

Case Acquisition in a Project Planning Environment

Sasidhar Mukkamalla & Héctor Muñoz-Avila

Department of Computer Science and Engineering
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015, USA
{sam6,munoz}@cse.lehigh.edu

Abstract: In this paper, we propose an approach to acquire cases in the context of project planning, without any extra effort from the end user. Under our definition, a case has a one to one correspondence with the standard elements of a project plan. We exploit this correspondence to capture cases automatically from project planning episodes. We provide an algorithm for extracting cases from project plans. We implemented this algorithm on top of a commercial project-planning tool and perform experiments evaluating our approach.

Introduction

Knowledge acquisition is a problem frequently faced when using intelligent problem-solving techniques in real-world situations. It is well known that over the years intelligent systems have been developed but failed to be used because it was not feasible to feed such systems with adequate knowledge. Research on this area has typically concentrated on developing interfaces to capture knowledge from users. Although a lot of progress has been made in this direction, it is still difficult to convince users to take advantage of such systems because of the overhead needed to learn how to use those interfaces and then using them to feed the intelligent system with the knowledge needed.

We present an alternative for addressing the knowledge-acquisition problem, namely, to extract knowledge from the same interactive tools that users regularly use in achieving their tasks. By doing so, the knowledge acquisition effort becomes transparent to the user. This alternative derives from our ongoing effort to provide a knowledge-layer to enhance project-planning tools.

Project planning is a business process for successfully delivering one-of-a kind products and services under real-world time and resource constraints. In previous work (Muñoz-Avila et al, 2002), a knowledge-layer for existing tools supporting project planning was proposed; the core idea in this proposal called knowledge-based project planning (KBPP) was to reuse cases containing pieces of project plans when creating new project plans. In this paper, we also report on the first implementation of a KBPP on top of a commercial Project Planning (PP) tool, but our primary focus is on the case acquisition capabilities that we developed.

On the core of our case acquisition effort is the definition of cases. As a result of our definition, a case has a one to one correspondence with the standard elements of a

project plan. This notion reflects our belief that by only using standard PP elements to define cases the intelligent component of the KBPP system will be less intrusive, the cases are more natural from the point of view of the end-user and the case acquisition effort will be simplified.

This paper is organized as follows. In the next section we discuss related work. Then, we summarize the proposal for KBPP presented in (Muñoz-Avila et al, 2002) and discuss the first implementation of these ideas. The following section discusses our case capture approach. Then, we discuss an implementation of our case acquisition ideas in a project-planning tool, Microsoft Project. Next we evaluate our approach and finally we make concluding remarks.

Related Work

Providing tools for knowledge acquisition has been a frequently studied research topic. For example, in the EXPECT project (Blythe et al, 2001), an integrated suite of intelligent interfaces is used to capture knowledge. The EXPECT project shows that by integrating these interfaces it is possible to elicit the context of the users actions, which enhances the knowledge acquisition capabilities. This is somewhat related to our work, since the use of a KBPP tool by the user provides the context for the case capture knowledge. However, the main difference lies in that we are not constructing ad-hoc interfaces to capture, instead we are capturing the cases from the data given by the user during his/her regular interactions with the PP tool.

Authors have long observed that the problem-solving episodes can be captured as cases (e.g., Veloso, 1993). If users develop a project plan using tools such as Microsoft Project, the elements of a project plan could be stored as cases. As we will discuss, our method traverses different elements within a project plan in a process that is similar to the foot-printing process, which is used to identify relevant features of a plan generated (Veloso, 1993).

In the system CaMeL (Elgami et al, 2002), a process is shown that allows automatic elicitation compiled knowledge forms from cases. These compiled knowledge forms are called methods and indicate how to decompose tasks following the hierarchical task network representation (HTN) that we use in this work. A similar process developed by Carrick and Cunningham (Carrick and Cunningham, 1993) can elicit rules from cases. In our work, we study the acquisition of cases, which is a complementary problem to these two approaches. We envision a two-step process in which cases are learned using the process described in this paper. In the rest of this paper we will concentrate only on the case acquisition process.

