Integrating Conversational Case Retrieval with Generative Planning

Héctor Muñoz-Avila[†‡], David W. Aha[‡],
Leonard A. Breslow[‡], & Dana S. Nau[†], Rosina Weber[⊤]

† Department of Computer Science, University of Maryland
College Park, MD 20742-3255 *lastname*@cs.umd.edu
‡ Navy Center for Applied Research in AI, Naval Research Laboratory (Code 5510)
Washington, DC 20375 *lastname*@aic.nrl.navy.mil
⊤Department of Computer Science, University of Wyoming, Laramie, WY 82071

Abstract. Some problem-solving tasks are amenable to integrated case retrieval and generative planning techniques. This is certainly true for some decision support tasks, in which a user controls the problem-solving process but cannot provide a complete domain theory. Unfortunately, existing integrations are either non-interactive or require a complete domain theory and/or complete world state to produce acceptable plans, preventing them from being easily used in these situations. We describe a novel integrated algorithm, named SiN, that is interactive and does *not* require a complete domain theory or complete world state. SiN users leverage a conversational case retriever to focus both partial world state acquisition and plan generation. We highlight the benefits of SiN (e.g., quadratically fewer cases needed) in an experimental study using a new travel planning domain.
Key words: Conversational case retrieval, planning, integrations

1 Introduction

Mixed-initiative planning is an interactive plan generation process in which two or more independent planners jointly solve a problem. At any time, one of them is controlling the plan generation process. Control is passed either when the planner p in control *cedes* it to another planner p', or through p''s *interruption*, which typically occurs when a human planner interrupts a generative planner.

Many systems support mixed-initiative planning, but they are either automated, require a complete domain theory, and/or require a complete world state. This prevents them from being used for (interactive) decision support tasks in which these are unavailable, which characterizes the focal tasks of our current projects (e.g., planning for non-combatant evacuations (NEOs)). We introduce a mixed-initiative plan generation algorithm, named SiN, that integrates a generative planner (SHOP) (Nau et al., 1999) with a conversational case retriever (NaCoDAE/HTN), an extension of NaCoDAE (Aha & Breslow, 1997), to produce plans in decision support contexts. SiN can be characterized as follows:

– It employs a *unified object representation* (i.e., NaCoDAE/HTN's cases and SHOP's methods).
– It uses a *mixed-initiative plan generation process*. In SiN, SHOP cedes control to NaCoDAE/HTN whenever none of SHOP's methods or operators can be applied, while NaCoDAE/HTN cedes control to SHOP whenever the user completes a conversation.
– It can *generate plans given incomplete domain theories*, which prevents the application of typical generative planners. We will show how SiN's problem-solving experiences (i.e., NaCoDAE/HTN's cases) complement generalized domain knowledge (i.e., SHOP's methods and operators).
– It can *plan with an incomplete world state*. SiN uses NaCoDAE/HTN to interactively perform information gathering activities for SHOP.

After explaining how SiN differs from other planners in Section 2, we detail its hierarchical task network (HTN) representation in Section 3. Section 4 then describes the two modules and details their integration in SiN. We then describe an experimental study in Section 5 to highlight SiN's benefits.

2 Contributions in comparison with other planners

The key distinguishing feature of SiN (SHOP interleaved with NaCoDAE/HTN) is its interleaved control structure for plan generation, which is highlighted in Table 1 versus example planning systems in seven categories. SHOP, a generative planner, requires a complete domain theory. CHEF (Hammond, 1989) and DIAL (Leake et. al., 1997) are case-based, but does not exploit a generative component, and thus requires a large case base to perform well across a wide variety of problems. Prodigy/Analogy (Veloso & Carbonell, 1993) integrates generative and case-based planning, but requires a complete domain theory. Similarly, Paris (Bergmann & Wilke, 1995) integrates these two approaches, but is also not interactive.

SIPE II (Wilkins, 1998) is a mixed-initiative generative planner, but, while it can acquire world state information, it requires a complete domain theory. NaCoDAE/HTN (Muñoz-Avila et al., 1999) is a mixed-initiative case-based planner, but, like CHEF, it does not support generative planning. Mitchell's (1997) architecture, which uses cases to select which task to perform in a given tactical situation, is a more advanced example in this category. Similarly, the CHARADE and CARICA systems (Avesani et al., 1998) interactively acquire state information (i.e., for situation assessment), which is used to retrieve and adapt planning cases that are then given to a resource allocation scheduler. However, they do not support generative planning.

