tools that we have built to facilitate plan sketching (namely, an interactive sketch editor and a sketch space exploration aid). Finally, we close with a discussion of related work and conclusions.

Planning Model

We employ a hierarchical task network (HTN) model of planning, based loosely on that of [Erol et al. 1994].

The cornerstones of HTN planning are *task networks* and *templates* (alternatively, operators or methods). Informally, a task network is a partially ordered set of tasks to be achieved, along with conditions on the world state before and after tasks are executed. Templates specify methods for reducing an individual task to some set of subtasks, under appropriate conditions. HTN planning consists of taking a description of an initial world state, an initial task network, and a set of templates for task refinement, and then searching for templates that can be applied to reduce the initial task network to a set of executable tasks.

Formally, we define a task network $<\mathcal{N},\mathcal{L},\mathcal{W}>$, where $\mathcal{N}$ is a set of task nodes, $\mathcal{L}$ is a set of ordering constraints on those nodes, and $\mathcal{W}$ is a set of world constraints. An HTN planning problem is defined by $<O, \mathcal{T}_0, W_0>$, where O is a set of templates, $\mathcal{T}_0$ is an initial task network, and W_0 is a set of propositions describing the initial world state. A template o is characterized by its purpose *Purpose(o)* (i.e., the tasks to which it can be applied), the preconditions for applying the template *Preconds(o)*, and the task network *Tasks(o)* to which a task matching the purpose can be reduced by applying the template. The tasks, constraints, and goals in the task networks and templates are defined using a first-order language with existential interpretation of variables.

Tasks can be either *primitive* or *nonprimitive*, with the former having no possible refinements. A solution to an HTN problem consists of a refinement of the original task network to a network of primitive tasks for which all constraints can be resolved. A solution is characterized by a *plan refinement structure* $<P,N,D>$, where P is the set of task networks produced, N is the set of nodes in any of the task networks, and D defines a directed acyclic graph of the refinement relations from a node to each of its descendants.

Each node in a plan refinement structure has attributes defined by its associated task. Key attributes for sketch interpretation include the task for the node *Task(n)*, the ancestor node *Ances(n),* the template that has been used to refine that node *Template(n)*, and the bindings for the refinement *Bindings(n)*. We use the notation p^{σ} to denote the application of bindings σ to object p (a template, term, proposition). The notation $\sigma_1 \cup \sigma_2$ denotes the composition of bindings. With this notation, $Task(n)^{Bindings(Ances(n))}$ denotes the instantiated task for node n.

Tolerant Plan Sketch Compliance

We begin by defining a plan sketch.

Definition 1 (Plan Sketch) A *plan sketch* is a set of tasks.

Note that the tasks within a plan sketch can be primitive or nonprimitive, ground or nonground.

The work in [Myers 1997] focused on the concept of *plan sketch compliance*, namely, finding a plan refinement structure that embeds an instantiation of the plan sketch. Definition 2 formalizes this requirement.

Definition 2 (Plan Sketch Compliance) A plan refinement structure $H=<P,N,D>$ is *compliant* with a plan sketch S iff there is a substitution β such that for every sketch task $A \in S$, there is some node $n \in N$ with $\sigma = Bindings(Ances(n))$ such that $Task(n)^{\sigma} = A^{\beta}$.

Robust plan sketching requires a less stringent condition on solutions than that of compliance from Definition 2. This weaker condition must account for both (a) user misconceptions about the task domain (i.e., situations where the user has incorrect models of when and how activities can be undertaken), and (b) background knowledge that may be incorrect or incomplete. In this paper, we focus on two types of problem within sketches that derive from user misconceptions and faulty domain knowledge:

- *Type 1:* violations of constraints from the templates used to interpret a plan sketch
- *Type 2:* sketch tasks that do not map to any high-level goal (i.e., *orphaned* tasks).

We define the weaker notion of *maximal compliance* to accommodate these problem types. In contrast to the concept of full compliance from Definition 2, maximal compliance captures the notion of embedding a maximal subset of the original sketch within a plan refinement structure while minimizing constraint violations.

The formal definition of maximal compliance builds on the concept of *conditional compliance*. Conditional compliance for a plan sketch allows a designated set of constraints to be ignored. For a set of templates O, define O/C to be the set of templates that is identical to O except that all template preconditions that unify with constraints in C have been removed.

Definition 3 (Conditional Compliance) A plan refinement structure H for a problem $<O, \mathcal{T}_0, W_0>$ is *conditionally compliant* with a sketch S and set of conditions C iff H both is compliant for S and is a plan refinement structure for the problem $<O/C, \mathcal{T}_0, W_0>$.

Definition 4 (Maximal Compliance) Let $H=<P,N,D>$ be a plan refinement structure and S_0 be a plan sketch. H is *maximally compliant* with S_0 iff H is conditionally compliant with some sketch $S \sqsubset S_0$ and conditions C, and there is no plan refinement structure H' such that for some conditions $C' \sqsubset C$ and sketch S' where $S \sqsubset S' \sqsubset S_0$ either:

- H' is conditionally compliant with S' and C, or
- H' is conditionally compliant with S and C'.

Maximal compliance characterizes the class of solutions to a planning problem that best reflect a given sketch, subject to the constraints of the background knowledge. Ideally, a robust sketch interpretation algorithm should aim to identify one or more plan refinement structures that are maximally compliant. However, domain complexity may preclude finding such optimal solutions in practice.