Another related research direction is to capture user intent and use this as conditions for case retrieval. For example, by analyzing the input of a user in an interactive system, intent about the rational for user actions can be inferred (El Fattah, 2001).

Knowledge-Based Project Planning

Several software packages for project management are commercially available. These include *Microsoft Project*TM (Microsoft) and *SureTrak*TM (Primavera Systems Inc). These interactive systems help a planner store a work-breakdown structure (WBS), which indicates how the project's tasks can be decomposed into manageable work units. These packages also contain a suit of tools to control the scheduling of the tasks and the management of resources.

In (Muñoz-Avila et al, 2002), KBPP is proposed to assist planners in the development of WBS by using hierarchical case decomposition techniques. This proposal was based on the recognition that WBS representations, as commonly used in project planning, are very similar to the HTN representations used in the planning community.

Hierarchical Planning

The particular hierarchical planning approach that we use is hierarchical task network planning (HTN). In HTN, high-level tasks are decomposed into simpler tasks, until eventually so-called **primitive** tasks are reached. Primitive tasks indicate concrete actions to be undertaken. Tasks that can be decomposed are called **compound**. For example, creating an advertising strategy can be a high-level task. Giving an advertisement in the local newspaper can be a primitive task. Both tasks can be related, the latter being a descendant of the former, through a hierarchy formed by the task-subtask relations.

The algorithm that we implemented for HTN planning is a variation of an algorithm called SiN (Muñoz-Avila et al, 2001; 2000). SiN uses two sources of information: methods and cases. A **method** has the form $(:method\ h\ P\ ST\ <)$ and indicate the preconditions P to decompose a task h into subtasks ST . $<$ defines an order between the subtasks in ST . Methods are compiled forms of generic knowledge; they are always applicable for any situation for which the preconditions P hold. A method is applicable in a situation or state S , which is defined as a collection of facts, if there is a substitution θ such that the condition $P\theta$ is valid in S . A set of preconditions $P\theta$ is **valid** in S if for each positive precondition, p , p is in S and for each negative precondition, $not(q)$, q is not in S .

Since creating a set of methods, completely describing a domain is in general infeasible, SiN uses cases to enhance the knowledge that a system has about the domain. A **case** has the form $(:case\ h\ P\ ST\ <\ Q)$ where h , P , ST and $<$ play the same role as in the methods and Q are question-answer pairs indicating preferences for the applicability of the case. Cases are examples of decompositions and thus are situation-specific. A knowledge base in SiN consists of cases, methods and operators. Operators indicate the effects of performing the primitive tasks.

As we said before, the algorithm that we implemented for HTN planning is a variation of the SiN algorithm. SiN integrates the SHOP planning system (Nau et al, 1999) and NaCoDAE/HTN, an extension for HTN planning of the NaCODAE conversational CBR (CCBR) system (Aha and Breslow, 1998). To decompose tasks, SiN uses SHOP's methods and NaCoDAE/HTN's cases. At any point of time either

SHOP or NaCODAE is decomposing tasks and will cede control to the other one if it reaches a compound task that it cannot decompose.

Our variation is called SHOP/CCBR as it extends SHOP while mimicking the CCBR style interactions of NaCoDAE. The main advantage of SHOP/CCBR over SiN is that it allows simultaneous consideration of cases and methods to decompose a task instead of SiN's approach of considering either cases or methods but not both at the same time. Another advantage is that cases can include variables and complex applicability conditions like numerical constraints that in SiN are only available for methods. An important point is that SHOP/CCBR like SiN, but unlike SHOP does not require either methods or operators to generate task decompositions. That is, SHOP/CCBR can use cases only to generate task decompositions. This is crucial in the context of project planning, where a complete knowledge base might not be available.

