Paper at EWCBR-94

Table of Contents

Integrating Induction in a Case-based Reasoner

Eva Armengol & Enric Plaza

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Scientific Research Council
Campus de la Universitat Autónoma de Barcelona
08193- Bellaterra, Catalonia, Spain
{eva | enric}@iiia.csic.es

Abstract. This paper focuses on two key issues in building case-based reasoners (CBRs). The first issue is the knowledge engineering phase needed for CBRs as well as knowledge-based systems (KBS); the second issue is the integration of different methods of learning into CBRs. We show that we can use a knowledge modelling framework for the description and implementation of CBR systems; in particular we show how we used it in developing a CBR in the domain of protein purification. In order to encompass CBR (and learning in general) our knowledge modelling framework extends the usual frameworks with the notion of memory. Including memory we provide the capability for storing and retrieving episodes of problem solving, the basis of case-based reasoning and learning. We show here that this framework, and the supporting language NOOS, allows furthermore to integrate other learning methods as needed. Specifically, we show how a method for the induction of class prototypes can be implemented and integrated with case-based methods in a uniform framework.

1 Introduction

This paper focuses on two key issues in building case-based reasoners (CBRs). The first issue is the knowledge engineering phase needed for CBRs as well as knowledge-based systems (KBS); the second issue is the integration of different methods of learning into CBRs. Regarding knowledge engineering, the last years have shown that knowledge modelling frameworks are adequate methodologies for building KBS (cifra (Steels, 1990) and (Wielinga, 1992)).We show that we can use a knowledge modelling framework for the description and implementation of CBR systems; in particular we show how we used it in developing a CBR in the domain of protein purification. In order to encompass CBR (and learning in general) our knowledge modelling framework extends the usual frameworks with the notion of memory. Including memory we provide the capability for storing and retrieving episodes of problem solving, the basis of case-based reasoning and learning (Arcos and Plaza, 1993). We show here that this framework, and the supporting language NOOS, allows furthermore to integrate other learning methods as needed. Specifically, we show how a method for the induction of class prototypes can be implemented and integrated with case-based methods in a uniform framework. The structure of the paper is the following. First, we describe our knowledge modelling framework and its supporting object-oriented frame-based language named NOOS. Section 3 shows how a case-based reasoner on the domain of chromatography applied to protein purification can be developed in the framework and implemented using the NOOS language. Finally, related work and conclusions are outlined.

2 The Knowledge Modelling Framework

Our knowledge modelling framework augments the ideas of the components of expertise (Steels, 1990) with the notion of episodic memory: the memorization of problem-solving episodes allows learning methods to be integrated since they require to access the past experience to improve the system performance. The elements of our knowledge modelling framework are tasks, methods, theories and case models. Tasks are goals to be achieved by the system in a problem setting. Problem solving methods are specifications of ways to achieve tasks. Usually, tasks are decomposed into subtasks by means of a problem-solving method, e.g. the `generate-and-test` method applied to a task decomposes it into the `generate` and `test` subtasks. Other methods are elementary methods (like union and intersection of sets). Case models and theories embody the domain knowledge modelled for our problem. A case model contains all the factual knowledge of a topic and consists on the set of tasks that make sense for it; for instance, the case model of John are those tasks we have solved about John (like his fever being 39) and those tasks to be solved (like finding his diagnosis). Thus solving a problem consists on completing the case model of the problem (e. g. finding the diagnosis of John) by means of a method (e.g. `generate-and-test`). As the case model is characterized by its tasks, a theory is defined by the methods declared usable to solve the tasks of a case model.

How can we integrate case-based reasoning and learning into this framework? The first step involves the memorization of successful methods and the fact that all solved case models are also memorized as cases to be used by a case-based method. The second step involves a metalevel (called *inference level*) consisting of inference methods and inference theories. An inference method (e.g. the CBR-method) is invoked when a domain method is missing. The result is retrieving and selecting a method from a past case able to solve the task at hand. This method is then instantiated into the current task and executed there (like in derivational replay). In fact, several methods can be retrieved from different cases in memory and tried out to see whether one can solve the current task. Different CBR methods can be described uniformly in a `retrieve/select/reflect` decomposition by using different methods in the `retrieve` and `select` subtasks; these different methods embody the domain-dependent knowledge that may be used in case-based reasoning (Arcos and Plaza, 1993). Inference theories hold several inference methods plus some preferences to choose among them (they are like metafunctions one level up). Examples of this are shown in section 3. Other kinds of inference methods are inheritance that retrieve domain methods from super type theories (e.g. John may inherit domain knowledge from theory `person` and theory `mammal`), see (Plaza, 1992).