Robust Sketch Interpretation

In this section, we define an algorithm for robust sketch interpretation that is motivated by the notions of conditional and maximal compliance. The algorithm builds substantially on the 'nonrobust' algorithm of [Myers 1997]. We first provide a high-level summary of that method, and then define a set of extensions and modifications that enable robust sketch interpretation.

Summary of the Nonrobust Method

The nonrobust method consists of two steps: (a) an initial *abduction phase* for linking sketch tasks to a high-level goal, and (b) a subsequent *refinement phase* in which the abduction results guide decision making to produce a full plan that is compliant with the sketch.

The abductive phase produces a collection of *chains* for each sketch task, where a chain encodes an abstraction path from a sketch task to a designated high-level objective through the templates defined for the planning domain.

Definition 5 [Abductive Chains] The *abductive chains* for a task A and objective G are the set of labeled linear graphs

$$\mathrm{A} = T_n \xrightarrow{[O_n,\sigma_n]} T_{n-1} \xrightarrow{[O_{n-1},\sigma_{n-1}]} T_{n-2} \xrightarrow{\cdots} T_1 = G$$

where each O_j is a template with purpose T_{j-1} and a subtask Q_j such that σ_j is a most-general unifier of Q_j and $T_j^{\,\beta}$ for $\beta = \cup_{n \geq i > j} \sigma_i$.

We say a task A is *orphaned for an objective* G (or just *orphaned* when the objective is clear) iff there are no abductive chains linking A to G.

The abductive chains are used to guide HTN refinement in order to ensure that the resultant plan contains each of the anchors in the specified sketch. Standard task refinement involves selecting a template that applies to a given task (i.e., the template's purpose unifies with the task and all template preconditions are satisfied). For sketch processing, refinement must further restrict template choices and extend variable substitutions so that the resultant plan structure is *consistent* with at least one chain for each sketch task. Consistency requires that there be a path in the hierarchical plan structure from the top-level objective to a leaf node for which the choice of template is identical to that of the chain, and all variable substitutions are consistent. An inability to identify a compatible set of abductive chains for a refinement step indicates that the current plan cannot be expanded to a complete plan that is compliant with the original sketch; hence, further exploration of that option is pointless.

Tolerating Sketch Problems

To accommodate the two classes of sketch problem described above, we generalize and extend the nonrobust algorithm in three ways. First, violated preconditions for template application are ignored temporarily in both the abduction and refinement phases, provided they are deemed *potentially fixable* (discussed below). Second, orphaned sketch tasks are ignored during the refinement phase. Finally, a *repair phase* is added in which detected problems are resolved.

Plan Sketch Repairs

We define four types of repair: *drop constraint*, *drop task*, *modify task*, and *replace task*.

- *DropConstraint(c)* – ignore the constraint c.
- *DropTask(T)* – delete task T from the current sketch.
- *ModifyTask(T,i,v))* – change the ith argument of sketch task T to be v
- *ReplaceTask(T1,T2)* – replace sketch task $T1$ with task $T2$

When considering repairs performed by a human (as opposed to automatically), these repair types can be categorized according to what they say about user versus system expertise. The *drop constraint* repair would be invoked in situations where the user's knowledge overrides that of the system. In contrast, application of the other repairs indicates a preference for the system's knowledge over that of the user (as reflected in his original sketch).

To provide focus, we employ two criteria to limit the applicability of repairs: (a) *relevance of the repair*, as captured by a requirement for deductive linkage between sketch tasks and violated constraints, and (b) *prespecified domain knowledge* that identifies classes of constraints and tasks to which the repairs apply.

Deductive Linkage Deductive linkage requires a logical relationship between a sketch task A and a violated constraint c through an abductive chain. Specifically, some argument to a sketch task A is connected to some argument in the violated constraint c via unification constraints defined by the templates within the abductive chain. This linkage introduces the potential (but not a guarantee) that a change that involves the relevant sketch task argument could eliminate the violation *c*. For example, a sketch task that designates the use of a certain class of helicopter for an airlift operation might lead to violation of a constraint higher up in an abductive chain related to lift capacity. Switching to a more powerful class of helicopter could fix the problem.

In the definitions below, we use the proposition

$$Links(Task(a_1, \ldots ,a_n), i, P(b_1, \ldots ,b_m), Chain)$$

to indicate that within the abductive chain *Chain*, there is deductive linkage from argument a_i in $Task(a_1, \ldots ,a_n)$ to b_j in some predicate $P(b_1, \ldots ,b_m)$, where $P(b_1, \ldots ,b_m)$ is a precondition for a template used in the chain abstraction.

Domain Knowledge Prespecified domain knowledge is used to restrict the classes of task and constraint to which various types of repair apply. We consider three categories.

A. *Droppable Constraints.* Droppable constraints correspond to predicates with a 'soft' interpretation in that they denote preferences or guidelines rather than gating conditions. For example, a template for a helicopter airlift may require wind speed below a certain threshold; a planner may decide to drop that constraint in the event that the current wind speed only slightly exceeds the threshold and all other requirements are satisfied.

B. *Modifiable Task Arguments* A task argument is categorized as modifiable to indicate that changes to that argument are allowed. For example, with the task *FLY(start, destination, flight)* in a travel planning domain, it would make sense to consider alternate flights but not start or destination locations.