Work-Breakdown Structure and HTNs

A WBS is a hierarchically organized set of *elements* that need to be performed to deliver the required goods and/or services (e.g., creating a marketing strategy). Elements in a WBS can be of two kinds: tasks and activities. *Tasks* can contain activities and other tasks. Activities are terminal nodes (e.g., giving an add in the local newspaper). Elements in the WBS can be ordered using precedent constraints. The mapping of WBS to hierarchical plans is straightforward: the WBS tasks map the HTN compound tasks, the WBS activities map the HTN primitive tasks, and the precedence constraints map the ordering constraints. This mapping means that the AI techniques used for hierarchical plan generation could be used for WBS generation. However as pointed in (Muñoz-Avila et al, 2002), the *main condition for using these techniques is that cases and methods are available*. This is precisely the issue we are addressing in this paper.

Case Contents

In the conversational case-based reasoning approach, the case's question-answer pairs and the preconditions were conceived to indicate conditions under which a particular case is applicable. In the context of KBPP, we want the conditions for applicability of a case to correspond to project plan elements. In a project plan, several elements are maintained, including:

1. The hierarchical relation between elements (i.e., tasks and activities) in the WBS
2. The resources available for the project
3. The assignments between resources and tasks
4. The precedence constraints between elements in the WBS

Accordingly we simplified definition of a case to $(:case\ h\ C\ ST\ <)$. That is, we will have conditions *C* only instead of the preconditions and question-answer pairs defined

for cases in CCBP. These conditions state the collection of task-resource assignments that are relevant for the case. In the HICAP system (Muñoz-Avila et al, 1999), basically the same five elements above are represented. The main difference is that cases use question-answer pairs to assess their applicability rather than the task-resource assignments.

Although the simplification in the form of the cases is only slight, it reflects an important goal of our approach for KBPP. We want the cases to have a one to one correspondence with the standard elements of a project plan. This goal reflects our belief that by using only the standard PP elements, the intelligent component of the KBPP system will be less intrusive, the cases fit into the natural perspective of the end-user and the case acquisition effort will be simplified.

One vs. Multi-Level Cases

A WBS can be seen as a collection of one-level decompositions. Each one level-decomposition refines a single task into (sub)tasks and/or activities. The whole collection decomposes the most high-level tasks (i.e., tasks having no parent tasks) into activities (i.e., the non-decomposable elements of the WBS). In our approach, each one-level decomposition is stored as a single case. The alternative could have been cases that contain several decomposition levels or even a complete WBS. There are some trade-offs that we consider:

- Cases containing several decomposition levels or the complete WBS provide a better or complete picture of the overall plan. Their main drawback is that the case re-usability is limited, as more levels are added, more constraints should be considered to assess the applicability of the WBS in new situations.
- Cases containing one-level decomposition give no overall picture of the plan. Their main advantage is that they can be reused in several situations as cases from different WBS can be combined. For example, it is possible to decompose a task with a case captured from one WBS and to decompose one of the children tasks with a case captured from another WBS.

We decided to store only one-level decompositions in our cases to maximize the re-usability of the cases. As we will see, our experiments suggest that even after solving few problems the coverage provided by the captured cases is greatly improved.

Concrete versus Generalized Cases

Another question that we explored was whether to store in the knowledge base generalized cases or not. Given a case $CA = (:case\ h\ C\ ST\ <)$, we define the generalization of CA , denoted by $\mathbf{Gen}(CA)$, as a case obtained by mapping each parameter, ψ , occurring in the arguments of the head, conditions, or subtasks of CA ,

with a unique variable, $?ψ$. In this way, each occurrence of same parameter in h , C and ST is replaced by the same variable.

Thus, the question was whether CA or Gen(CA) should be stored in the knowledge base. The problem of using Gen(CA) instead of CA is that Gen(CA) might not **correctly model the target domain**. That is, If $X = \{?x_1, \dots, ?x_n\}$ is the set of the variables in Gen(CA), there might be instantiations of X for which Gen(CA) that do not reflect the behavior of the target domain. Using CA has its own problems; the most important of which is the coverage of a case base consisting of concrete cases. For each possible task in the domain we will require one concrete case having the same head. If there is a maximum of n task names, with an average number of arguments, m, and each argument can take an average number of values, v. The minimum number of cases required will be $n \cdot m^v$, one for each possible task. Notice that this is the minimum number, since we compute similarity on the conditions for determining applicability of a case to a given task. So we need several cases for each task to obtain a better coverage.

