

CBM-Gen+: An Algorithm for Reducing Case Base Inconsistencies in Hierarchical and Incomplete Domains

Ke Xu & Héctor Muñoz-Avila

Department of Computer Science and Engineering
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015, USA
{kex2,hem4}@lehigh.edu

Abstract. We propose an algorithm, CBM-Gen+, to refine case bases for hierarchical and incomplete domains. In these domains, the case bases are the main source of domain information because of the absence of a complete domain theory. CBM-Gen+ revises the existing cases when a new solution is captured. The main purpose of this revision is to reduce inconsistencies in the cases. We will prove that CBM-Gen+ is sound relative to the captured solutions. We also perform experiments showing that CBM-Gen+ is on average at least as efficient as a previous approach for constructing case bases for hierarchical domains.

1 Introduction

One of the main current research topics in CBR is case base maintenance (Leake & Wilson, 1998; Watson, 1997, Chapter 8). In recent years researchers have formulated case base maintenance (CBM) policies for reducing the size of the case base without losing coverage (Racine & Yang, 1997; Kitano & Shimazu, 1996; Smyth & Keane, 1998), for constructing compact case bases (Smyth & McKenna, 1999), for refining case libraries (Aha & Breslow, 1998), and for adding new cases (Ihrig & Kambhampati, 1996; Muñoz-Avila, 2001).

In this paper we present an approach for case base maintenance in hierarchical and incomplete domains. In hierarchical domains high-level tasks are decomposed into simpler tasks. Incomplete domains are those for which no complete domain theory exists explaining how to solve problems in that domain. An example of such a domain is project planning. Project Planning is a business process for successfully delivering one-of-a-kind products and services under real-world time and resource constraints. Project plans are hierarchical in nature being expressed in so-called work-breakdown structures. Work-breakdown structures closely resemble a hierarchical (HTN) plan representation (Muñoz-Avila et al, 2002). There is no complete domain theory for generating project plans. Knowledge about how to formulate project plans is mostly episodic, even though general guidelines have been formulated to help with the project plan elicitation (PMI, 1999).

In previous work we proposed an algorithm, Gen+, for acquiring cases automatically by capturing user sessions with project planning tools (Mukammalla & Muñoz-Avila, 2002). The main motivation for our case acquisition procedure was to

make case capture transparent to the user. Cases captured with Gen+ are generalizations of project planning episodes. The main advantage of this generalization is that it increases the opportunities for reusing the cases. In addition, cases contain pieces of project plans rather than complete project plans. Again, this results in greater flexibility since pieces of different projects can be combined to create new projects. However, Gen+ can over-generalize project planning episodes. That is, potential instantiations of the cases may not be valid in the target domain.

Because of the generalization procedure in Gen+, multiple cases may be applicable for a given problem. Thus, any of these cases can be reused. If a problem and a solution were captured to acquire a case, there is no guarantee that the case will be reused when the problem is given once again. Depending on the domain, reusing a different case may yield an incorrect solution. In such situations, we say that Gen+ is not sound relative to the captured solutions. This may be a surprising result but it is a trade-off issue between soundness and coverage. As pointed out in (Mukammalla & Muñoz-Avila, 2002), if cases are not generalized, a very large case base will be required to obtain an adequate coverage.

We will present a new procedure, CBM-Gen+, for refining cases having a hierarchical representation. Cases still are generalized solutions as in Gen+. However, CBM-Gen+ revises previously acquired cases when a new case is captured. This revision process is performed when potential conflicts between the new and the existing cases are identified. Thus, it is possible that the order in which the solutions are captured will determine which cases will be reused. We will show that CBM-Gen+ is sound relative to the captured solutions. That is, if S is a solution for a problem P and (P, S) is used to learn generalized cases in a case base, then S will be generated as a solution whenever P is given as a problem.

This property is a weak version of soundness for planning procedures. We cannot ensure that every solution generated is correct in the target domain unless we make some stronger assumptions (e.g., (Muñoz-Avila et al, 2001)). Still, this property guarantees soundness for those problems whose solutions have been previously captured.

As we will see, existing cases may be split into two separate cases during the revision process in CBM-Gen+. As a result, the case bases generated with CBM-Gen+ are generally larger than the case bases with Gen+ even if they received the same input cases. Despite this, we will describe experiments showing that, in average, problem solving with CBM-Gen+ case bases is not less efficient than with Gen+ case bases. Furthermore, our experiments suggest that problem solving with CBM-Gen+ is much faster if the input problems are solvable.