C. *Replaceable Tasks* A task is categorized as replaceable to indicate that alternatives for that task can be considered. [2]

We represent these declarations as follows, using *KB* to refer to the predefined knowledge base of the planning system and x_i and y_j to denote variables. The statement

$$KB \models DroppablePredicate(P(x_1, \ldots,x_m))$$

indicates that any predicate that unifies with $P(x_1, \ldots x_m)$ is considered droppable for sketch repair; similarly

$$KB \models ChangeableTask(Task(x_1, \ldots ,x_m),i)$$

indicates that the *i*th argument of any task that unifies with $Task(x_1, \ldots ,x_m)$ can be modified for sketch repair, and

$$KB \models ReplaceableTask(Task(x_1, \ldots ,x_m), Task(y_1, \ldots ,y_n))$$

indicates that any task that unifies with $Task(x_1, \ldots ,x_m)$ can be replaced by a task that unifies with $Task(y_1, \ldots ,y_n)$.

We can now formally characterize the class of *induced repairs* for a given sketch and its abductive chains. The induced repairs constitute a minimal set of relevant repairs to consider when repairing a sketch.

Definition 7 (Induced Repairs) The set of *induced repairs* for a sketch *S* with abductive chains *Chains* consists of

(a) $DropConstraint(P(b_1, \ldots,b_m))$ for any unsatisfied constraint $P(b_1, \ldots b_m)$ in *Chains* such that $KB \models DroppablePredicate(P(x_1, \ldots,x_m))$

(b) $DropTask(Task(a_1, \ldots ,a_n))$ for any task $Task(a_1, \ldots ,a_n) \in S$ that is either orphaned, or for which there is some unsatisfied constraint $P(b_1, \ldots b_m)$ and some $C \in Chains$ such that $Links(Task(a_1, \ldots ,a_n),k,P(b_1, \ldots ,b_m),C)$, for some $1 \le k \le n$

(c) $ModifyTask(T(a_1, \ldots ,a_n),i,v)$ for any task $T(a_1, \ldots ,a_n) \in S$ that is orphaned, or for which $KB \models ChangeableTask(Task(x_1, \ldots ,x_n),i)$ and there is some $C \in Chains$ and unsatisfied constraint $P(b_1, \ldots b_m)$ such that $Links(Task(a_1, \ldots ,a_n),i,P(b_1, \ldots ,b_m),C)$

(d) $ReplaceTask(Task(a_1, \ldots ,a_n), Task'(b_1, \ldots ,b_m))$ for any task $T(a_1, \ldots ,a_n) \in S$ that is orphaned, or for which $KB \models ReplaceableTask(Task(x_1, \ldots ,x_{n1}), Task'(y_1, \ldots ,y_{n2}))$ and there is some unsatisfied constraint $P(b_1, \ldots,b_m)$, and $C \in Chains$ such that $Links(Task(a_1, \ldots ,a_n),k,P(b_1, \ldots,b_m),Chain)$ for some $1 \le k \le n$

For cases (b) through (d) in Definition 7, we say that the repair *covers* the orphaned sketch task $Task(a_1, \ldots ,a_n)$; for cases (a) through (d), we say that the repair *covers* the violated constraint $P(b_1, \ldots,b_m)$. The set of *potentially fixable* constraint violations is defined to be the constraint violations covered by the induced repairs.

The induced repairs provide a means to focus the repair process. Because the space of possible sketch changes can be enormous (as discussed further below), this filtering is essential for restricting the number of options considered.

Within a mixed-initiative framework, one can envision user modifications to a plan sketch that go beyond the induced repairs. Such changes could reflect additional user knowledge about the domain, or a change in strategy from that embodied in the original sketch.

[2] More generally, the properties of droppability, modifiability, and replaceability should be characterized as preference orderings. We will address this issue in future work.

ProcessSketch$(S, C, \langle O, T_0, W_0 \rangle)$

- *Step 1 [Abduction]:* Generate abductive chains *Chains(T)* for each task $T \in S$ while ignoring potentially fixable constraint violations
  - Set: $Orphans \leftarrow \{T \in S \mid Chains(T) = \{\}\}$
- *Step 2 [Refinement]:* Generate a task refinement structure H that is
  - consistent with at least one abductive chain for each $T \in S - Orphans$, and
  - ignores potentially fixable constraint violations

  If no such task refinement structure exists, then return *failure*.
  - Set: $V \leftarrow$ the potentially fixable constraint violations for H
- *Step 3 [Repair]:*
  - *Step 3a:* If $V = Orphans = \{\}$, then return solution $\langle H, S, C \rangle$.
  - *Step 3b:* Else repair the sketch:
    - $S' \leftarrow S$
    - $C' \leftarrow C$
    - Nondeterministically select a set of induced repairs $\{r_1, \ldots r_m\}$ to cover $v \in V$ and $T \in Orphans$ If no such set exists, then return *failure*.
    - Perform the repairs as follows:
      - If $r_i = DropConstraint(v)$: $C' \leftarrow C' \cup \{v\}$
      - If $r_i = DropTask(T)$: $S' \leftarrow S' - \{T\}$
      - If $r_i = ReplaceTask(T1,T2)$: $S' \leftarrow \{S' - \{T1\}\} \cup \{T2\}$
      - If $r_i = ModifyTask(T(a_1, \ldots, a_n), i, d)$: $S' \leftarrow S' - \{T(a_1, \ldots, a_n)\} \cup \{T(a_1, \ldots, a_{i-1}, d, a_{i+1}, \ldots, a_n)\}$
- *Step 4 [Validation]:* Invoke *ProcessSketch*$(S', C', \langle O/C', T_0, W_0 \rangle)$

