A Domain-Independent System for Case-Based Task Decomposition without Domain Theories

Ke Xu and Hector Muñoz-Avila
Department of Computer Science and Engineering, Lehigh University
{kex2, hem4}@lehigh.edu

Abstract

We propose using domain-independent task decomposition techniques for situations in which cases are the sole or the main source for domain knowledge. Our work is motivated by project planning domains, where hierarchical cases are readily available, but neither a planning domain theory nor case adaptation knowledge is available. We present DInCaD (Domain-Independent System for Case-Based Task Decomposition), a system that encompasses case retrieval, refinement, and reuse, following from the idea of reusing generalized cases to solve new problems. DInCaD consists of a case refinement procedure that reduces case over-generalization, and a similarity criterion that takes advantage of the refinement to improve case retrieval precision. We will analyze the properties of the system, and present an empirical evaluation.

Introduction

One of the main motivations for case-based reasoning (CBR) is that in many domains, cases (i.e., previous problem-solving episodes) are readily available. This is one of the crucial reasons for successful applications of CBR to help-desk, diagnosis and prediction tasks (Watson 1997). Despite these successes, a stumbling block for using CBR in an even wider range of application domains is the difficulty to develop adequate case reuse techniques. Most CBR applications deal with analysis tasks such as classification. An important reason for this situation is that relatively simple domain-independent case reuse techniques, such as taking a majority vote of the classification from similar cases, have been proven to be effective for analysis tasks. In contrast, few deployed CBR applications exist for synthesis tasks such as planning. For synthesis tasks, domain-independent case adaptation techniques exist but require complete planning domain theories, which are not available in many domains. An alternative is to develop domain-specific case adaptation techniques. But developing such techniques is also frequently unfeasible because of the large knowledge acquisition effort involved.

In this paper we present DInCaD, a system for domain-independent task decomposition, designed to deal with situations in which cases are the sole or the main source for planning domain knowledge. DInCaD encompasses case retrieval, refinement, and reuse. It reuses generalized cases to solve new problems. DInCaD consists of a case refinement procedure that reduces case over-generalization, and a case similarity criterion that takes advantage of the case refinement to improve retrieval precision. We support our claim by an experimental validation of DInCaD. We also discuss the properties of the system.

Motivation

Our work is motivated by domains in which cases are readily available but neither a planning domain theory nor case adaptation knowledge is available. An example is project planning, which is a business process for successfully delivering one-of-a kind products and services under real-world time and resource constraints (PMI 1999). Project planning covers several domains, including research proposal development, public events organization, and civil construction management. Several software systems for project planning are commercially available. These systems provide tools for editing work-breakdown structures (WBS), which indicate how complex tasks can be decomposed into simple work units.

Authors have found that there is a one-to-one mapping between elements in a WBS and a hierarchical plan (Mukkamalla and Muñoz-Avila 2002). Based on this mapping, an algorithm has been developed that automatically captures hierarchical cases from a commercial project planning system.

Related Work

Table 1 compares several case-based reasoning systems against DInCaD. The comparison explores six features. The first two features, DT and CB, indicate if the systems require a planning domain theory or a case base, respectively. The next two features, DI and DS indicate if the case adaptation procedure is domain-independent or if domain-specific adaptation rules are required. The last two features, ST and AT, indicate if the system performs planning tasks or analysis tasks, respectively.

System	DT	CB	DI	DS	ST	AT
CHEF		√		√	√	
Prodigy/Analogy	√	√	√		√	
SiN	√	√	√		√	
Ensemble		√	√			√
DInCaD		√	√		√	

Table 1: Comparisons between different systems. Conventions: DT=Domain theory; CB=Case Base; DI=Domain Independent; DS=Domain Specific; ST=Synthesis Tasks; AT=Analysis Tasks.

CHEF is representative of case-based planning systems in which no planning domain theory is required (Hammond 1986). Instead, these systems use cases to represent domain knowledge, and require to encode domain-specific case adaptation rules. DInCaD also uses cases to represent knowledge, but the case adaptation knowledge is domain-independent.

Prodigy/Analogy is representative of case-based planning systems that use cases as search control knowledge (Veloso 1994). These systems assume that a complete domain theory is available, and implement domain-independent case adaptation procedures. Paris is another example of such systems. It uses taxonomical relations and the domain theory to generate and reuse abstract cases (Bergmann and Wilke 1995).