2 The Gen+ Procedure

In (Mukammalla & Muñoz-Avila, 2002), a procedure, Gen+, is presented to acquire cases representing pieces of a project plan. The main motivation of Gen+ is to assist users in the development project plans by using hierarchical case decomposition techniques (Muñoz-Avila et al, 2002). Starting point for this approach is the recognition that the representations commonly used for project plans are very similar

to the hierarchical task network (HTN) representations. Thus, HTN planning techniques can generate project plans.

2.1 Cases Contents

In HTN planning, high-level tasks are decomposed into simpler tasks. This decomposition process continues until so-called primitive tasks are obtained. Primitive tasks indicate concrete actions to be undertaken. HTN planning techniques requires the availability of a domain theory capturing all possible decompositions in the target domain. Such a requirement is unfeasible in project planning where users follow general guidelines and their own experience rather than a well-determined collection of rules. For this reason, our KBPP approach advocates capturing episodic project planning episodes as cases that have an HTN-like representation.

Cases have the form $(:case\ h\ cond\ ST)$, where h is the task being decomposed, $cond$ is a collection of conditions, ST is a collection of subtasks. Given a task h' , if h matches h' with a substitution β (i.e., $h\beta = h'$) and the instantiated conditions $cond\beta$ are satisfied in the current situation with a substitution α , then the case can be used to decomposed h' into $ST\beta\alpha$.

2.2 Case Capture

Given a case $C = (:case\ h\ cond\ ST)$, the generalization of C , denoted by $Gen+(C)$ is obtained by considering each parameter, ψ , occurring in the arguments of the head, conditions, or subtasks of C (Mukkamalla & Muñoz-Avila, 2002). Two situations may occur: first, if the type of ψ is known, then all occurrences of ψ in h , $cond$, and ST are replaced with a unique variable, $?\psi$. If $tName$ denote the type of ψ , then the condition $(tName\ ?\psi)$ is added to $cond$. This condition indicates that any instantiation of $?\psi$ must be of type $tName$. Second, parameters for which the type is not known are not generalized. In addition the following constraints are added as conditions of the case:

- For each two different variables, $?x$ and $?y$, of the same type, the constraint $(:different\ ?x\ ?y)$ is added
- For each constant, c , and each variable, $?x$, the constraint $(:different\ ?x\ c)$ is added

The condition $(tName\ ?\psi)$ is added to reduce the chances of over-generalization. Although over-generalization may still occur, the alternative (e.g., not generalizing the cases) will require too many cases to get adequate coverage. If there is a maximum of n task names, with an average number of m arguments and each argument can take an average number of v values, there are nm^v different tasks. We require at least one case for each task. Thus, we would require at least nm^v cases. Notice that this is the minimum number of cases required. But we may need more cases to obtain a better coverage because similarity is computed not only based on the tasks but also on the current situation (e.g., available resources).

As an example, suppose that the equipment equip103 must be delivered between two specific locations. Table 1 shows a concrete solution for this task, namely, contracting the delivering company Deliver Incorporated (first column). Gen+ stores this solution by adding variables with their corresponding types (second column).

Table 1. A concrete solution and the corresponding Gen+ case

Concrete Solution	Gen+ Case 1
Task: (deliver equip103 dept107 office309) Condition (resources): Deliver_inc equip103 dept107 office309 Subtask: (contract Deliver_inc equip103 dept107 office309)	Task: (deliver ?e ?d ?o) Condition: (deliverCompany ?fd) (equipment ?e) (depot ?d) (office ?o) Subtask: (contract ?fd ?e ?d?o)

3 Soundness Relative to the Captured Solutions

One of the most challenging aspects of working with domains such as project planning is the lack of a domain theory. The Gen+ procedure fills this gap by proposing generalizations of the solutions it receives as inputs. Depending on the domain, these generalizations may be correct. However, it is easy to construct examples where instantiations of the case do not model the target domain correctly. Furthermore, as more cases are added to the case base, problem-solution pairs used as input to generate cases may not be necessarily reconstructed. This motivates the following definition:

Definition: Suppose that a collection of pairs (P_i, S_i) is given, where P_i is a problem and S_i is a solution for P_i in a target domain. Suppose that this collection is used as input for a procedure to construct a case base CB. The procedure is said to be **sound relative to the captured solutions** if for every problem P_i , the solutions generated using CB are correct in the target domain.