Figure 1. Algorithm for Robust Sketch Processing

Sketch Repair Algorithm

Figure 1 presents our algorithm for robust sketch processing.[3] Processing a sketch S for a problem $\langle O, T_0, W_0 \rangle$ would involve a call to *ProcessSketch*$(S, \{\}, \langle O, T_0, W_0 \rangle)$; the results returned (via Step 3a) would be a modified sketch S^*, a set of conditions C^*, and a plan refinement structure H^* that is conditionally compliant with S^* and C^* for $\langle O, T_0, W_0 \rangle$ (see Definition 3).

Steps 1 and 2 correspond to the abduction and refinement phases of the nonrobust algorithms from [Myers, 1997], although modified to ignore potentially fixable constraint violations and orphaned sketch tasks. Step 3 nondeterministically selects and applies induced repairs to cover all detected problems, yielding a modified sketch S' and collection of dropped constraints C'. Step 4 recursively invokes the sketch processing algorithm for S' and C' to produce a plan refinement structure that is conditionally compliant with the revised sketch (if one exists) or to identify additional problems to repair.

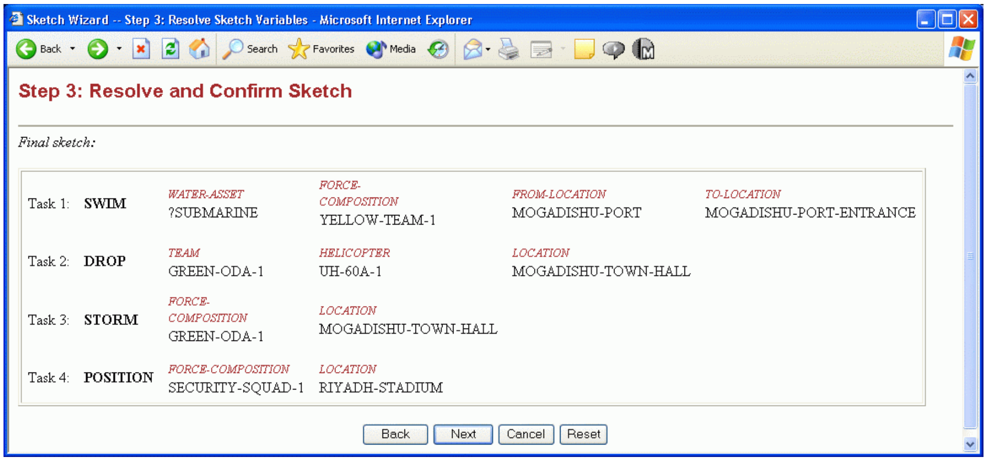
The algorithm as stated does not guarantee maximal plan sketch compliance (see Definition 4), although it could easily be restructured as an optimization process to identify maximal solutions. As discussed further below, we believe that optimization is an inappropriate goal because of the potentially explosive size of the repair search space. Furthermore, our experience indicates that while users prefer solutions that are close to a proposed sketch, maximal compliance is generally not necessary.

[3] To simplify the presentation, the algorithm ignores the potential for repairs that preempt each other (e.g., one repair changes an argument of a sketch task while a second replaces the sketch task with a different task).

Mixed-initiative Repair

The algorithm for sketch repair in Figure 1 does not commit to a specific implementation design. One option is to automate fully the algorithm, including the process of selecting and applying repairs. In the general case, the space of candidate sketch revisions to consider during each call to *ProcessSketch* will be of size $O(k^v)$ where k is the number of induced repairs for a violation and v is the number of violations. While v could be expected to be a relatively small number (say, in the range 5-10), k could be quite large. In particular, *modify task* repairs could encompass changes to any of a task's arguments, and may need to consider a broad range of possible values for each. A fully automated approach to sketch repair would require powerful heuristics to be effective for such a large space.

Our interests lie with more user-centric planning aids, which led us to develop a mixed-initiative realization of the sketch progressing algorithm. In our framework, the system identifies violations and possible repairs while the user selects repairs and directs the overall search. The framework is designed for iterative use, with a human planner incrementally refining a sketch in response to detected problems until finding a satisfactory solution.



One important characteristic of the algorithm from Figure 1 for a mixed-initiative approach is the articulation of a separate repair phase subsequent to the abduction and refinement phases. Delaying repairs until after abduction and refinement complete (as opposed to performing repairs while chains or refinement structures are constructed) means that a plan structure is available to ground the repair process. This context is important for two reasons. First, the user is not making a decision in a vacuum; rather, it is possible to understand the potential impact of a repair on the current plan. Second, interactions with the user are limited to a single candidate solution, thus providing focus. Work in the collaborative problem-solving community views focus as an essential requirement for coherent user-system interactions [Rich and Sidner, 1998]. The work of [Allen and Ferguson, 2002] similarly builds on a candidate solution (the 'straw plan') to guide mixed-initiative planning.

