is overshadowed by search and constraint costs. Even though the size of the search space was not significantly different among the various decompositions (because there was no backtracking), search cost still increased with decreased localization. This is because the plan structures become unwieldy as plans become large and the search implementation is affected by the size of these structures.

6 Conclusion

This paper has described an algorithm *for* localized search, as well as complexity and empirical results that illustrate how localized search can provide substantial gains in performance. The idea is quite intuitive and natural, but has, surprisingly, not been a fundamental aspect of most AI systems. Its application to automated reasoning is vital if such systems are to meet the requirements of large, complex domains.

Acknowledgments

Lode Mission helped design and implement GEMPLANE locals ized search algorithm and Anna Karlin assisted with the complexity results. John Bresina, Mark Drummond, Megan Eskey, Andrew Philpot, Monte Zweben, and the 1JCAI reviewers provided useful comments.

References

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

[2] deKleer, J. "Exploiting Locality in a TMS," *AAAI-90,* Boston, Massachusetts, pp. 264-271 (1990),

[3] Fox, M.S., Sadeh, N., and C. Bay lean. "Constrained Heuristic Search/[1] *IJCAI-69,* Detroit, Michigan, pp. 309-315 (1989).

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

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

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

[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. "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).

[9] Missiaen, L. "Localized Search," Technical Note 476, AT Center, SRI International, Menlo Park, CA 94025 (1989).

[10] Wilkins, D.E. "Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence (22,3)* pp. 269-301 (1984).