MI-CBP (Veloso et al., 1997), which extends Prodigy/Analogy, uses a control structure where interaction is limited to providing the system with user feedback on completed plans. This requires MI-CBP to input, or learn thru feedback, a sufficiently complete domain theory to solve problems. In contrast, SiN gathers information it requires from the user through NaCoDAE/HTN conversations,

Table 1. Contrasting SiN's characteristics with other planning systems.

System	Generative	Case-based	Mixed-Initiative	Interleaved
SHOP	√			
CHEF		√		
Prodigy/Analogy	√	√		
SIPE II	√		√	
NaCoDAE/HTN		√	√	
MI-CBP	√	√	√	
SiN	√	√	√	√

but does not learn from user feedback. CAPlan/CbC (Muñoz-Avila et al., 1997) is another integrated, interactive planner, but its interaction does not include acquiring world state information.

Finally, SiN's interleaved control strategy allows both the case-based and generative planning modules to contribute task decompositions during planning. Because the SiN user supplies world state information incrementally thru its interaction with NaCoDAE/HTN, as needed, not all of the world state is needed a priori to generate plans.

Perhaps the most closely related architecture to SiN is the one described by Carrick et al. (1999), which uses pre-stored hierarchical plans to perform information gathering activities for a conversational case retriever to solve interactive diagnosis tasks. SiN instead uses a conversational case retriever to gather information, and provide task decompositions, for a generative planner. Integrating SiN with their approach would yield a powerful interactive planner, especially if we do not require the information-gathering plans to be pre-constructed. This is an interesting avenue for future work.

SiN is a subset of an extended HICAP (Muñoz-Avila et al., 1999). We summarize their relationship in Section 6. Figure 1 shows a snapshot of HICAP. It displays a plan for trip from Greenbelt to downtown New York City (NYC). The left side shows a hierarchy of *tasks* and the right side a hierarchy of *resources*. For the rest of this paper we will concentrate on the generation of task hierarchies.

3 Hierarchical task network plans

This paper concerns a mixed-initiative elicitation process for generating task hierarchies. A *task hierarchy*, is a triple $(T, <, \wedge)$, where T is a set of *tasks*. The relation $<$ is an *ordering relation* between elements in T. We write $t1 < t2$ if $t2$ should be executed after $t1$. In Figure 1, the $<$ relations are represented by the arrows. For example, the arrow from the task *Travel-segment from Greenbelt to Greenbelt-Metro* to the task *Travel-segment from Greenbelt-Metro to Union Station* indicates that *Travel-segment from Greenbelt to Greenbelt-Metro* $<$ *Travel-segment from Greenbelt-Metro to Union Station* holds. The notation $\wedge$ denotes a hierarchical relation, where $t1 \wedge t2$ indicates that $t2$ is a subtask of $t1$. Hierarchical task relations are represented by indentations in Figure 1. For example,

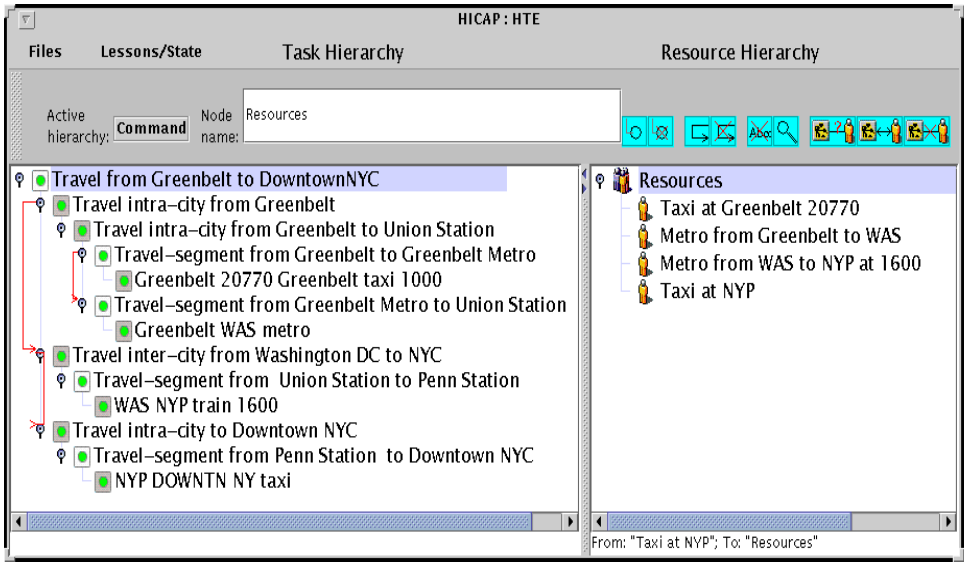

Fig. 1. A snapshot of HICAP.