Our implementation differs slightly from the algorithm of Figure 1 in that it does not generate a single complete plan refinement structure in Step 2. Instead, it computes a set of *expansions*, each of which amounts to a least-commitment partial HTN structure that embeds the sketch and all derived consequences. In particular, expansions do not make commitments that are not required to connect sketch tasks to the high-level goal. For example, a sketch for a hostage rescue objective that contained only tasks related to reconnaissance would yield expansions limited to the reconnaissance subportion of the plan. This switch to expansions was motivated primarily by our desire to support a more user-centric planning process, where strategic decision making is left to the user. It would be straightforward to extend the approach to support generation of complete plans for sketches using standard HTN techniques.

Sketch Example

To illustrate the sketch-processing capabilities within PASSAT, we consider an example from a special operations domain that has motivated much of our work. The example focuses on a hostage rescue scenario in which a group of hostages is being held captive by guerrillas in Mogadishu's town hall. Riyadh Airport has been selected as the jumping-off location for the mission while the hostages are to be evacuated to Riyadh Stadium. The high-level task for this plan is represented as

```
RESCUE-HOSTAGE(MOGADISHU-TOWN-HALL,
               RIYADH-AIRPORT,
               RIYADH-STADIUM)
```

Figure 2 shows a sketch that consists of four tasks: (1) a reconnaissance force (Yellow-Team-1) swimming from a submarine (denoted by the variable `?SUBMARINE`) at Mogadishu Port to the port entrance, (2) inserting a combat team (Green-ODA-1) at the town hall via a UH-60A helicopter, (3) having the combat team storm the town hall, and (4) positioning a security team at the evacuation site. The labels above each task argument identify that argument's role in the task.

Processing of this sketch by PASSAT yields three expansions, with a range of three to four violated constraints in each. The expansions interpret the role of the sketch tasks somewhat differently; for example, one expansion interprets the `DROP` task as part of the hostage extraction effort while the others interpret it as part of a reconnaissance operation.

The user can select one of these expansions and explore options for repairing its associated problems. Figure 3 summarizes the constraint violations (top) and the hierarchical task/template structure (bottom) for one expansion; sketch tasks are highlighted. This expansion does not contain any orphaned tasks.

Figure 4 displays the window that is presented to the user to repair the original sketch. The window summarizes the repair options for each violation, which may consist of dropping the constraint, changing a parameter for a designated task, or making no repair. (Our interface does

Violated Constraints

VC1. `(SITUATION-TYPE RIYADH-STADIUM HOSTILE)`
VC2. `(DISTANCE-< RIYADH-AIRPORT MOGADISHU-TOWN-HALL (RANGE UH-60A-1)`
VC3. `(> (SEA-TEMPERATURE MOGADISHU-PORT-ENTRANCE) 40)`
VC4. `(PLATOON-SIZED SECURITY-SQUAD-1)`

Expansion Task Structure

```
Task: RESCUE-HOSTAGE(MOGADISHU-TOWN-HALL, RIYADH-AIRPORT, RIYADH-STADIUM)
Template: Hostage-Recovery-To-A-Potentially-Unstable-Area
  Task: ADVANCED-RECON(COUNTRY-OF(MOGADISHU-TOWN-HALL))
  Template: Advanced-Recon-Of-Target-Area
    Task: RECON-SEAPORTS(SOMALIA)
    Template: Recon-Seaports-In-Area
      Task: RECON(MOGADISHU-PORT)
      Template: Recon-With-Covert-Ground-Force
        Task: EXFILTRATE(YELLOW-TEAM-1, MOGADISHU-PORT, ?TO-LOC)
        Template: Swim-Exfiltrate-To-Submarine
         Task: SWIM(?SUBMARINE, YELLOW-TEAM-1, MOGADISHU-PORT, MOGADISHU-PORT-ENTRANCE)
  Task: RESCUE-AND-RECOVER(?FORWARD-POINT, MOGADISHU-TOWN-HALL, ?RECOVERY-LOCATION)
  Template: Rescue-And-Recover-Hostages
    Task: STORM(GREEN-ODA-1, MOGADISHU-TOWN-HALL)
    Task: INFILTRATE(GREEN-ODA-1, RIYADH-AIRPORT, MOGADISHU-TOWN-HALL)
    Template: Helicopter-Insertion-Rope
      Task: DROP(GREEN-ODA-1, UH-60A-1, MOGADISHU-TOWN-HALL)
  Task: PROVIDE-SECURITY(RIYADH-STADIUM)
  Template: Site-Defense-Large-Reaction-Force
    Task: POSITION(SECURITY-SQUAD-1, RIYADH-STADIUM)
```

Figure 3. Violated Constraints and Plan Structure for the Selected Expansion

not yet support *replace task* repairs.) Because the use of constraint dropping and task parameter changes is restricted by predefined domain knowledge, only some of these repairs may apply in each case.

To support the user in changing a task parameter, the interface provides a drop-down list of candidate values. This set consists of instances for the type associated with that argument, with values that lead to violation of the given constraint (in accord with the deductive linkage from the sketch task to the constraint) explicitly marked as such.