Our answer to this question was to combine these two approaches; if the **type** for each parameter in the case is known, we generalize the parameter by uniformly replacing it with the same variable. Parameters for which the type is not known are not generalized. In addition we add the following constraints as conditions of the case:

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

We denoted by **Gen+(CA)** the procedure that generalizes only typed parameters as indicated above. Our case acquisition strategy adds this kind of generalized cases to the knowledge base. It is possible that this generalization might still not correctly model the target domain. As more cases are added, these kinds of errors can be detected if the heads of the new and the existing case match, the conditions of one is contained in the other one but their subtasks and/or the orderings are different. We plan to address this issue in future work.

Algorithm for Case Capture

The case acquisition process starts on the most high-level tasks as indicated in the *caseAcquisition* algorithm. PL is the current project plan, which includes the four elements indicated in the previous section. KB is the knowledge base consisting cases and methods (operators present in KB, if any, are ignored).

Algorithm: caseAcquisition(PL,KB)

For each most high-level task t do
captureCase(t, PL, KB)

The caseCapture algorithm is described below. The *assignedResources* function returns the **associations** of a task *t* in the current project plan. The associations of a task is a pair (t, ra) where *ra* consists of:

1. The set of resources associated with *t* and their attributes (e.g., type), and
2. The set of resources associated with each child activity of *t*

The rationale for the first set of resources is that those are the ones specified by the user as relevant for the particular task and these resources are stored as conditions for that particular case. Since activities are not stored as separate cases (they have no subtasks), the resources associated with them are stored as conditions for the cases in which activities appear as tasks.

The subtasks and orderings functions return the subtasks/activities of the task and the orderings between the tasks/activities in *ST* respectively.

Algorithm: captureCase(*t*, PL, KB)

```

If t is an activity then return
C ← assignedResources(t, PL)
ST ← subtasks(t, PL)
ord ← orderings(ST, PL)
storeChecking(Gen+((:case t C ST <)), KB)
for each t' in ST do
    captureCase(t', PL, KB)

```

The procedure storeChecking stores a new case Gen+((:case *t* C ST <)) in KB if there is no case (:case *t'* P' ST' <') and there is no method (:method *h'* P' ST' <') for which the case is an instance of. That is, there is no substitution θ , such that $h'\theta = h$, $P'\theta = C$, $ST'\theta = ST$ and the orderings relations are preserved (i.e., For every t_1 and t_2 in *ST* if $t_1 <' t_2$ holds then $t_1\theta < t_2$ also holds). The reason being that if the conditions *C* of the candidate case (:case *t* C ST <) are applicable in any state *S*, then the corresponding case or method in KB is also applicable in *S*. That is, if there is a substitution α such that $C\alpha$ is valid in *S*, then $P\theta\alpha$ must be valid in *S*. So essentially storeChecking makes sure that no case or method in KB subsumes the candidate case.

The caseAcquisition algorithm terminates because the task-subtask relation in a WBS is always a collection of trees (one tree for each most high-level task) and thus it cannot contain loops. For example, the WBS view of Microsoft Project only allows the addition of tasks/activities as children of a single task.

The process of traversing the WBS identifying relevant resources at each level is similar to the foot-printing process in Prodigy/Analogy (Veloso, 1993). The key difference is that instead of complete knowledge base, we only use the four elements of the project plan to extract the cases. In Prodigy/Analogy the knowledge base is

used to annotate the cases with the operations from which a plan was derived. The whole point of our work is that such a complete knowledge base is not available for many real world problems. This distinction reflects a difference on the role of the cases: in our work cases enhance the knowledge base whereas in prodigy/Analogy (and any other system performing derivational analogy on top of a first-principles planner) cases serve as meta-knowledge to guide the usage of the complete knowledge base.