SiN (Muñoz-Avila et al. 2001) is a case-based planning system that requires both a planning domain theory and a case base. Cases represent domain knowledge that enhances the domain theory. SiN implements a domain-independent case reuse procedure. DInCaD does not require a domain theory.

Ensemble classifiers combine votes from individual classifiers to classify new problems. This kind of adaptation method has been used successfully for classification tasks (Dietterich 1997). DInCaD performs synthesis tasks (i.e., task decompositions).

To our knowledge, DInCaD is the first case-based reasoning system that can perform task decompositions (i.e., synthesis tasks) with a domain-independent case adaptation procedure, using cases as its sole or main source of domain knowledge.

Our work is also related to learning planning domain theories from episodic knowledge. The CaMeL system (Ilgami et al. 2002) uses the Candidate Elimination Algorithm to obtain domain theories for hierarchical planning from solution traces. CaMeL requires a complete set of operators. It also requires the current state to be annotated at each planning step in the input solution traces. DInCaD does not require these annotations. The DISTILL system learns domain-specific planners from an input of plans that have certain kinds of annotations (Winner and Veloso 2003). The input includes the initial state and the effects of each action in a plan. DInCaD does not require this information.

Preliminaries

To perform hierarchical decompositions, we follow the principles of Hierarchical Task Network (HTN) planning as in the SHOP system (Nau et al. 1999) and case reuse as in the SiN system. HTN planning achieves complex tasks by decomposing them into simpler subtasks. Planning continues by decomposing the simpler tasks recursively until tasks representing concrete actions are generated. These actions form a plan achieving the high-level tasks. In addition to obtaining these plans, we are also interested in the task hierarchy that led to these plans because the task hierarchy is a WBS in project planning.

The main knowledge artifacts that indicate how to decompose tasks are called methods. A ***method***, *M*, is a 3-tuple: (*h*,*Q*,*ST*), such that: *h*, called the ***head*** of *M*, is the task being decomposed; *Q*, called the ***conditions***, are the preconditions required for using the method; and *ST* are the ***subtasks*** achieving *h*. To achieve a task that can be decomposed (called a ***compound*** task), an HTN planner searches for applicable methods. A method *M* is ***applicable*** to a compound task *t*, relative to a ***state*** *S* (a set of ground atoms), iff match(*h*,*t*) (i.e., *h* and *t* have the same predicate and arity, and a consistent set of bindings *Q* exists, which maps variables to constants so that all terms in *h* match their corresponding ground terms in *t*) and *Q* are ***satisfied*** by *S* (i.e., there exists a consistent extension *Q'* of *Q* such that $\forall q \in Q\ \{qQ' \in S\}$ and $\forall \neg q \in Q\ \{qQ' \notin S\}$). To achieve a task that represents an action (called a ***primitive*** task), HTN planners use operators. An ***operator*** *O* is of the form (*h*,*al*,*dl)*, such that: *h* (the operator's head) is a primitive task, and *al* and *dl* are the so-called ***add-list*** and ***delete-list***. The two lists define how the operator will transform the current state *S* when applied: every atom in the add-list is added to *S* and every atom in the delete-list is removed from *S*. An operator *O* is ***applicable*** to a primitive task *t*, relative to a state *S*, iff match(*h*,*t*). A ***planning problem*** is a triple (*T*,*S*,*D*), where *T* is a set of tasks, *S* is a state, and *D* is a ***planning domain theory*** -- a collection of methods and operators. A ***plan*** is a collection of primitive tasks. Informally, given a planning problem (*T*,*S*,*D*), the collection of primitive tasks that recursively decompose all compound tasks in *T*, relative to *S* and *D*, is a ***correct*** plan (Nau et al. 1999).

A ***case*** *C* has the same form as a method, (*h*,*Q*,*ST*). The only difference is that in a case the task *h*, the tasks in *ST* and the conditions in *Q* are all ground (i.e, containing no variables). The rationale is that cases capture concrete episodes (e.g., how a delivery task was accomplished in an specific project plan). Cases can also be used to decompose tasks. A case *C* is ***applicable*** to a compound task *t*, relative to a state *S* iff *t* and *h* are identical, and the conditions in *Q* are satisfied by *S* (i.e., $\forall q \in Q\ \{q \in S\}$ and $\forall \neg q \in Q\ \{q \notin S\}$).