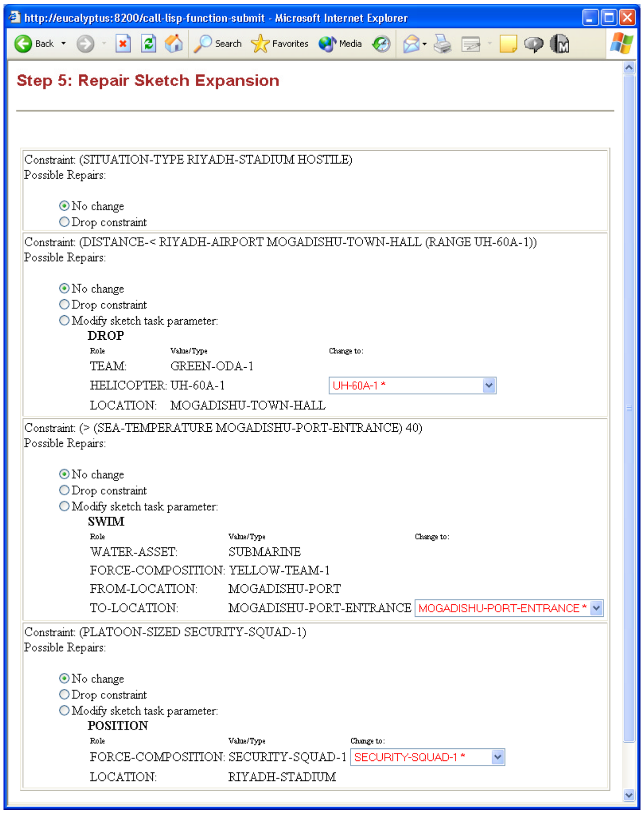
As one approach to repairing the chosen expansion, the user could perform the following repairs:

- drop the constraint *VC1*
- modify the *Helicopter* argument of the `DROP` task to be `UH-60L-1` rather than `UH-60A-1`, given that UH-60Ls have greater range (to address the violated constraint *VC2*)
- drop the constraint *VC3*
- modify the *Force-Composition* argument of the `POSITION` task to be `SECURITY-PLATOON-1` (to address the violated constraint *VC4*)

Given a set of repairs, PASSAT attempts to validate the revised sketch by reinterpreting it while ignoring the dropped constraints. In this case, the repairs resolve the original problems but introduce a violation of the constraint `(COMBAT-EFFECTIVE SECURITY-PLATOON-1)`. This new problem can be repaired by changing the *Force-Composition* argument to be `SECURITY-PLATOON-2` (i.e., a platoon that has been certified ready for combat). Processing of this revised sketch yields a single expansion with no constraint violations.

Figure 5 displays a snapshot of PASSAT's interface after sketch processing has completed. The large frame on the left contains a hierarchical decomposition of the current plan refinement structure, showing the insertion of the final expansion for the Hostage-Rescue task. Items next to folder icons are tasks that have been expanded; items next to star icons are nonprimitive tasks that can be expanded further; items next to document icons are primitive tasks. Sketch tasks appear italicized and highlighted in bold font. The frame on the upper right shows the current *agenda* – the list of planning steps the user must perform to address outstanding issues. PASSAT maintains this agenda automatically to assist a user in managing the planning process. Constraints that the user chose to drop as part of the repair process appear highlighted on the agenda. Planning tasks that remain to be expanded are also added to the agenda. The frame on the lower right shows the list of *information requirements* – sources of information that have been identified by the user or PASSAT's planning knowledge as relevant to the planning process.

At this stage, the user could continue developing the current plan, by using any of PASSAT's capabilities for interactive planning, or by providing a plan sketch for a nonprimitive task.



Sketching Tools

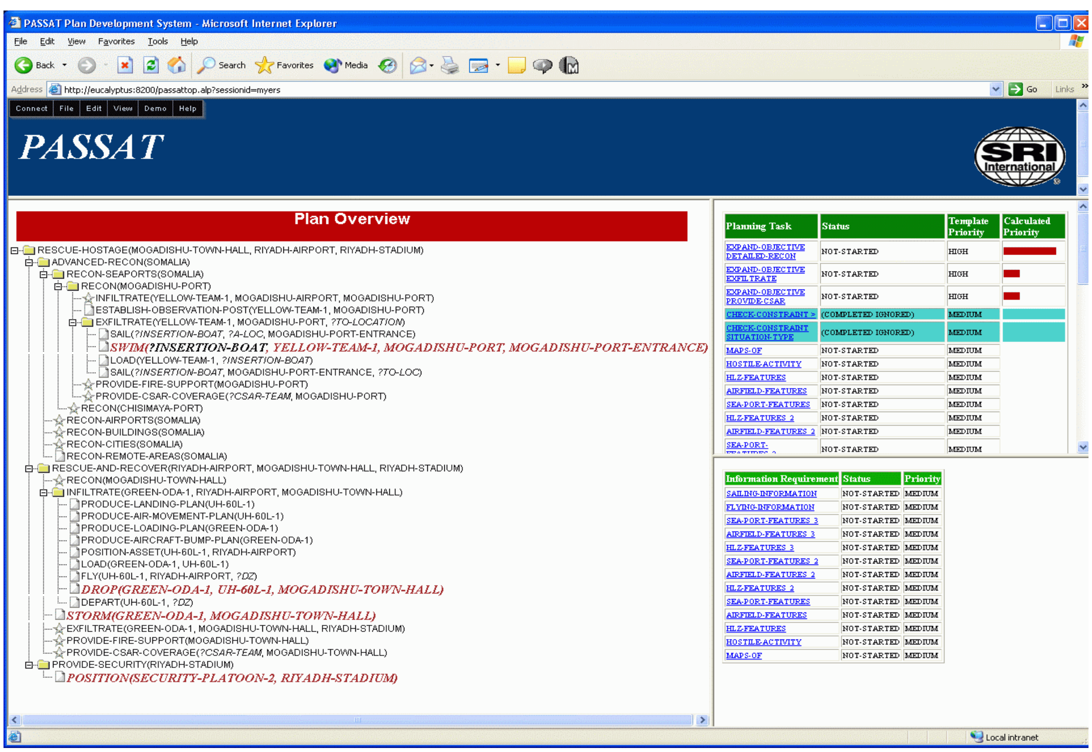
Mixed-initiative systems require powerful and flexible interfaces to facilitate interactions with a user. To support mixed-initiative sketch repair, we developed two interactive tools: a *sketch editor* and a *sketch space exploration aid*.