Case Study: Microsoft Project

Microsoft Project is a popular commercial project management tool. It helps users to store WBS, allocate resources and follow deadlines. It incorporates a Visual Basic editor that allows users to extend its functionality. The WBS is edited in a spreadsheet like window called the **Task View**. All the resources are edited in the **Resource Sheet**. The user can allocate resources to each task separately.

We added a new functionality into Microsoft Project. The user has now the possibility to start a semi-automatic plan generation process. When the user selects a task and chooses the KBPP decompose function, the following steps are performed:

- A KBPP module that is implemented in Microsoft Project reads all the resources and their attributes. Then, it invokes SHOP/CCBR with this task and resources.
- SHOP/CCBR computes all applicable methods and cases for the task and displays them for the user to select one of them.
- The user then selects a case or method and the corresponding decomposition is inserted in Microsoft project.

We used COM (Component Object Model) as the software design model to facilitate the communication between Microsoft Project and SHOP/CCBR.

Figure 1 presents an overview of the information flow of the overall system. The user inputs a project plan, including the resources and resource assignments (label 1 in figure 1). Our case acquisition module implemented within Microsoft Project is used to capture cases from the project plan (label 2 in the figure 1). These cases are used to augment the knowledge base used by SHOP/CCBR. Given a new situation and a task in Microsoft Project (label 3 in the figure 1), SHOP/CCBR is invoked to create a WBS for achieving that task relative to the state.

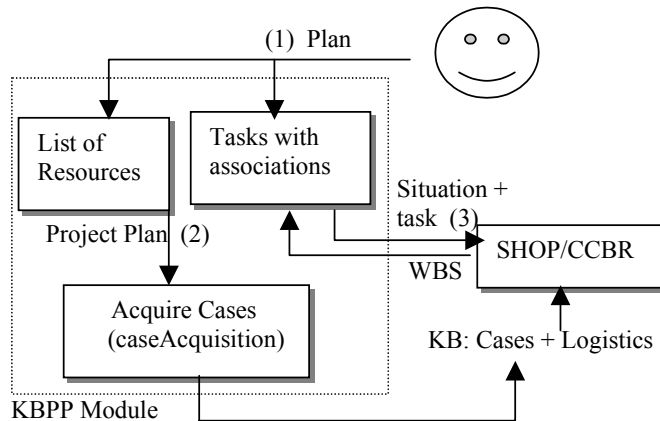


Figure 1: Information flow in KBPP System

As an example consider Figure 2, which shows the Task View of all the tasks and their associated resources of a partial plan created in Microsoft Project. If the user selects the task *Transport Package1 from NottingHill to Hasenheide* to store as a case, the system can potentially learn 4 cases.

Task Name	Resource Names
<input type="checkbox"/> Transport Package1 from Notting Hill to Hasenheide	Package1,Hasenheide,Notting Hill
<input type="checkbox"/> Pick-up Package1 from Notting Hill	Package1,Notting Hill
Load Truck1 with Package1	Truck1
<input type="checkbox"/> Carry Package1 from Notting Hill to Hasenheide	Package1,Notting Hill,Hasenheide
Drive Truck1 from Notting Hill to Heathrow	Truck1,Notting Hill,Heathrow
Unload Package1 from Truck1	Truck1,Package1
Load BA-981 with Package1	BA-981,Package1
Fly BA-981 from Heathrow to Tegel	BA-981,Heathrow,Tegel
Unload BA-981 with Package1	BA-981,Package1
Load Truck2 with Package1	Truck2,Package1
Drive Truck2 from Tegel to Hasenheide	Truck2,Hasenheide
<input type="checkbox"/> Deliver Package1 at Hasenheide	Package1
Unload Truck2 with Package1	Truck2,Package1,Hasenheide

Figure 2: A plan created in Microsoft Project

The head of the case is *Transport ?Package1 from ?NottingHill to ?Hasenheide*. To compute the conditions, the KBPP module reads all the resources associated with the task and their corresponding attributes. Here, the system reads that Package1 is at