We assume that a type ontology is available. This assumption is also motivated by project planning, where cases and type ontologies are frequently available (Xu and Muñoz-Avila 2004). We define a type ontology *W* as a collection of relations. These relations can be of two types: *v isa v'* and *?x type: v*. The relation *v isa v'* indicates that a type *v* is a subtype of another type *v'*. The relation *?x type: v* indicates that a variable *?x* is of a type *v*. These relations extend the applicability of the cases and methods. For example, a condition *q* in a case can also be satisfied if *q* is of the form v_1 *type:* t_1 and there is a condition of the form v_1 *type:* t_2 in the state such that t_2 is a subtype of t_1.

Overview of DInCaD

The case applicability criterion requires the current task being decomposed to be identical to the task of the case. This implies that if there are *n* tasks, each with an average number of arguments, m, and each argument can take an average number of instantiations, i, the number of cases required to decompose any task will be $n * m^i$. This is only the minimum number of cases as it is desirable to have alternative cases for some tasks.

To reduce the number of cases required, there are two alternatives. The first alternative is to relax the task equality criterion by defining similarity metrics between non-identical ground tasks. Similarity metrics that use taxonomical representations for cases have been proposed (e.g., (Stahl and Bergmann 1998)). This alternative also requires to create a case reuse mechanism for transforming the ground subtasks of the case into other ground tasks. This alternative is typical of case-based planning systems such as CHEF (e.g., (Hammond 1986)) that rely on cases as the main source of knowledge. The second alternative is to generalize cases and use task matching during case retrieval and HTN task decomposition for case reuse. These two alternatives are related in that an implicit generalization is performed when computing similarities between non-identical ground tasks. They both have to deal with the issue of the correctness of any plan found, because cases are generalized (explicitly or implicitly) and reusing them may yield incorrect plans. We selected the second alternative because HTN task decomposition is well defined, which avoids the knowledge engineering effort to obtain domain-specific adaptation procedures.

We define a ***generalized case*** as a 4-tuple $gC = (h,Q,P,ST)$, where h, Q, and ST are the head, conditions, and subtasks as in the definition of a method. A generalized case is applicable to a compound task t, relative to a state S, iff match(h,t) and the conditions in Q are satisfied by S. P is a collection of preferences, which are annotations used to rank applicable cases. We distinguish between two kinds of preferences: constant and type preferences. ***Constant preferences*** have the form *equal ?v c*, indicating that a variable *?v* takes the value *c*. Constant preferences annotate in the generalized case the original bindings from the case used to obtain the generalized case. ***Type preferences*** have the form *not ?v type: t*. This preference indicates that the variable *?v* is not of type *t*.

Generalized Case Retrieval

Given a task t and a state S, there might be several applicable generalized cases. We define a similarity criterion that is biased towards giving a higher similarity value to the more specific generalized cases. The following is the similarity criterion:

$$\boldsymbol{sim(gC, CB, Prob) = appl*(w_1 * fc + w_2 * ftp + w_3 * fcp)}$$

where $gC = (h,Q,P,ST)$ is a generalized case, CB is the case base containing gC, and $Prob = (t,S)$ is a task-state pair (i.e., the current task being decomposed and the current state). The formula returns values between 0 and 1. The elements in the formula have the following properties:

- The factor *appl* can take a value of either 0 or 1. It takes a value of 1 if gC is applicable to (t,S), and a value of 0 otherwise.
- The value of fc is obtained by dividing the number of satisfied conditions in Q by the maximum number of conditions of any case in CB.
- The value of ftp is obtained by dividing the number of satisfied type preferences by the total number of type preferences in gC. If gC has no type preferences, ftp is assigned a value of 1 (as we will explain later, a very specific generalized case may not have type preferences).
- The value of fcp is obtained by dividing the number of satisfied constant preferences by the total number of constant preferences in gC. Generalized cases always contain constant preferences.
- The weights w_1, w_2 and w_3 are constants such that $0 < w_i < 1$ (i = 1,2,3) and $w_1 + w_2 + w_3 = 1$.

We assigned the weights w_1, w_2 and w_3 such that for any two applicable cases: (1) the one with the higher percentage of satisfied preferences will be ranked higher, and (2) if they have the same percentage of satisfied preferences, the one with more conditions will have a higher similarity value.

Cases are ranked according to their similarity values. If the case-based reasoning process is automatic, the highest ranked case is selected. If it is interactive, the user selects one of these cases. Typically, cases with higher similarity values are preferred, because they represent a recommendation for more suitable cases from the system (Aha and Breslow 1998).