This definition presupposes a correct case reuse algorithm. In our work, we use the SiN reuse algorithm which is provable correct (Muñoz-Avila et al, 2001).

We can now show that Gen+ is not sound under this definition: continuing with the example shown in Table 1, suppose now that a new solution and the corresponding generalized case shown in Table 2 are added to the case base. The main difference between the solution in Table 1 and the new solution is that the equipment is delivered in two stages. First, Deliver Incorporated is once again contracted to deliver the equipment to depot55. Second, one of the company's own truck is used to make the final delivery to office700. The generalized case is also shown in Table 2. Suppose that the reason why this delivery must be performed in two stages is because there is no company that delivers goods from dept107 to office700. Lets assume that

the case base consists of both Gen+ cases and that the task and the conditions in the first column of Table 2 are given once again as a new problem. With this assumption, it is obvious that Gen+ Case 2 is applicable because it is the generalization of the concrete solution. Gen+ Case 1 is also applicable because its conditions are a subset of the conditions of Gen+ Case 2. However, under our assumption selecting Gen+ Case 1 is incorrect. Thus, Gen+ is not sound relative to the captured solutions.

Table 2. Second concrete solution and the corresponding Gen+ case

Concrete Solution	Gen+ Case 2
Task: (deliver equip103 dept107 office700) Condition (resources): Deliver_inc Unlimited_Deliver equip33 dept107 office70 depot55 truck654 Subtask: (contract Deliver_inc equip33 dept107 Depot55) (internalDeliver truck654 equip33 depot55 office70)	Task: (deliver ?e ?d ?o) Condition: (deliverCompany ?fd) (deliverCompany ?ud) (equipment ?e) (depot ?d) (office ?o) (depot ?d1) (truck ?t) (:different ?d ?d1) Subtask: (contract ?fd ?e ?d1 ?o) (internalDeliver ?t ?e ?d1 ?o)

4 The CBM-Gen+ Procedure

CBM-Gen+ and Gen+ both capture problem-solution pairs and generalize them to construct case bases. The difference is that Gen+ simply adds generalized cases into the case base without making any revisions. On the other hand, CBM-Gen+ may refine existing cases and the new case before adding it into the case base. *Unless stated otherwise, in the remaining of this paper, we assume that the all the cases have been generalized.* As before, a case (:case *h cond ST*) consists of three parts: the task *h*, the conditions *cond*, and the subtasks *ST*.

4.1 Preliminaries

We introduce some definitions before detailing the CBM-Gen+ algorithm:

- *Notation:* Given a case $C = (:case\ h\ cond\ ST)$, $T(C)$ denotes the task *h*, $P(C)$ represents the conditions *cond* and $ST(C)$ represents the subtasks *ST*.
- *Task-Match:* two cases C and C' task-match, if there is a substitution θ such that: $T(C)\theta = T(C')$.

- *Condition Intersection Set*: If two cases C and C' task-match with a substitution θ . $P(C)\theta$ denotes the instantiated conditions in C . The intersection set of $P(C)\theta$ and $P(C')$ is called the condition intersection set of C and C' , which contains every condition appearing in both $P(C)\theta$ and $P(C')$. The condition intersection set of cases C and C' is denoted as $P(C \cap C')$.
- *Condition Difference Set*: Given two sets of conditions $P(C)$ and $P(C')$, $P(C) - P(C')$ denotes the set difference of $P(C)$ and $P(C')$.
- *Parent case* and *Child Case*: For two cases C and C' , if $ST(C)$ contains only one subtask, i.e., $ST(C) = \{st\}$, and $T(C') = st$, then C is called the parent case of C' . C' is called a child case of C .
- *Leaf Case*: A case C is called a leaf case if C has a parent case but no child cases in the case base.
- *Root Case*: A case C is called a root case if C has no parent case.

As with Gen+, the cases in CBM-Gen+ are stored in a plain list. However, in CBM-Gen+, there is a logical connection as illustrated in Figure 1. If the only subtask of C is equal to the head of $C1$ and $C2$ (i.e., $ST(C) = \{T(C1)\} = \{T(C2)\}$), C can be seen as the parent case of $C1$ and $C2$ ($C1$ and $C2$ can be seen as the child cases of C).

The case base may contain several root cases. In addition, CBM-Gen+ will guarantee that no child case will have more than one parent case, and each parent case has exactly two child cases.