2.1 The Framework Implementation: The NOOS Language

The knowledge modelling framework described above is supported by the implementation of the object-oriented frame-based language NOOS in which every concept is represented as an object. The dynamics of NOOS is impasse-driven: every time a lack of knowledge to do something is detected, an impasse is generated and an opportunity for learning arises. As in SOAR (Newell, 1990), there are two kinds of impasses: either the next step is unknown (e.g. there is no known method to solve a task) or there are several ways to proceed (e.g. there are several possible methods that may be applicable to a task). Every time an impasse is generated the NOOS language generates a reflective task whose goal is to solve that impasse.

The results of solving an impasse in a task are stored into that task for future usage and future decisions may be based on this precedent (they are cases that record method utility). This is called the self-model of NOOS and can be used by NOOS learning methods to reason about its own performance and about method usage. In a uniform way, inference methods (that succeed or fail) are also recorded at the inference level task so the system may learn from their usage. Finally we mention two aspects of NOOS implementation: automatic backtracking and consistency maintenance. It is possible to declare in a metafunction several methods applicable to a specific task. NOOS assures that a method will be tried out to see if it solves the task (selected

according to the declared preferences), backtracking to other declared methods if it fails. When all methods for a task fail, and this task is a subtask generated by a method of a super ordinate task, that method fails and a higher-level backtracking to new possible methods is performed. Thus, backtracking in NOOS assures that all methods declared in all tasks will be explored until a solution to the overall task is achieved, and that methods will be selected according to the preferences expressed in metafunctions. Moreover, NOOS has a consistency maintenance mechanism that supports incremental knowledge modelling. Every time some NOOS object (theory, method, task, etc.) is changed the consistency maintenance mechanism invalidates all objects that depend on that change.

3 The Chromatography Domain Application

In this section we will explain an application of the knowledge modelling framework and the support provided by the NOOS language to the integration of several case-based and inductive methods. First we describe the application domain of protein purification using chromatographic techniques. After that, CHROMA, an application program written in NOOS, is explained in detail. In particular, the different tasks and methods of CHROMA, and their integration, are described.

3.1 Introduction to the Chromatography Domain

Our domain of application is the purification of proteins from biological sources (animal tissues, bacteriological cultures, etc.) for industrial and research purposes. Purification is an essential process in the analysis of the properties of molecules from biological origin and widely used in industries and in research. Proteins are a type of biological macromolecules that are purified by a sequence of laboratory operations. These operations can use different techniques but the more used for molecule purification are chromatographic techniques. They exploit the different distribution of molecules between both a stationary and a moving phase. There are different chromatographic techniques according to the physic-chemical principle in which are based: Ion Exchange is based in the Coulomb's law; Hydrophobic Interaction is based in the Van der Vaals law; Gel Filtration separates the molecules according its size; Affinity exploits the existence of specific unions between certain types of molecules; etc. A plan to purify a molecule can be composed by several steps involving one o more chromatographic techniques. There is no unique way to purify a molecule but there are multiple useful purification plans according the expected use of the purified molecule. To find the adequate purification plan the experience of an expert is required. Usually, a human expert makes a focused search in the literature in order to obtain adequate precedent cases of purification and then he analyzes them to choose the most appropriate. From a blind search it should obtain many experiments and probably an apprentice cannot choose between them. The literature search has to be focused according the molecule to purify, the sample origin (species, tissue, etc.), the future study with the purified molecule, etc. It is necessary much domain knowledge in order to decide how to do the search and how to choose between the obtained precedents. This domain knowledge is complex and difficult to formalize. Let's suppose that we have proteins classified according to the substance to which they have affinity and the protein to purify belongs to one of these classes. In that situation, the protein may be purified using an affinity chromatography technique with the substance of the specified class. Nevertheless, this classification is not useful if the protein has no affinity with any substance or to decide if other steps can be applied.