Generalized Case Reuse

Once a generalized case $gC = (h,Q,P,ST)$ is retrieved for a task-state pair (t,S), it is reused in standard HTN planning fashion. If Q is a substitution fulfilling the applicability requirement of gC, the task t is decomposed with the subtasks STQ. The task decomposition process continues recursively with the subtasks until primitive tasks are obtained. If operators are available, they are applied to transform the current state as required in HTN planning. However, when no operators are available, DInCaD still obtains the decompositions which are viewed as WBSs for project planning tasks. For the discussion on the theoretical properties and for the empirical evaluation, we will assume that the operators are available, so we can state the correctness of the obtained plans.

Eliciting and Refining Generalized Cases

We now discuss how generalized cases are elicited and refined. This process is done in three phases: the simple case generalization phase, the constant preference phase, and the type preference phase.

Simple Case Generalization Phase

The simple case generalization phase obtains a generalized case *gC* = (*h'*,*Q'*,*P*,*ST'*) from case *C* = (*h*,*Q*,*ST*) by replacing each constant *y* in C with a unique variable *?y*, if the type, *v*, of *y* is known. In this situation, the condition *?y type: v* is added to *Q'*. If the type of *y* is unknown, *y* is kept as a constant in *gC*. In addition, the condition: *different ?x ?y* is added to *Q'*, for each two different variables *?x* and *?y* of the same type.

Table 2 shows a case and its corresponding case generalization. The case, *C*, accomplishes a task of delivering a piece of equipment, e_3, between two offices, o_7 and o_9. The task is accomplished by contracting a delivery company, dc_2. The generalization, *gC*, replaces constants with variables and adds condition 5.

C	*gC*
Task: deliver e_3 o_7 o_9 **Condition**: 1. e_3 *type:* Equipment 2. o_7 *type:* Office 3. o_9 *type:* Office 4. dc_2 *type:* Delivery_Company **Subtask**: contract dc_2 e_3 o_7 o_9	**Task**: deliver $?e_3$ $?o_7$ $?o_9$ **Condition**: 1. $?e_3$ *type:* Equipment 2. $?o_7$ *type:* Office 3. $?o_9$ *type:* Office 4. $?dc_2$ *type:* Delivery_Company 5. different $?o_7$ $?o_9$ **Preferences**: <none> **Subtask**: contract $?dc_2$ $?e_3$ $?o_7$ $?o_9$

Table 2: A case and its simple case generalization

Constant Preference Phase

Reusing generalized cases will result in a larger coverage compared to reusing ground cases. Coverage is defined as the set of problems that can be solved by using a case base. The reason for this larger coverage is that each binding of the variables in a generalized case will result in a new plan. However, the major drawback is that incorrect plans can be generated. Suppose a case *C* solves a problem *P*, and a generalized *gC* is obtained from *C*. Without the original bindings from *C*, there is no guarantee that *gC* will be retrieved if *P* is given again. It is easy to construct a situation where retrieving other generalized cases would yield an incorrect plan.

To address this limitation, DInCaD adds constant preferences to *gC* based on the original constants in *C*. A new constant preference of the form *equal ?con con* is added for each constant *con* in *C* and its corresponding variable ?*con* in *gC*. In the generalized case shown in Table 2, the following preferences are added: *equal* $?e_3$ e_3, *equal* $?o_7$ o_7, *equal* $?o_9$ o_9 and *equal* $?dc_2$ dc_2. If *P* is given again, *gC* will have all of its constant preferences satisfied whereas other cases will have some constant preferences not satisfied. Based on the similarity criterion, *gC* will be preferred. As we are going to explain later, the constant preferences will ensure a restricted form of soundness.

Type Preference Phase

There are situations in which more than one generalized case can be applicable to the same problem. Depending on the case retrieved, an incorrect plan may be obtained. These situations are referred as ***case over-generalization***. To reduce case over-generalization, our bias is to select the case that are more specific relative to type ontology *W*. To implement this bias, DInCaD adds type preferences to the cases. As a result, the more specific cases will have higher similarity values according to the similarity criterion. Our experiments show the adequacy of this bias.