the task *Travel inter-city from Washington DC to NYC* is a subtask of *Travel from Greenbelt to Downtown NYC*. The task hierarchy is displayed as a tree (see Figure 1) with a root indicating the top-level task, called the *initial task*. Tasks that are not decomposable are called *primitive*. The others are called *compound*.

Given a task hierarchy, The ordering ($<$) and hierarchical ($\wedge$) relations induce a hierarchical ordering relation ($<_\wedge$) as follows: given two tasks $t1$ and $t2$, the ordering relation $t1 <_\wedge t2$ holds iff:

- $t1 < t2$ holds, or
- there are ancestors $t1'$ of $t1$ in T and $t2'$ of $t2$ in T, w.r.t. $\wedge$, such that $t1' < t2'$, $t1 < t2'$, or $t1' < t2$ holds.

In Figure 1, *Travel intra-city from Greenbelt* $<_\wedge$ *Travel-segment from Union Station to Penn Station* holds because *Travel inter-city from Washington DC to NYC* is an ancestor of *Travel-segment from Union Station to Penn Station* and *Travel intra-city from Greenbelt* $<$ *Travel inter-city from Washington DC to NYC* holds.

Given a task hierarchy $(T, <, \wedge)$, its *concrete plan*, or simply *plan*, is a pair $(T_{action}, <_\wedge)$, where T_{action} is the set of all primitive tasks in T. For example, the concrete plan associated with the task hierarchy shown in Figure 1 consists of four steps:

1. travel by taxi from Greenbelt to the Greenbelt Metro Station,
2. take the Metro to Union Station,
3. take the train to Penn Station in NYC, and

4. take a taxi from Penn Station to downtown NYC.

Hierarchical task network (HTN) planning proceeds by decomposing a given task into simpler tasks. A *domain theory* indicates how tasks are to be decomposed; we will detail this in Section 4. A *state* is a set of ground atoms that denote known true facts. A *planning problem* is a triple (t, S, D), where t is as task, S is a state and D is a domain theory. t becomes the initial task of the resulting task hierarchy.

4 Integrating case retrieval with plan generation

This section introduces a novel approach for interleaving case retrieval and plan generation in a decision support context. The first two subsections describe the individual modules, while the third describes their integration.

4.1 SHOP: A simple hierarchical ordered planner

SHOP (Nau et al., 1999) is a domain-independent HTN planner that plans for tasks in the order that they will be executed. SHOP has been shown to be several orders of magnitude faster than some well-known AI planning systems.

SHOP applies a domain theory, consisting of methods and operators, to generate plans. A *method* is an expression of the form $M = (h, P, ST)$, where h, (the method's *head*) is a compound task, P is a set of *preconditions*, and ST is the set of M's (children) *subtasks*. M is *applicable* to a task t, relative to a state S, iff matches(h, t, S) (i.e., h and t have the same predicate and arity, and a consistent set of bindings Θ exists that maps variables to values such that all terms in h match their corresponding ground terms in t) and the preconditions P are *satisfied* in S (i.e., there exists a consistent extension of Θ, named Θ', such that $\forall p{\in}P\{p\Theta'{\in}S\}$), in which case $M(t, S) = ST\Theta'$.

Figure 2(a) displays an example method that defines how to move between two locations. The preconditions for applying this method indicate that the cities in these locations must have train stations. The subtasks are to travel to the train station, travel between the two train stations, and then travel to the final location. (SHOP always assume a total order between the subtasks; the subtasks are planned in the order that they are listed in the method.) This was the method applied to the compound task *travel from Greenbelt to Downtown NYC* in Figure 1. This task is represented internally as *travel(Greenbelt,DowntownNYC)*. The variable bindings for the parent task of this method were $\{\ ?locFrom \rightarrow Greenbelt, ?locTo \rightarrow DowntownNYC\}$.