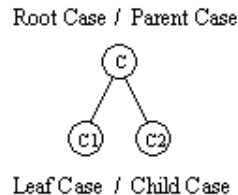


Fig. 1. Example illustrating the relations between cases

4.2 The Main Algorithm

The CBM-Gen+ algorithm receives as input a new case N . The algorithm first finds a root case C that tasks -matches with N and calls the auxiliary procedure $Insert(N,C)$. If the case base is empty or N does not task-match a root case then N is inserted as a new root case in the case base and the procedure ends. The procedure CBM-Gen+ is shown below:

```

Procedure CBM-Gen+(N) {
  IF CB is empty or N does not task-match with any root case THEN
    Add N into CB as a root case;
  IF CB is not empty, THEN {
    IF N task-match a root case C in CB THEN {
      Insert(N, C) } }
}

```

We are going to define now the auxiliary procedure $\text{Insert}(N, C)$. The parameter N represents the new generalized case to be added into the case base, and C represents an existing case. When $\text{Insert}(N, C)$ is called from the CBM-Gen+ procedure, C is a root case. But the procedure may be called recursively with child cases. We identify three main situations. The first situation occurs when $P(N \cap C)$ is a strict subset of $P(N)$ and $P(C)$. The pseudo-code for this situation is shown below:

```

IF  $P(N \cap C)$  is a strict subset of  $P(N)$  and  $P(C)$  THEN {
  Add the following cases in CB:
   $N \cap C = (: \text{case } T(C) \ P(N \cap C) \ \{vT\})$ 
   $C - (N \cap C) = (: \text{case } vT \ P(C) - P(N \cap C) \ ST(C))$ 
   $N - (N \cap C) = (: \text{case } vT \ P(N) - P(N \cap C) \ ST(N))$ 
  Remove  $C$  from CB }

```

The task vT is a new task not mentioned anywhere in the case base. This situation is illustrated in Figure 2. The new root case will have as preconditions the intersection $P(N \cap C)$. The left child case of the new root case will have as preconditions: $P(C) - P(N \cap C)$. The right child case will have as conditions $P(N) - P(N \cap C)$. Notice that the original case C can be reconstructed by taking the task of the new root case, collecting the conditions of the left child case and of the new root case, and taking the subtasks of the left case. The new case N can be reconstructed in a similar fashion.

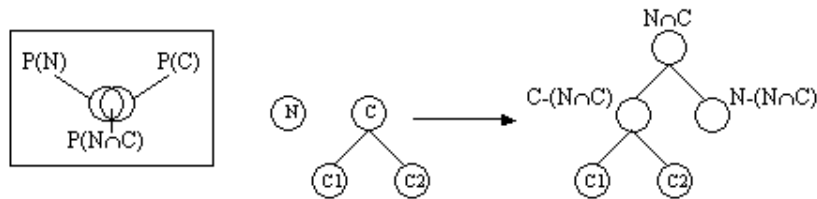


Fig. 2. $P(N \cap C)$ is a strict subset of $P(N)$ and $P(C)$

The second situation occurs when $P(N)$ is a strict subset of $P(C)$. The pseudo-code for this situation is shown below:

```

IF  $P(N)$  is a strict subset of  $P(C)$  THEN {
  Add the following cases in CB:
   $N \cap C = (: \text{case } T(C) \ P(N) \ (vT))$ 
   $C - N \cap C = (: \text{case } vT \ P(C) - P(N), \ ST(C))$ 
   $N - N \cap C = (: \text{case } vT, \ \neg(P(C) - P(N)) \ ST(N))$ 
  Remove  $C$  from CB }

```

As before νT is a new task name. If $cond$ is a collection of conditions, $\neg cond$ is the disjunction of the negation of each condition in $cond$. This situation is illustrated in Figure 3. The new root case has $P(N)$ as conditions. The left child case has $P(C)-P(N)$ as conditions and the right child case has $\neg(P(C)-P(N))$ as conditions. As before, the case C can be reconstructed by taking the task of the new root case, collecting the conditions of the left child case and of the new root case, and taking the subtasks of the left case.