As an example, consider the two generalized cases in Table 3. The case gC_1 achieves a task to deliver a liquid, $?e_1$, between two locations, $?d_1$ and $?d_3$, using a tanker truck, $?t_5$. The case gC_2 achieves a task to deliver a perishable liquid, $?e_4$, between two locations, $?d_6$ and $?d_7$, using a refrigerated tanker truck, $?t_1$. Consider a type ontology *W* defining the following relations: *RefrigTanker isa Tanker*, *RegularTanker isa Tanker,* and *PerishableLiquid isa Liquid*. Suppose that a new problem is given where a perishable liquid has to be delivered between two locations, and that two trucks are available, one is a refrigerated tanker truck and another one is a regular tanker truck. Both cases are applicable. Reusing gC_2 will result in a correct plan since it will pick the refrigerated tanker. However, reusing gC_1 may not yield a correct plan if it picks the regular tanker. Because *RefrigTanker* and *RegularTanker* are siblings in *W*, there is no criterion to select one over the other one.

gC_1	gC_2
Task: deliver $?e_1$ $?d_1$ $?d_3$ **Condition**: 1. $?t_5$ *type:* Tanker 2.$?e_1$ *type:* Liquid 3.$?d_1$ *type:* Depot 4.$?d_3$ *type:* Depot 5.different $?d_1$ $?d_3$ **Subtask**: drive $?t_5$ $?e_1$ $?d_1$ $?d_3$	**Task**: deliver $?e_4$ $?d_6$ $?d_7$ **Condition**: 1. $?t_1$ *type*: RefrigTanker 2. $?e_4$ *type:* PerishableLiquid 3. $?d_6$ *type:* Depot 4. $?d_7$ *type:* Depot 5. different $?d_6$ $?d_7$ **Subtask**: drive $?t_1$ $?e_4$ $?d_6$ $?d_7$

Table 3: Two generalized cases that may result in case over-generalization

In this situation, DInCaD adds type preferences *not* $?t_5$ *type: refrigTanker* and *not* $?e_1$ *type: perishableLiquid* to gC_1. As a result of adding these preferences, gC_2 will have a higher similarity value than gC_1 in the same situation as before. If gC_1 picks the regular tanker, it will only satisfy one of the two type preferences (i.e., *not* $?t_5$ *type: refrigTanker*). The value of *ftp* will be 0.5. The case gC_2 will have no type preferences because it is more specific and, therefore, *ftp* will have a value of 1.

Properties of DInCaD

Our analysis focuses on situations where the domain knowledge consists of cases, with or without methods. We also assume that a set of operators is available, which is necessary to be able to state the correctness of a plan.

Let's consider the following case bases and their corresponding case retrieval procedures:

- CB-c, a case base consisting of cases. Case retrieval selects any applicable case.
- CB-S, the case base obtained by having the simple case generalization of cases in CB-c. Case retrieval selects any applicable generalized case.
- CB-CP, the case base obtained by adding constant preferences to the cases in CB-S based on the corresponding cases in CB-c. Case retrieval selects the most similar case (assuming *ftp* = 0).
- CB-CTP, the case base obtained by adding type preferences to the cases in CB-CP. Case retrieval selects the most similar case.

If *I* is an incomplete planning domain theory (i.e., only a subset (possibly empty) of the methods for the target domain is known) and *CB* is a case base, then a planning domain theory *D* is ***consistent*** with a knowledge base $I \cup CB$ iff: (1) Every method and operator in *I* is an instance of a method or operator in *D*, and (2) For every case *C* = (*h'*,*Q'*,*P*,*ST'*) in *CB*, there is a method *M* = (*h*,*Q*,*ST*) in *D* such that *h'*, *Q'*, and *ST'* are instances of *h*, *Q* and *ST*, respectively. Particularly, we are interested in planning domain theories that are consistent with $I \cup$CB-c, because cases in CB-c represent the episodic knowledge we have about the domain.

Theorem 1. The following statements are true:

1. Plans obtained by using $I \cup$CB-c are correct in every planning domain theory consistent with $I \cup$CB-c.
2. There are planning domain theories consistent with $I \cup$CB-c such that plans obtained with $I \cup$CB-S, $I \cup$CB-CP, or $I \cup$CB-CTP are not correct.

Now lets consider the following definition. Let $PS = \{(p_1, sol_1), (p_2, sol_2), \ldots, (p_n, sol_n)\}$ be the problem-solution set used to generate CB-c. Each p_i is a task-state pair (T_i, S_i) and each sol_i is a plan to p_i. A case base *CB* is ***sound relative to PS*** iff whenever a p_i in *PS* is given as a problem, the plan generated by using $I \cup$CB is correct in any planning domain theory *D* consistent with $I \cup$CB-c. The rationale behind this definition is that at the very least the case-based planner should find correct plans to the problem-solution pairs it has previously acquired as cases.