An *operator* is an expression of the form $O = (h, A, D)$, where h (the operator's *head*) is a primitive task, and A and D are the so-called *add*- and *delete*-lists. These lists define how the operator's application transforms the current state S: every element in the add-list is added to S and every element in the delete-list is removed from S. An operator O is *applicable* to a task t, relative to a state S, iff matches(h, t, S).

Head:
$travel(?locFrom, ?locTo)$

Preconditions:

$in(?locFrom, ?city1)$
$in(?locTo, ?city2)$
$trainStation(?station1, ?city1)$
$trainStation(?station2, ?city2)$

Subtasks:

$travelC(?locFrom, ?station1)$
$travelIC(?station1, ?station2)$
$travelC(?station2, ?locTo)$

Head:
$travelC(PennSt, DowntownNYC)$

Questions:

Is it raining hard in NYC? : No
Is the traveler carrying luggage? : Yes

Subtasks:

$take(taxi, PennSt, DowntownNYC)$

Fig. 2. (a) Example of a method in the personal travel domain. Terms beginning with a question mark denote variables. (b) Case from the travel domain.

SHOP Planning Algorithm. At any point during the planning process, SHOP is decomposing a task hierarchy $(T, <, \wedge)$ relative to a state S. The initial task is the focus of the current problem (t, S, D). Leaves in the task hierarchy are either decomposable or not decomposable (i.e., SHOP has no decomposition methods or applicable operators for them). SHOP plans, using preorder traversal on the leaf tasks, as follows:

1. If it finds a leaf task t that is primitive and has an applicable operator O, then O is applied to t, and S is updated as needed.
2. Else if t is compound and an applicable method M exists, then M is applied, which expands t with M's subtasks.
3. Else if all of the leaves in the task hierarchy are primitive, this process terminates with a success.
4. Else, SHOP backtracks.

4.2 NaCoDAE/HTN: A conversational case retriever

NaCoDAE/HTN is an extension of the NaCoDAE conversational case retriever (Aha & Breslow, 1997) that can be used to suggest task decompositions. A *case* is an expression of the form $C = (h, QA, ST)$, where h is the case's head (i.e., a task), QA is a list of $\langle$question,answer$\rangle$ pairs, and ST is a set of subtasks.

Users interact with NaCoDAE/HTN in *conversations*, which begin when the user selects a task t. NaCoDAE/HTN responds by displaying the top-ranked cases whose heads are equal to t. Cases are ranked according to their *similarity* to the current state S, which is the state that exists at that time during the conversation. Similarity is computed by comparing the contents of the current state with a case's question-answer pairs. (That is, each question-answer pair is represented as a monadic predicate in S, and similarity for a given $\langle$question,answer$\rangle$ pair becomes a membership test in S.) NaCoDAE/HTN also displays questions, whose answers are not known in S, ranked according to their frequency among the top-ranked cases. The user can select and answer (with a) any displayed question q, which inserts $q(a)$ into S. This state change subsequently modifies the case and question rankings. A conversation ends when the user selects a case C, at which time t is decomposed into ST (i.e., C's subtasks).

Figure 2(b) displays an example case from the personal travel domain. A person traveled by taxi between Penn Station and downtown because it was not raining and the traveler has luggage. In Figure 1, this was the case applied to decompose the task *Travel-segment from Penn Station to Downtown NYC* (internally represented as *travelC(PennSt,DowntownNYC)*) into *NYP DOWNTN NY taxi* (internally represented as *take(taxi,PennSt,DowntownNYC)*).

NaCoDAE/HTN Case-Based Planning Algorithm. As with the SHOP generative planner, NaCoDAE/HTN refines task hierarchies. When given a task t to refine, NaCoDAE/HTN uses t as an index for initial case retrieval and conducts an interactive conversation, which ends when the user selects a case C, which is then used to decompose t into ST (i.e., C's subtasks).

4.3 The SiN integration

SHOP's methods and NaCoDAE's HTN cases are similar in that they indicate how to decompose a task into subtasks. Their main difference is that a method's preconditions must always be satisfied in the current state, which need not be true for a case's $\langle$q,a$\rangle$ pairs (i.e., cases support partial matching). Also, a user can select and apply a case's task decomposition even when some of its questions have not been answered. The decision to apply a case is the user's responsibility.