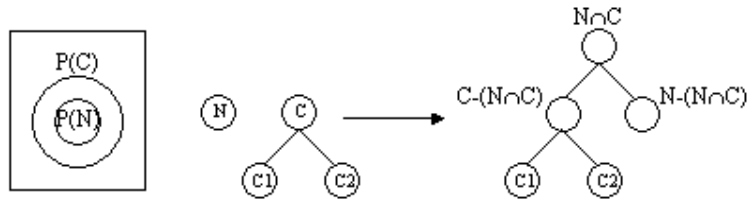


Fig. 3. $P(N)$ is a strict subset of $P(C)$

The third situation occurs when $P(C)$ is a strict subset of $P(N)$. The pseudo-code for this situation is shown below:

```

IF  $P(C)$  is a strict subset of  $P(N)$  THEN {
  IF  $C$  is a leaf case THEN {
    Add the following cases in CB:
       $N \cap C = (:case T(C) P(C) \{ \nu T \})$ 
       $C - (N \cap C) = (:case \nu T \neg(P(N) - P(C)) ST(C))$ 
       $N - (N \cap C) = (:case \nu T P(N) - P(C) ST(N))$ 
    Remove  $C$  from CB
  } ELSE {
    Let  $C1$  and  $C2$  be the child cases of  $C$  and  $ST(C) = \{ \nu T \}$ 
    Let  $N' = (:case \nu T P(N) - P(N \cap C) ST(N))$ 
    IF  $|P(N') \cap P(C1)| \geq |P(N') \cap P(C2)|$  THEN
      Insert( $N'$ ,  $C1$ )
    ELSE Insert( $N'$ ,  $C2$ ) } }

```

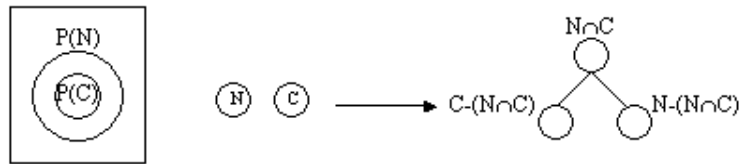


Fig. 4. $P(C)$ is a strict subset of $P(N)$

As before, νT is a new task name. There are two possibilities. If C is a leaf case then we add $N \cap C$ as the new root, and $C - (N \cap C)$ and $N - (N \cap C)$ as the child cases (see Figure 4). The negation in $C - (N \cap C)$ ensures that either N or C are selected. The

situation in which C is not a leaf is illustrated in Figure 5. The algorithm will call recursively insert with the case C1 or C2 that has more common conditions with N'.

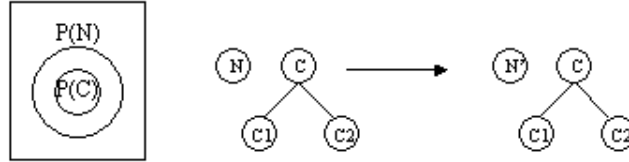


Fig. 5. P(C) is a strict subset of P(N)

Table 3. Refinement of Case 1 with CBM-Gen+

CBM-Gen+ Root Case	CBM-Gen+ Child Case 1	CBM-Gen+ Child Case 2
Task: (deliver ?e ?d ?o) Condition: (deliverCompany ?fd) (equipment ?e) (depot ?d) (office ?o) Subtask: (vT ?fd ?e ?d ?o)	Task: (vT ?fd ?e ?d ?o) Condition: (\neg (:different ?d ?d1) \vee \neg (deliverCompany ?ud) \vee \neg (depot ?d1) \vee \neg (truck ?t)) Subtask: (contract ?fd ?e ?d ?o)	Task: (vT ?fd ?e ?d ?o) Condition: (deliverCompany ?ud) (depot ?d1) (truck ?t) (:different ?d ?d1) Subtask: (contract ?fd ?e ?d1 ?o) (internalDeliver ?t ?e ?d1 ?o)

4.3 Example

We will illustrate the CBM-Gen+ procedure with the generalized cases, Case 1 and Case 2, of Tables 1 and 2. Suppose that initially Case1 is added as a new root in the case base. Suppose next that Case2 is added. Under these circumstances, we will be in the second situation of the insert algorithm. The reason for this is that the Case1 task-matches Case 2 and the conditions of Case 1 are a subset of the conditions of Case 2. The resulting case base is illustrated in Table 3. The task and the conditions of the root case, CBM-Gen+ Root Case, are the same as those in the original Case 1. The subtask of the root case is a new task named vT. The child cases of the root case, CBM-Gen+ Child Case 1 and CBM-Gen+ Child Case 2, contain the conditions $P(\text{Case 2})-P(\text{Case 1})$ and $\neg(P(\text{Case 2})-P(\text{Case 1}))$ respectively. Their subtasks contain the original subtasks from Case1 and Case2 respectively.