The notion of soundness relative to *PS* does not imply that the plan obtained by using *CB* must be sol_i when the problem p_i is given again. Suppose that C_i is the case obtained from (p_i, sol_i), and that when p_i is given again as a problem, a different case C_k is retrieved. Under the assumptions, C_i must imply C_k. A case C_i ***implies*** a case C_k if whenever C_i can be retrieved, C_k can also be retrieved. This happens if (1) the tasks of C_i and C_k are identical, (2) there is a one-to-one mapping from conditions in C_i into conditions in C_k such that for each condition *q* in C_k, *q* is also a condition in C_i, or if a condition of the form v_1 *type:* t_1 occurs in C_k but not in C_i, then there must be a condition of the form v_1 *type:* t_2 in C_i, such that t_2 is a subtype of t_1.

Theorem 2. The following statements are true:

1. $I \cup$CB-c is sound relative to *PS*.
2. $I \cup$CB-S is not sound relative to *PS*.
3. $I \cup$CB-CP is sound relative to *PS*.
4. $I \cup$CB-CTP is sound relative to *PS*.

The last issue is the coverage of the case bases (Smyth & Keane, 1995). We extend the traditional notion of coverage of case bases to coverage of a knowledge base $I \cup CB$, defined as:

Coverage(IÈCB) = {*p*: *p* is a planning problem that can be solved by using $I \cup CB$}

The definition of coverage does not indicate if the plan found for *p* is correct or not. It just indicates if a plan can be found or not. We state the following theorem:

Theorem 3. The following statements are true:

1. Coverage($I \cup$CB-c) $\subseteq$ Coverage($I \cup$CB-S)
2. Coverage($I \cup$CB-S) = Coverage($I \cup$CB-CP) and Coverage($I \cup$CB-CP) = Coverage($I \cup$CB-CTP)

From this analysis, we conclude that, compared to the other two case bases, CB-CTP and CB-CP have the largest coverage while preserving soundness relative to problem-solution sets. However, as we will see in the experimental evaluation, CB-CTP will result in a better retrieval precision than CB-CP.

Empirical Evaluation

We performed experiments to measure the precision and recall of retrieval with CB-S, CB-CP, and CB-CTP. In the context of case-based planning, we define precision as A/B and recall as A/(A+C), where A is the number of times that correct plans were generated, B is the number of times that plans were generated, and C is the number of times that a correct plan existed but either an incorrect plan or no plan was generated.

For the experiments we used two synthetic domains to be able to state if the generated plans are correct or not. We implemented a variant of the UM Translog domain (Andrews 1996). In this domain, trucks and airplanes are used to transport packages between different sites. A type ontology of vehicles and packages is defined so that vehicles can only deliver compatible packages. For example, medium trucks transport medium or small packages, but not large packages. We also implemented an HTN version of the process planning domain reported in (Muñoz-Avila and Weberskirch 1996). In this domain, plans for manufacturing rotary symmetrical workpieces are generated. These plans must consider inter-relations between various parts of the workpieces and available manufacturing tools.

Each domain was used to generate cases by using a training set consisted of 240 randomly generated problems. These cases were used to generate CB-S, CB-CP, and CB-CTP as explained before. We then replaced the methods from the original domain with the three case bases. Thus, we created three knowledge bases consisting of cases and operators only. The test set consisted of 50 new problems randomly generated in the same domain. A threshold ***a*** was set such that the highest ranked generalized case *gC* satisfying *sim*(*gC*, *CB*, *P*) ≥ ***a*** was selected.

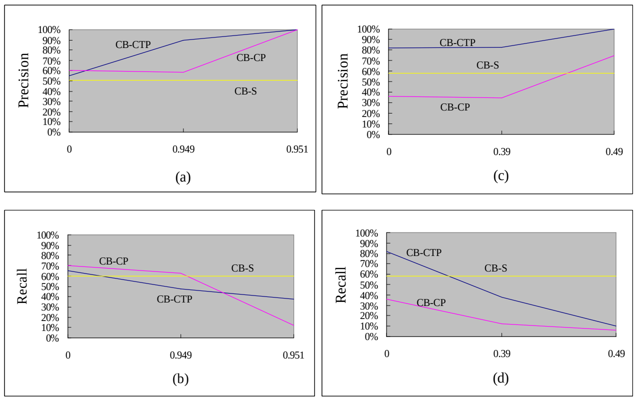

Figure 1: (a) Precision and (b) recall for the UM Translog domain. (c) Precision and (d) recall for the process planning domain. The x-axis takes the values of *a*. The y-axis goes from 0% to 100%.