This difference can be illustrated with the example method and case shown respectively in Figures 2(a) and 2(b). In the method, one can only use a train if there is a train station in both locations. However, the case can be applied (i.e., use the taxi) even if it is sunny or the traveler is not carrying substantial luggage.

A single (current) state S is maintained in SiN that is accessible to and updatable by both SHOP and NaCoDAE/HTN. Answers given by the user during an interaction with NaCoDAE/HTN are added to S (i.e., each question has a translation into a first-order predicate). Changes that occur by applying SHOP's operators are also reflected in S.

Both SHOP and NaCoDAE/HTN assist SiN with refining task hierarchies. At any point of time, one of these is in control and is focusing on a compound, leaf task t to decompose. SiN proceeds as follows:

1. If SHOP is in control and can decompose t, it does so. If SHOP cannot decompose t, and if t can be decomposed by NaCoDAE/HTN, SHOP cedes control to NaCoDAE/HTN.

2. If NaCoDAE/HTN is in control and can decompose t, it does so. If NaCoDAE/HTN cannot decompose t, and t can be decomposed by SHOP, NaCoDAE/HTN cedes control to SHOP.
3. If neither SHOP nor NaCoDAE/HTN can decompose t, SHOP will backtrack, if possible. If backtracking is impossible (for example, because t is the initial task), this planning process is interrupted and a failure is returned.

By continuing in this way, and assuming that the process is not interrupted with a failure, SiN will eventually yield a task hierarchy in which all leaves are primitive tasks. Typically, this final task hierarchy contains decompositions generated by SHOP and others generated by NaCoDAE/HTN. The task hierarchy depicted in Figure 1 illustrates this situation; all interior nodes (i.e., nodes that are neither the root nor leaves) are generated by SHOP whereas all leaves are generated by NaCoDAE/HTN.

5 Evaluation

As explained in Section 2, planning algorithms comparable to SiN require a complete domain theory, a complete world state, or, as explained in Section 5.2, a much larger knowledge base. Therefore, empirical comparisons with these other planning algorithms would not be particularly revealing; SiN is in a different category of planners.

Therefore, for this evaluation, we focused on our claim that SiN's capabilities extend those of its component algorithms. Our arguments rest on brief theoretical and empirical analyses. Section 5.1 focuses on the simpler argument: comparing SiN with SHOP. Section 5.2 then compares SiN with NaCoDAE/HTN.

5.1 Comparing SiN with SHOP

Because SHOP is a generative planner, it requires a complete world state and domain theory to solve a given planning problem. Otherwise, if one of its methods requires world state information for testing its preconditions, and that information is unavailable, then it cannot be applied, and SHOP will fail to generate a plan. Furthermore, if it is not given a complete domain theory, then it may again fail when it is unable to generate a decomposition for a given task.

In contrast, SiN does not require a complete world state and domain theory; it uses NaCoDAE/HTN to incrementally collect world state information, and to provide decomposition suggestions for tasks that SHOP cannot decompose (i.e., because it lacks methods for those tasks). In regard to the world state, SiN uses NaCoDAE/HTN to elicit, from the user, only the world state information it needs to generate a certain plan. In comparison, because the user doesn't know in advance what information is necessary for SHOP to infer a plan, they must encode more information about the world state that is required by SHOP to generate a plan.

Table 2. Questions used to define personal travel states.

Question	Possible Answers
Is it raining hard?	Y/N
Is there any major accident or unusual event?	Y/N
What is the chance of large snow accumulation?	High, moderate, low
How many passengers?	1, 2, 3, 3+
Is there any luggage to check?	Y/N
Are any children or seniors involved?	Y/N
Is travel occuring on a holiday? Which?	Holiday name, or No
What is the starting day of the week for this trip?	7 distinct answers

5.2 Comparing SiN with NaCoDAE/HTN

SiN's advantage vs. using only NaCoDAE/HTN is that, for domains requiring different plans for different world states, NaCoDAE/HTN will need significantly more cases to generate the same space of plans as SiN. It is important for NaCoDAE/HTN to generate the same space of plans as SiN because plan quality varies greatly with the selected dependent measure(s). We first provide empirical evidence for this claim, and then focus on a theoretical justification.