5 Soundness of CBM-Gen+

In Table 3, notice that whenever Child Case 2 is applicable then Child Case 1 is not applicable. Thus, if we give the same task and conditions as in the first column of Table 2, Child Case 2 will be applicable but Child Case 1 will not. This illustrates that

CBM-Gen+ is sound relative to the captured solutions. In this section we are going to sketch proof demonstrating that this is always the case.

Theorem: Let $\{(:case_1 h_1 cond_1 ST_1), (:case_2 h_2 cond_2 ST_2), \dots, (:case_n h_n cond_n ST_n)\}$ be a collection of concrete cases. If this collection is used to learn generalized cases using CBM-Gen+, then ST_i and only ST_i will be generated as a solution whenever $(h_i cond_i)$ is given as a problem.

Proof: (sketch) Without loss of generality, we will assume that if we generalize the input cases, they task-match. If this is not the situation we can simply split the input cases into separate collections such that generalized cases in the same collection task-match. Suppose that, for some i and j , the conditions $cond_i$ is a subset of $cond_j$. That is, $cond_i = \{p_1, p_2, \dots, p_m\}$, $cond_j = \{p_1, p_2, \dots, p_m, p_{m+1}, \dots, p_{m+k}\}$. If $(h_j cond_j)$ is given as a problem, then ST_j must be returned because the task and the conditions of the generalized case of $case_j$ match h_j and $cond_j$ respectively. The decomposition ST_i will not be returned because the conditions of the generalized case of $case_j$ do not match $cond_i$. The reason is that the generalization of $case_j$ with CBM-Gen+ will contain not only the generalization of the conditions $cond_i$ but also the disjunction of the negation of the conditions in the Condition Difference Set of $case_j$ and $case_i$ (i.e., $(\neg p_{m+1} \vee \dots \vee \neg p_{m+k})$), which cannot be satisfied in $cond_i$.

This proof is a sketch. As mentioned during the insertion algorithm, each generalized case may be split into several cases. Typically, for rebuilding the case, the path from a root case to a leaf case must be followed collecting all preconditions along the way. The details of this proof are too cumbersome and we omit these details for the sake of simplicity.

6 Empirical Evaluation

We performed an experimental evaluation of CBM-Gen+. The goal of these experiments was to see the impact of the overhead in CBM-Gen+ as a result of adding new cases when compared to Gen+.

6.1 Domains

We used the same domains that were created originally for demonstrating Gen+ (Mukkamalla & Muñoz-Avila, 2002). These domains are an HTN version of the Logistics domain (Veloso and Carbonell, 1993). In the Logistics domain, packages need to be moved between different places. Trucks and planes are used to transport objects. These experiments also used a subset of the UMTranslog (Erol et al, 1994). The subset of UMTranslog extends the logistics domain by defining objects that can be a combination of the various types such as *liquids* and *perishables*. For example, milk can be defined as a liquid and perishable object. The transportation means in the extension can be a combination of various types such as *tanker* and *refrigerated*. Each

type is specially suited to transport one object type. For example a refrigerated tanker truck is suitable to transport milk.

6.2 Experimental Setup

A random problem generator feeds problems to the HTN planner SHOP that uses our subset of the UMTranslog domain to generate solution plans (Nau et al, 1991). In the previous work, Gen+ was integrated with SHOP to generate solutions that were acquired as cases automatically. In this experiment, we used CBM-Gen+ to generate a case base for SHOP, a correct case reuse HTN algorithm (Mukkamalla & Muñoz-Avila, 2002). SHOP was run with Gen+ and CBM-Gen+, and the same test set.

The UMTranslog knowledge domain used in the experiment contains 85 cases. A problem generator was used to generate the test set. The generator can randomly generate a problem with a certain number of packages. The more packages in a problem, the more complicated the problem will be, and the more time will be needed by SHOP to solve the problem. In our test set, the number of packages was incremented from 1 to 6. Five problems were randomly generated for each number of packages. Thus the test set contained a total of 30 problems.

The generator can formulate randomly a group of problems with the same number of packages once a time. But since SHOP will try to solve these problems separately, there is no difference whether we input a group of problems to SHOP or input the same problems one after another. That is the reason we generated the problems in the test set one by one.