Figure 1 shows the results from the two domains. The value of *a* was set to 0, 0.949 and 0.951 for the UM Translog domain, and 0, 0.39 and 0.49 for the process planning domain. The results show that CB-CTP has a much better precision than CB-CP and CB-S. For the UM Translog domain we get best results when *a* is set to 0.949. CB-CTP reaches an precision of almost 90% compared to 60% for CB-CP and 50% for CB-S. At the same time, the recall is only reduced by a factor of 10%. Only when *a* is set to 0.951 in the UM Translog domain do the precisions of CB-CTP and CB-Gen become the same. But at this point the recall is too low. For the process planning domain, we get best results when *a* is set to 0. CB-CTP has an precision of 80% compared to 40% for CB-CP and 60% for CB-S. At the same time, the recall is 20% higher than CB-S and 40% higher than CB-CP. We conclude that using CB-CTP will result in a better performance, informally defined as a balance between precision and recall, than using CB-CP and CB-S.

Final Remarks

DInCaD is a system for domain-independent task decomposition designed to deal with situations in which cases are the sole or the main source for planning domain knowledge. DInCaD generalizes cases to improve coverage and adds constant preferences to preserve soundness relative to the task-state pairs. To improve retrieval precision, DInCaD refines the cases by adding type preferences, and defines a similarity criterion that takes these preferences into account. Our experiments show an improvement in retrieval precision and a balance between precision and recall with certain values of the threshold *a*. For future work, we will study techniques to determine these values of *a* in advance.

References

Aha, D.W., and Breslow, L. Refining conversational case libraries. In *Proceedings of EWCBR-98*. Providence, RI: Springer, 1998.

Andrews, S., Kettler, B., Erol, K., & Hendler, J. *UM Translog: A Planning Domain for the Development and Benchmarking of Planning Systems*. Technical Report, Dept. of CS, Univ. of Maryland at College Park, 1995.

Bergmann, R., and Wilke, W. 1995. Building and refining abstract planning cases by change of representation language. *JAIR*, 1995.

Bergmann, R. and Stahl, A. Similarity Measures for Object-Oriented Case Representations. In *Proceedings of EWCBR-98*. Springer, 1998.

Dietterich, T.G.. Machine learning research: Four current directions. *AI Magazine*, 18(4):97--136, 1997.

Hammond, K.J. Chef: A model of case-based planning. In *Proceedings of AAAI-86*. AAAI Press, 1986.

Ilghami, O., Nau, D.S., Muñoz-Avila, H., & Aha, D.W. CaMeL: Learning Methods for HTN Planning. In *Proceedings of AIPS-02*. AAAI Press, 2002.

Leake, D.B, & Wilson, D. Categorizing Case-Base Maintenance: Dimensions and Directions. In *Proceedings of EWCBR-98*, Springer-Verlag, 1998.

Mukammalla, S. & Muñoz-Avila, H. Case Acquisition in a Project Planning Environment. In *Proceedings of 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. In *Proceedings of IJCAI-2001*. AAAI Press, 2001.

Muñoz-Avila & Weberskirch, Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In *Proceedings of AIPS-96*. AAAI-Press, 1996.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. SHOP: Simple hierarchical ordered planner. In *Proceedings of IJCAI-99*. Stockholm: AAAI Press, 1999.

Project Management Institute (PMI). *PMI's A Guide to the Project Management Body of Knowledge*. Technical Report. No.: PMI 70-029-99, 1999.

Smyth, B., and Keane, M.T., Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of IJCAI-95*. AAAI Press, 1995.

Veloso, M. *Planning and learning by analogical reasoning*. Springer-Verlag, 1994.

Watson, I. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufman Publishers, 1997.

Winner, E. and Veloso, M.M. 2003. DISTILL: Learning Domain-Specific Planners by Example. In *Proceedings of ICML-2003*, 2003.

Xu, K., & Muñoz-Avila, H. CaBMA: Case-Based Project Management Assistant. In *Proceedings of IAAI-2004*. AAAI Press, 2004.