NottingHill, NottingHill is a location in London, Hasenheide is a Location in Berlin, London and Berlin are cities. Thus, the resulting conditions of the case are:

(PACKAGE ?Package1)
(PACKAGE-AT ?Package1 ?NottingHill)
(LOCATION ?NottingHill)
(IN-CITY ?NottingHill ?London)
(CITY ?London)
(LOCATION ?Hasenheide)
(IN-CITY ?Hasenheide ?Berlin)
(CITY ?Berlin)
(DIFFERENT ?NottingHill ?Hasenheide)
(DIFFERENT ?London ?Berlin)

The ordering relation < is read from the relation between tasks as stored in Microsoft Project (these orderings can be viewed by the user in the **Gantt Chart** View). The resulting subtasks of the case are:

(Pick-up ?Package1 from ?NottingHill)
(Carry ?Package1 from ?NottingHill to ?Hasenheide)
(Deliver ?Pacakge1 at ?Hasenheide)

Evaluation

We performed an experimental evaluation of our case acquisition strategy. The purpose was to observe the effects of adding more cases to the coverage of the knowledge base. We use the standard definition of coverage i.e. the number of problems that can be solved with the knowledge base (Smyth and Keane, 1995).

Domains

Initially the knowledge base consisted of methods describing the complete 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 between different places. We also defined a subset of the UMTranslog (Erols et al, 1994). Our subset of UMTranslog extends the logistics domain by defining objects that can be a combination of the following types: *regular*, *liquids*, *perishables* and *hazmat*. For example, milk can be defined as a liquid and perishable object. The transportation means in our extension can be a combination of four types: *regular*, *tanker*, *refrigerated* and *hazardous*. Each type is specially suited to transport one object type. For example a refrigerated tanker truck is suitable to transport milk. All the problems solvable by Logistics domain were also solvable by UMTranslog but not the other way around (i.e. only a strict subset of the problems that are solvable with UMTranslog are solvable with Logistics).

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. The solution plan is passed to Microsoft Project as a WBS because there is a one to one correspondence between HTN plans and WBSs. In addition, a specialized procedure is used to represent the preconditions of the methods as resources allocated to that particular high-level task. SHOP and this specialized procedure form the automatic user which play the role that the human user plays in Figure 1.

Our test data was obtained as follows: 70 problems were randomly generated. These were divided in two groups: one consisting of 50 problems, the learning set, and the other one consisting of 20 problems, the test set. Each of the 70 problems is solvable by UMTranslog but none is solvable by Logistics. The learning set was split in 5 groups of 10 problems each. We did 5 runs with each of these 5 groups. At each run each of the 10 problems within the group was solved using SHOP and UMTranslog and passed to Microsoft Project (label 1 in the figure 1). Cases were captured from each of these plans using the caseAcquisition algorithm and stored in the case base (label 2 in the figure 1). Cases captured from previous runs were kept for the following runs. Then, SHOP/CCBR tried to solve each of the 20 test problems using the augmented knowledge base (label 3 in the Figure 1). We thus ran a total of 100 problem-solving episodes.

Extracting Resources from an HTN Plan

As we saw, getting a WBS from an HTN plan is a straightforward process because of the one-to-one correspondence between them. Feeding the information about the resource usage is another matter. Since the purpose of extracting resources from HTN plan was simply to feed Microsoft Project so we can perform our experiments, we decided to write a simple procedure specialized for this domain. For each compound task t in the plan we identified the method that was used to decompose it. For each precondition in the method, we identified the resources mentioned in the precondition and:

- Added each resource to the list of resources in Microsoft Project if the resource was not there already
- Assigned each of these resources to the task t

In our previous example, if the precondition stated that the Package1 is at NottingHill, we added Package1 to the list of resources, assigned this resource to the task *Transport Package1 from NottingHill to Hasenheide*. In the entry for Package1 we added 2 columns: "NottingHill" and "Hasenheide". The order guaranteed that when the resources are extracted by the caseCapture procedure, they match the form

of the preconditions in the UMTranslog domain and we could verify that the cases correspond to instances of the UMTranslog methods.