Sketch Editor

Sketch specification involves defining the tasks that comprise a sketch and their arguments. PASSAT provides an interactive editor to simplify this process. With this editor, the user first selects a set of tasks to be included in the sketch, and then specifies the arguments for those tasks. Allowed arguments consist of variables and all instances of the corresponding type for that argument. Figure 3 displays a final sketch created within the editor.

To help the user focus on relevant choices, the sketch editor incorporates context-sensitive presentation of options to the user for both task and argument selection.

- *Task selection:* The editor exploits linkage among templates to limit task selection for a sketch to tasks that could possibly appear in any expansion of the 'objective' currently under consideration. This filtering helps to eliminate many irrelevant options, thus both reducing clutter from the task selection menu and preventing the user from pursuing many fruitless avenues.



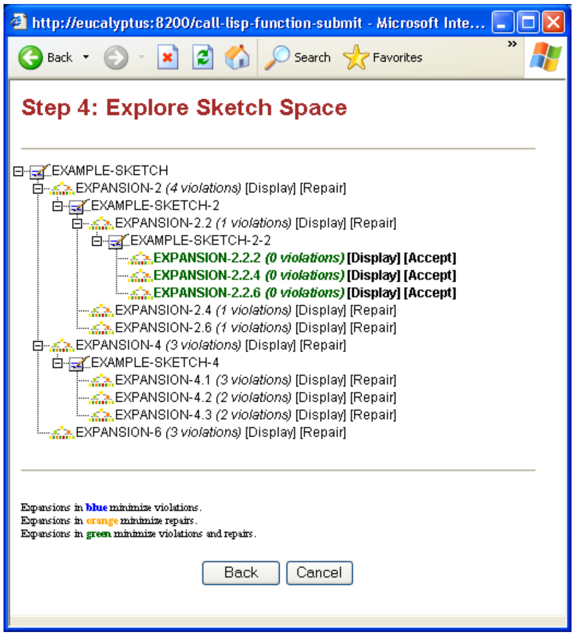


Figure 6. Sketch Space Exploration Tool

- *Argument selection:* It is often the case that many candidate values for a task argument fail to satisfy the preconditions of any templates that could be applied to the task. Eliminating such values from consideration prevents exploration of many dead-ends. However, one design requirement for PASSAT was the flexibility to let a user think 'out of the box'. In particular, PASSAT's constraint reasoning allows certain constraints to be overridden at the user's discretion. For this reason, the possible values presented to the user are flagged to indicate whether or not they satisfy all associated constraints.

This type of structured plan editor eliminates the possibility of syntactic mistakes (e.g., undefined tasks or arguments, use of inappropriate argument types) that can be a source of great frustration to a user. In doing so, it allows the user to focus on the conceptual design for a sketch.

Sketch Space Exploration Tool

The space of possible expansions for a given sketch can be dauntingly large, especially when interpretation is tolerant of invalid sketches. To support a user in navigating this large space, we have developed a *sketch space exploration* tool that aids a user in managing the sketch refinement process (see Figure 6). The tool is organized around a tree structure that reflects the space of sketches and expansions that a user has explored. The root of the tree corresponds to the initial sketch; it contains a descendant node for each expansion of the sketch. Each revision of an expansion in turn generates a descendant sketch node, from which a recursive structure emerges.

For a sketch node, the user can choose to generate expansions all at once or incrementally. For an expansion, a user can view the template structure and the detected problems. Expansions with minimal problems and minimal numbers of expected repairs to address those problems are highlighted. (One repair could fix multiple problems, thus these values can differ for a given expansion.) Eventually, the exploration tool will contain mechanisms to summarize and compare expansions and sketches.

Related Work

The NuSketch system [Forbus eta al. 2001] provides a framework for creating graphical sketches of plans (specifically, for military courses of action) via a drawing metaphor. As with our work, these sketches are intended to provide outlines rather than complete plans, but in a pictorial rather than logical language. NuSketch is focused on interpretation of visual inputs and the adequacy of mechanisms for specifying sketches visually, in contrast to our emphasis on interpreting sketches relative to a knowledge base of plan templates and helping a user refine a sketch to a satisfactory solution.