The purpose of our experimental study was to investigate how the quality of the generated plans vary. Towards this goal, we developed a knowledge base and plan evaluator for a planning domain where the contents of the world state can significantly impact choices during planning. Named the *personal travel domain*, its plans concern traveling from locations in Washington, DC to downtown New York City (NYC). By encoding 7 transportation methods (3 inter- and 4 intra-city) and defining plans to have 3-5 segments, approximately 756 plans could be generated. We defined states using the eight questions shown in Table 2. The knowledge base consists of 10 methods and 1 operator for SHOP and 40 cases for NaCoDAE/HTN. Each plan's task hierarchy was obtained by applying 4-6 methods, 3-5 operators (i.e., the same one repeatedly), and 3-5 cases.

Our personal travel plan evaluator can generate a different time duration each time it is given a plan and world state because of its nondeterministic execution. For each run, it outputs whether the plan succeeded and, if so, the trip's duration. A plan fails when segment delays cause a late arrival for a segment requiring a fixed time departure (e.g., an airplane flight). For each segment, we applied a delay function that is influenced by world state conditions. For example, a flight segment will incur a longer delay for higher chances of large snow accumulation, especially on holidays (i.e., high travel days). Segments are categorized into short, medium, and long lengths, and delays can range from 0 up to 4.5 times a segment's anticipated duration; smaller multiples are used for maximum delays for medium (3.5) and long (2.5) duration segments.

We selected ten goals, corresponding to ten pairs, each consisting of a departure location in Washington, DC and downtown NYC. For each goal, we generated 10 random world states (i.e., providing answers to all 8 questions), thus yielding 100 total planning problems. SiN was then used to generate a plan

Table 3. Comparative results on the travel domain.

Algorithm	Success Rate	Mean Duration
100 plans	70%	9h45min (3h14min)
Plans with highest success rates	92%	13h35min (3h13min)
Plans with lowest success rates	45%	7h40 (3h 7min)

for each problem, and each was executed 10 times by the plan evaluator. The first row of Table 3 displays the results.

We examined the output of the evaluator obtained for the 100 plans generated. For each goal g, we selected two plans; one for which the success rate was the highest and another for which the success rate was the lowest. For each of the 10 goals, we selected the plans with highest and lowest success rates and re-ran the evaluator, again using the 10 random world states used previously. The results are displayed in rows 2 and 3 of Table 3, along with parenthesized standard deviations. We observed a correlation between the success rate and the duration. For the plans with highest success rate, the duration increases by almost 4 hours, or 39% over the mean computed over the 100 plans. For the plans with the lowest success rate, the duration decreases by approximately 2 hours, or 22% relative to the mean.

These results show that there is a large variation in the quality of the plans that potentially could be generated by SiN in terms of the success rate and the duration. Because it is an interactive algorithm, SiN can generate plans with widely different qualities according to an *individual* user's needs by using NaCoDAE/HTN. Some users may want to infer plans that they think will be successful, others may want to infer plans with lower duration, and yet others may want plans with both lower duration and higher success rate. There are other measures of quality, including cost and waiting time (i.e., how long the traveler has to wait in the airport). We conclude from these experiments that, to provide a similar flexibility as in SiN, NaCoDAE/HTN must generate plans from the same space.

We can approximate the number of cases that NaCoDAE/HTN, working solo, would require to generate plans from the same space as SiN does by carefully analyzing each of SHOP's methods for this domain. These methods are used for all *interior* branches in the task hierarchy generated by SiN; cases, and the sole SHOP operator, are used for task decompositions only at the *bottom* branches (i.e., leading to leaves). Using the personal travel domain, Table 4 shows the number of methods and cases required by SiN and NaCoDAE/HTN, based on three domain characteristics. In summary, the number of methods required by SiN's SHOP module is constant, while the number of cases required by NaCoDAE/HTN to cover the same hierarchical decompositions as SHOP's methods increases quadratically with domain size. For example, with the current values of $L_s = 10$ and $L_d = 1$, the number of cases required by NaCoDAE/HTN to encode SHOP's operators is 165. But this rises to 1248 when $L_s = 31$ and $L_d = 4$

Table 4. The number of SHOP methods or NaCoDAE/HTN cases required for the personal travel domain, where L_s is the number of starting locations, L_d is the number of destination locations, and N is the number of possible plan segments.

Branch Type	**SiN**		**NaCoDAE/HTN**
	SHOP	NaCoDAE/HTN	
Interior	10	0	$O(L_s(L_s + L_d))$
Bottom	0	N	N

(i.e., the number of start and destination locations that are known by the plan evaluator).