Results and Discussion

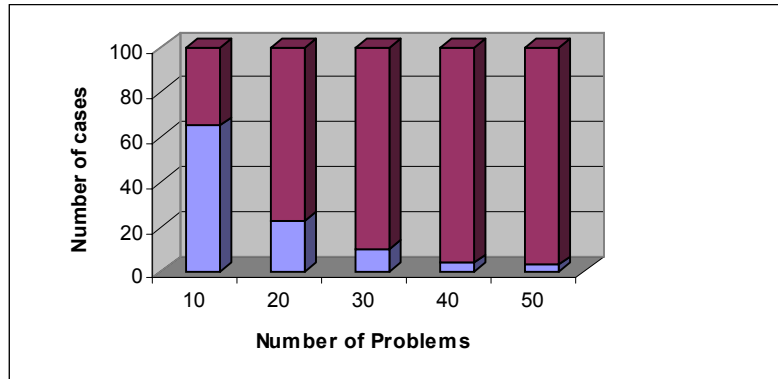


Figure 3: Number of problem in the learning set vs. Number of cases learnt

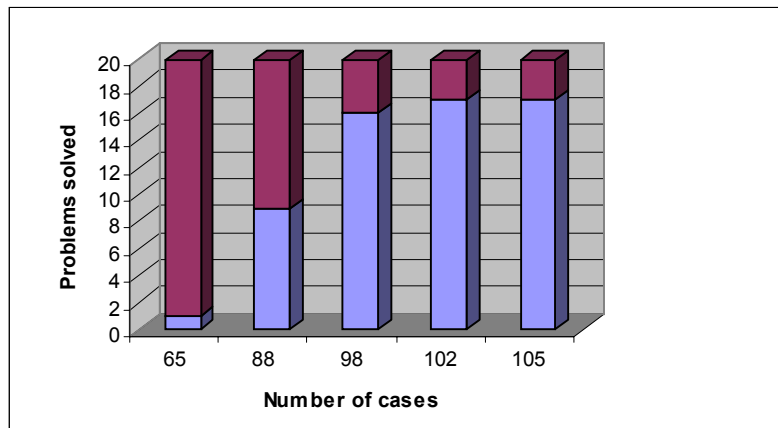


Figure 4: Number of Cases learnt vs. Number of test problems solved

Figure 3 shows the number of problems in the learning set (x-axis) versus number of cases learnt for each learning set (y-axis). We observe that the number of cases learnt for each learning set decreases rapidly as more problems are solved. Figure 4 shows the number of cases learnt versus number of test problems solved. The percentage of problems that are solved augments rapidly as more runs are made. More interestingly we observe a co-relation between the results of both figures. Namely that at the point where fewer cases are learnt, more problems are solved. This shows that

after that point we reach a near saturation point in the coverage of knowledge base. To observe this effect more clearly, Figure 5 shows the normalized change rate of the cases learned and the problems solved. As we can see in the fourth learning set the rate of cases learned and new problems solved is very small.

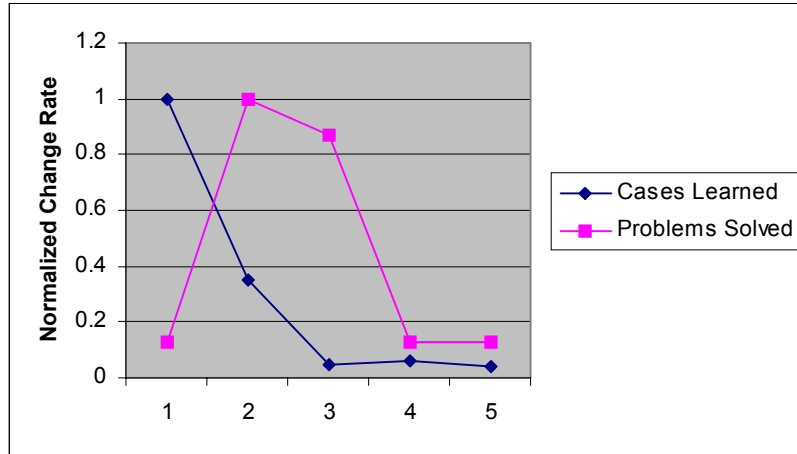


Figure 5 Normalized change rate for cases learned and problems solved

This illustrates the flexibility of our definition of cases; by making cases consist of one-level decompositions rather than complete plan and by generalizing them, the opportunities for re-usability of the cases are greatly improved.