6.3 Results and Discussion

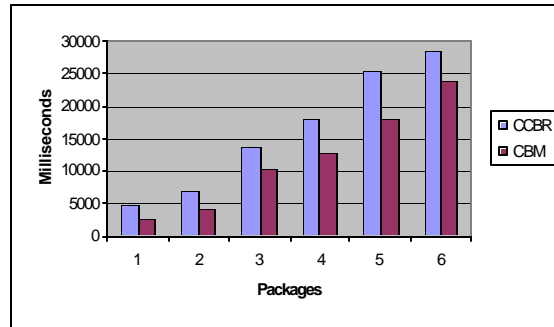


Fig. 6. Average Time for Solvable Problems

Figure 6 shows the results when the input problems are solvable. It plots the number of packages in the problems (x-axis) versus the average required time (y-axis). We observe that when there is a solution for a problem, SHOP with CBM-Gen+

will find the solution using less time than the one with Gen+ by a factor of at least 20% .

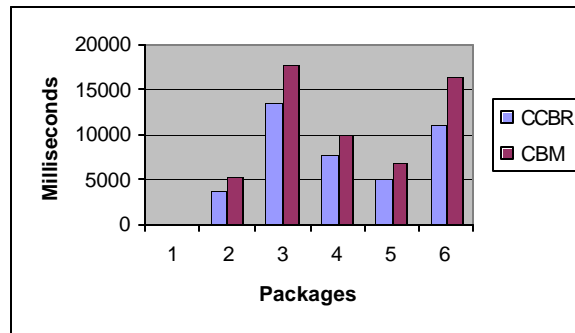


Fig. 7. Average Time for Unsolvable Problems

Figure 7 shows the results for the unsolvable problems. This figure plots the number of packages in the problems (x-axis) versus the average required time (y-axis). In this situation, SHOP with CBM-Gen+ needs at least 20% more time than with Gen+ to determine that there is no solution for the given problem .

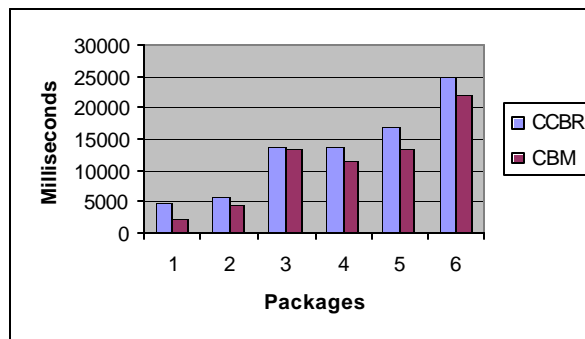


Fig. 8. Average Time for All Problems

Figure 8 shows the number of packages in all the problems of the whole test set versus the average required time (y-axis). We observe that the problem-solving time with CBM-Gen+ is in average faster than Gen+ by a factor of at least 10%.

This experiments shows that despite that more cases are generated with CBM-Gen+, still it runs faster than with Gen+. Notice that case bases generated with both procedures are represented in a plain list. We conjecture that if we add physical links for the corresponding logical links between parent and child cases that result from the CBM-Gen+ procedure, time for problem solving will be reduced even further.

7 Related Work

In the system CaMeL (Elgami et al, 2002), a process is shown that allows automatic elicitation compiled knowledge forms from cases. A similar kind of process developed in (Hanney, K., & Keane, 1996) for eliciting rules from cases. In our work, we propose to modify existing generalized cases rather than extracting rules/compiled knowledge from the cases.

Prodigy/Analogy is a case-based planning system that uses a similar criterion to the one discussed in Figure 2 (Velo, 1994). Cases having common conditions are grouped together for speeding the retrieval process. There is a fundamental difference with our approach: the domains in Prodigy/Analogy are complete. That is, there is a complete domain theory for generating plans. The role of the cases in Prodigy/Analogy is to provide meta-level knowledge by indicating how to use the domain theory for solving problems efficiently. In contrast, cases in our approach contain knowledge about the domain otherwise unavailable. As a result, plans in Prodigy/Analogy are always correct and thus case revision, the main motivation for the CBM-Gen+ algorithm, is unnecessary. As a result, situations such as the ones illustrated in Figures 3 and 4 are not considered in Prodigy/Analogy.

In the PARIS (Bergmann & Wilke, 1995), a method to abstract and generalize concrete cases is also introduced. Then difference between the PARIS and GGS is that the first one requires a complete domain theory and abstract operators, while the latter one captures cases without requiring any domain theory or operators.