Qu and Beale's work on cooperative response generation provides a mixed-initiative framework for constraint-based variable assignment problems [Qu and Beale 1999]. Users can perform 'repairs' by changing selections or dropping constraints. The system detects violations and can assist the user by proposing new values and summarizing possible solutions. While similar to our mixed-initiative sketch repair, this work does not incorporate any notion of plans. The authors note that, while there has been much work on cooperative response generation, most of it does not consider interactions among choices.

Our work on sketch interpretation shares with plan recognition techniques the objective of finding a plan that 'covers' a set of specified tasks (see [Carberry, 2001] for a comprehensive overview of the field of plan recognition). These two lines of work differ, however, in several respects. One difference is that the plan recognition work is grounded in the assumption that there is a single intended plan to be determined; in contrast, our work supports the more general notion of identifying a range of possible interpretations for a given sketch. A second fundamental difference relates to the starting point: plan recognition techniques assume a complete, ordered set of tasks for a plan, while our model of a plan sketch consists of a partial and unordered set of tasks. In particular, plan recognition work does not consider the problem of extending a partial plan to a complete solution. Finally, most plan recognition work has been done in the context of STRIPS models of planning, in contrast to our focus on HTN models (although see [Gertner and Webber, 1996] for another HTN-based approach).

Most plan recognition work has assumed that observed actions (the analog of our sketch tasks) are part of a valid plan for an undetermined goal. However, there have been some notable attempts to address the problem of recognition of faulty plans. Classifications of different types of plan-based misconceptions are presented in [Pollack 1986, Quilici et al. 1988, van Beek et al. 1993], with a comprehensive and detailed list provided in [Calistri-Yeh 1991]. The emphasis in that work is on identifying user misconceptions, with no consideration given to potential problems in the underlying domain knowledge. Misconceptions can be broadly characterized in terms of missing actions, violated/unsupported preconditions, and unsupported actions. Within the context of plan sketching, only violated preconditions and unsupported actions make sense (since the plan is only partially specified). Many of these papers also present methods to detect misconceptions and (at times) suggest potential fixes. [Calistri-Yeh 1991] incorporates a probabilistic model of a user to identify 'more likely' explanations for observed actions. Such a model could be useful within the context of our work to focus the user on expansions and repairs with greater expected relevance.

In the long term, we are interested in tools that support user updates to background planning knowledge when gaps or errors are detected. [Cohen and Spencer 1994] present an ATMS-based method for incremental updates to plan recognition structures when knowledge is added.

Conclusions

Plan sketching provides a powerful paradigm for user specification of complex plans. Plan sketching can help a user quickly outline the key aspects of the plan, capitalizing on the system to fill in less important details around the sketch. In addition, it can serve as the basis for an exploratory process that allows a user to consider a variety of options when developing a plan.

Robustness is critical to ensuring that a plan sketching tool is usable and helpful. Robustness requires the ability to identify differences between a user's outline for a plan and what the planning knowledge within the system supports as possible, as well as mechanisms to address those problems.

The work presented here has defined an approach to robust plan sketch interpretation that accommodates two categories of problem: violated applicability conditions and extraneous actions. This approach has been embodied within a mixed-initiative plan sketching framework in which a system identifies options for repair while a user selects candidate interpretations and repairs.

Areas for further work include broadening a sketch to include user constraints and temporal information, and developing tools to improve user understanding of the sketch space (specifically, summarization and comparison tools for sketches and expansions).

Acknowledgments. This work was supported by DARPA under Air Force Research Laboratory Contract F30602-00-C-0058.

References

Allen, J. and Ferguson, G. (2002). Human-machine Collaborative Planning. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Calistri-Yeh, R. (1991). Utilizing User Models to Handle Ambiguity and Misconceptions in Robust Plan Recognition. *User Modeling and User-Adapted Interaction*, 1(4).

Carberry, S. (2001). Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11(1-2).

Cohen, R. and Spencer, B. (1994). Specifying and Updating Plan Libraries for Plan Recognition Tasks. In *Proceedings of IEEE Conference on Tools for AI*.

Erol, K., Hendler, J., and Nau, D. (1994). Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Gertner, A. S. and Webber, B. L. (1996) A Bias towards Relevance: Recognizing Plans where Goal Minimization Fails. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Guindon, R. (1990). Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 5(2).

Forbus, K. D., Ferguson, R. W., and Usher, J. M. (2001). Towards a Computational Model of Sketching. In *Proceedings of Intelligent User Interfaces*, Sante Fe, New Mexico.

Myers, K. L. (1997). Abductive Completion of Plan Sketches. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, AAAI Press.

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., and desJardins, M. (2002). PASSAT: A User-centric Planning Framework. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, TX.

Pollack, M. (1986). A Model of Plan Inference that Distinguishes between the Beliefs of Actors and Observer. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguisitics*, N.Y., N.Y.

Qu, Y. and Beale, S. (1999). A Constraint-Based Model for Cooperative Response Generation in Information Dialogues. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI Press.

Quilici, A., Dyer, M. G., and Flowers, M. (1988). Recognizing and Responding to Plan-Oriented Misconceptions. *Computational Linguistics*, 14(3).

Rich, C. and Sidner, C. L. (1998). COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*, 8(3-4).

Tate, A. (1977). Generating Project Networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*.

van Beek, P., Cohen, R., and Schmidt, K. (1993). From Plan Critiquing to Clarification Dialogue for Cooperative Response Generation. *Computational Intelligence*, 9(2).