Our experimental setup was made under the following ideal conditions that may not occur in real-world problems:

- Project plans entered by the automatic user were provable correct. This is the result of using SHOP, which always generates correct plans. In addition, our specialized extraction procedure extracts conditions from HTN plans and enters them as resources in Microsoft Project. In real-world situations users make errors and feed the system with incomplete information. Moreover, explicit criteria might not even exist to determine if a plan is valid or not.
- Each of the plans obtained by SHOP/CCBR with the augmented knowledge base correctly model the target domain. That is, each of these plans could be generated by SHOP using UMTranslog. Again in real-world situations it might not be possible to determine such correctness or as discussed previously, captured cases might not even correctly model the target domain.
- Related to both of the above points, operators were available that indicated the effects of performing the activities (i.e., the primitive tasks). Such effects are needed to ensure correctness of the plans generated by SHOP and SHOP/CCBR. Again, in many real-world situations it is simply not possible to state such operators.

The purpose of these ideal conditions was to allow us to measure the increase in coverage of the knowledge base without any noise in the data.

Conclusions and Future Work

In this paper we present an alternative approach to case capture. Instead of developing a specialized case capture interface, we capture cases by extracting information about the project plans that are created by the user. The key condition for our approach is that the case representation formalism, HTNs, has a one-to-one correspondence with the WBS. We developed the caseAcquisition algorithm that traverses the WBS, creating a case from each one-level decomposition. Conditions about the applicability of the case are extracted by observing the resources associated with the task being decomposed.

We implemented our ideas about case extraction on Microsoft Project and added a task decomposer system, SHOP/CCBR. An important point is that SHOP/CCBR does not require methods and/or operators to be available to generate task decompositions. We performed an experiment under ideal laboratory conditions, namely, starting plans in Microsoft Project being provable correct HTN plans, having operators and methods available in SHOP/CCBR that allow to test the correctness of the plans generated by the augmented knowledge base (consisting of operators, methods and captured cases). We observed the rapid increment in coverage of the augmented knowledge base while there is a rapid decrement of the number of cases learned. We discussed that the rapid increment was due to our strategy of storing one-level decompositions in the cases instead of multi-level decompositions and our case generalization procedure.

We want to explore case base maintenance issues in the near future. In our current implementation, a first step in this direction is made; the storeCheck procedure checks that no candidate case is added to the knowledge base when either a method or a case exists, that subsumes the candidate case. Despite this, redundancy can easily be generated. Another related issue that we want to address is the case over-generalization of the Gen+ function.

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.
- Blythe, J., Kim, J., Ramachandran, S., and Gil, Y. An Integrated Environment for Knowledge Acquisition. In: *Proceedings of the International Conference on Intelligent User Interfaces 2001*.
- El Fattah, Y. Structured Representation of Causal Relationships in Template Planning. (A Preliminary Report). Rockwell Scientific Technical Report, 2001.

- 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
- Hanney, K., and Keane, M.T Learning Adaptation Rules from a Case-Base. In: *Proceedings of the Third European Workshop on Case-Based Reasoning (EWCBR-96)*, Lausanne, Switzerland: Springer. 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.
- Muñoz-Avila, H., McFarlane, D., Aha, D.W., Ballas, J., Breslow, L.A., & Nau, D. Using guidelines to constrain interactive case-based HTN planning. In: *Proceedings of the Third International Conference on Case-Based Reasoning (ICCB-99)*. Munich: Springer, 1999.
- 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.
- 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.
-