8 Discussion and Final Remarks

CBM-Gen+ will find fewer solutions than Gen+ when conflicting cases are captured. In our examples, we constructed two case bases, one for Gen+ and the other one for CBM-Gen+, using the same problem-solution pairs as input (see Tables 1-3). We observed that in the CBM-Gen+ case base, Case 1 is not applicable in situations in which Case 2 is applicable. In contrast, Case 1 is always applicable when Case 2 is applicable in the Gen+ case base. The lost coverage in CBM-Gen+ relative to Gen+ corresponds to those solutions generated with conflicting cases. As we discussed, these are the solutions that may cause Gen+ not to be sound relative to the captured solutions. With the CBM-Gen+ algorithm, in the worst situation, if the number of cases in the original case base is N , then the newly generated case base will have $2N - 1$ cases. Thus, the case base generated by CBM-Gen+ will run into the utility problem only if the original case base runs into this problem.

Whether to use CBM-Gen+ or Gen+ is a trade-off issue. For those domains in which conflicting cases cause no loss of soundness, using Gen+ is adequate since more coverage will be gained. On the other hand, CBM-Gen+ is a better alternative for domains in which the soundness is lost as a result of conflicting cases. For example, in knowledge-based planning, user confidence, which is a crucial requirement, can be improved as a result of the gains in soundness.

As discussed in the section 2.2, not generalizing the cases will demand very large case bases. This is the reason why we use case generalization. One could argue that

we may not need many concrete cases if a more powerful similarity procedure and/or adaptation procedure is used. However, trying to obtain such procedures will just shift the generalization problem that we are addressing in this paper from the case representation container to the similarity knowledge and/or the adaptation knowledge container.

We also show experiments demonstrating that CBM-Gen+ runs faster than Gen+ on the same domain and input cases with which Gen+ was originally evaluated. We conjecture that if we add physical links for the corresponding logical links between parent and child cases that result from the CBM-Gen+ procedure, the time for problem solving will be reduced even further.

References

- Aha, D.W., and Breslow, L. Refining conversational case libraries. In: *Proceedings of the Fourth European Workshop on Case-Based Reasoning (EWCBR-98)*. Providence, RI: Springer, 1998.
- Bergmann, R. & Wilke, W. (1995). Building and refining abstract planning cases by change of representation language. *Journal of AI Research*. Volume 3, (pp. 53-118).
- Erol, K., Nau, D., & Hendler, J. HTN planning: Complexity and expressivity. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 123-1128). Seattle, WA: AAAI Press, 1994.
- Ihrig, L. & Kambhampati, S. Design and implementation of a replay framework based on a partial order planner. In Weld, D., editor, In: *Proceedings of AAAI-96*. IOS Press, 1996.
- Ilghami, O., Nau, D.S., Muñoz-Avila, H., & Aha, D.W. (2002) CaMeL: Learning Methods for HTN Planning. To appear in *Proceedings of the The Sixth International Conference on AI Planning & Scheduling (AIPS'02)*, 2002.
- Hanney, K., & Keane, M. T. *Learning Adaptation Rules From a Case-base*. August. In I. Smyth & B. Faltings (Eds.), *Advances in Case-Based Reasoning*. Amsterdam: Springer Verlag, 1996.
- Kitano, H. and Shimazu, H., The Experience-Saring Architecture: A Case Study in Corporate-Wide Case-Based Software Quality Control *Case-Based Reasoning*, AAAI/MIT Press, 1996
- Leake, D.B, & Wilson, D. Categorizing Case-Base Maintenance: Dimensions and Directions. In: *Advances in Case-Based Reasoning: Proceedings of EWCBR-98*, Springer-Verlag, Berlin. 13 pages.
- Mukammalla, S. & Muñoz-Avila, H. Case Acquisition in a Project Planning Environment. In proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR-02). Springer, 2002.
- Muñoz-Avila, H., Aha, D.W., Nau D. S., Breslow, L.A., Weber, R., & Yamal, F. SiN: Integrating Case-based Reasoning with Task Decomposition. To appear in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*. Seattle, WA: AAAI Press, 2001.
- Muñoz-Avila, H., Gupta, K., Aha, D.W., Nau, D.S. Knowledge Based Project Planning. To appear in *Knowledge Management and Organizational Memories*. 2002.
- Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. Stockholm: AAAI Press, 1999.
- Racine, K., & Yang, Q. Maintaining Unstructured Case Bases, in *Case-Based Reasoning Research and Development*, In Leake D., B., and Plaza E., (Eds). *Proceedings of the International Conference on Case Based Reasoning*, Springer, 1997.

- Smyth, B., and Keane, M.T., Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. *In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*. AAAI Press, 1995.
- Veloso, M. *Planning and learning by analogical reasoning*. Berlin: Springer-Verlag, 1994.
- Watson, I., *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publishers, 1997.