In summary, SiN effectively combines abstract and experiential knowledge to plan, where the abstract knowledge (SHOP's methods) refers to generalizations in planning space. This is a far more compact representation than using only experiential knowledge (i.e., cases), and is reminiscent of (non-planning) systems that effectively combine rules and cases (e.g., Golding & Rosenbloom, 1991).

6 Summary and relation to HICAP

In this paper we introduced SiN, a mixed-initiative planner that interleaves case-based and generative planning modules. We argued that this structure provides SiN with benefits relative to its individual components; it relaxes its generative planner's need for a complete world state and domain theory, and it can significantly reduce the number of cases required by its conversational case retriever. Furthermore, SiN is applicable to decision support tasks, and we summarized its benefits vs. other integrated, case-based planners.

SiN is a compound module of HICAP, the plan authoring tool that previously did not contain a generative planning module (Muñoz-Avila et al., 1999). In addition to SiN's two modules, HICAP's other modules include the HTE interface, the DecTS conflict resolution manager, and an active delivery module for lessons learned (Weber et al., 2000). Future work on HICAP includes incorporating a learning apprentice for assisting with knowledge (i.e., cases, methods, lessons) acquisition, incorporating case-adaptation capabilities, and integrating it with a more sophisticated front-end (e.g., SOFTools) for incremental planning.

Acknowledgments

This research was supported by grants from the Office of Naval Research, the Naval Research Laboratory, and the Army Research Laboratory. Thanks also to our submission's reviewers.

References

Aha, D. W., & Breslow, L. A. (1997). Refining conversational case libraries. *Proceedings of the Second International Conference on CBR* (pp. 267–278). Providence, RI: Springer.

Avesani, P., Perini, A., & Ricci, F. (1998). The twofold integration of CBR in decision support systems. In D.W. Aha & J.J. Daniels (Eds.), *Case-Based Reasoning Integrations: Papers from the 1998 Workshop* (Technical Report WS-98-15). Menlo Park, CA: AAAI Press.

Bergmann R., & Wilke W. (1995). Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, *3*, 53–118.

Carrick, C., Yang, Q., Abi-Zeid, I., & Lamontagne, L. (1999). Activating CBR systems through autonomous information gathering. *Proceedings of the Third International Conference on Case-Based Reasoning* (pp. 74–88). Munich, Germany: Springer.

Golding, A., & Rosenbloom, P. S. (1991). Improving rule-based systems through case-based reasoning. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 22–27). Anaheim, CA: AAAI Press.

Hammond, K. (1989). *Case-based planning: Viewing planning as a memory task*. Boston, MA: Academic Press.

Leake, D., Kinley, A., & Wilson, D. (1997). A Case Study of Case-Based CBR. *Proceedings of the Second International Conference on Case-Based Reasoning* (pp. 371–382). Providence, RI: Springer.

Mitchell, S.W. (1997). A hybrid architecture for real-time mixed-initiative planning and control. *Proceedings of the Ninth Conference on Innovative Applications of AI* (pp. 1032–1037). Providence, RI: AAAI Press.

Muñoz-Avila, H., McFarlane, D., Aha, D.W., Ballas, J., Breslow, L.A., & Nau, D. (1999). *Using guidelines to constrain interactive case-based HTN planning. Proceedings of the Third International Conference on Case-Based Reasoning* (pp. 288–302). Munich, Germany: Springer.

Muñoz-Avila, H., Weberskirch, F., & Roth-Berghofer, T. (1997). On the relation between the context of a feature and the domain theory in case-based planning. *Proceedings of the Second International Conference on Case-Based Reasoning* (pp. 337–348). Providence, RI: Springer.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968–973). Stockholm: AAAI Press.

Veloso, M. M., & Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, *10*, 249–278.

Veloso, M., Mulvehill, A.M., & Cox, M.T. (1997). Rationale-supported mixed-initiative case-based planning. *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence* (pp. 1072–1077). Providence, RI: AAAI Press.

Weber, R., Aha, D.W., Muñoz-Avila, H., & Breslow, L.A. (2000). Active delivery for lessons learned systems. To appear in *Fifth European Workshop on Case-Based Reasoning*. Trento, Italy: Springer.

Wilkins, D.E. (1998). *Using the SIPE-2 planning system: A manual for Version 5.0* (Working Document). Menlo Park, CA: Stanford Research International, Artificial Intelligence Center.