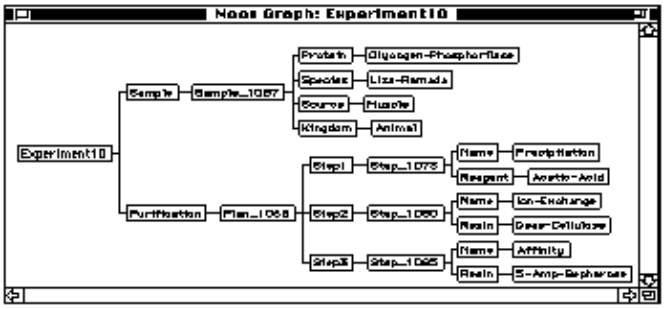

Fig. 1. Description of an experiment from case-base in CHROMA application. An experiment is composed of the sample from which the protein has to be extracted and the purification plan.

3.2 Solving the Purification task

From a knowledge engineering phase we detected that searching a case-base of purification experiments is an essential part in the human expert solving the purification task. Our goal is to build a system that, based on purification cases, will be capable to find precedent cases useful for solving new experiments. This system has to capture enough domain knowledge to allow it to focus the case-base search in an efficient and expert-comparable level. We limit the action field to the biological macro-molecules called proteins and exclude macro-molecules as the nucleic acids or the polysaccharides. A requirement that emerged in the knowledge engineering phase was that the user of the system has to have the final decision about which plan is finally chosen. This requirement arises from the fact that the user is knowledgeable of the chemical domain and wants to maintain control on the purification process. The CHROMA application supports the user in inspecting the candidate cases proposed for taking his final decision.

The main task of the CHROMA application is the purification task. Given a new experiment, CHROMA has to determine a purification plan using either a memory of cases or domain knowledge. The memory of cases is composed by objects named *experiments* having two slots: a sample (an animal tissue, bacteriological culture, etc.) and a purification (a plan to purify a specific protein from the sample). Figure 1 shows an experiment in the case-base. A sample is described in turn by four features: the protein to purify, the species where the sample comes from, the source of the sample (an animal or vegetal tissue, a culture, etc.) and finally, the kingdom to which the species belongs (animal, bacteria, protozoa, etc.). The purification is a plan composed by a variable number of chromatography steps. Each step has the name of the technique (affinity, gel filtration, etc.) and the name of the substance (reagent or resin) used to purify the protein. A new experiment to purify has only the sample and the task is to find an appropriate purification plan for it.

In order to obtain a purification plan, `purification` task can use four methods: `equal-sample`, `classify-by-prototype`, `analogy-by-determination` and `default-plan` (see figure 2). The `equal-case` method detects if there is an experiment in the base of cases having the same sample as our current experiment. The `classify-by-prototype` method uses domain knowledge in the form of generalized experiments (prototypes) that are generated from the case-base of experiments by an inductive method. The `analogy-by-determination` method retrieves experiments purifying the same protein as our current experiment but from different species or source. Finally, `default-plan` is a domain method based on statistical analysis of purification experiments. This method is used when there is no experiment in the case-base purifying the protein we are interested in and the other methods have failed. In the next sections we will explain the four methods in detail and their interaction.

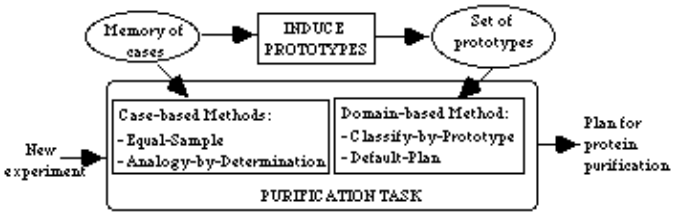

Fig. 2. Methods used in the `purification` task of CHROMA.

3.2.1 Equal-Sample Method

The `equal-sample` method is a case-based method constructed as a NOOS inference method for `purification` task. It is composed by two tasks (figure 3): `retrieve` and `reflect`. The method for `retrieve` task is `retrieve-by-pattern` that is a NOOS built-in method that considers the pattern as a graph and retrieves from the memory those cases (considered as graphs) that are subsumed by the pattern;

subsumption is defined in NOOS as the usual subsumption between directed acyclic graphs. In this situation, this method takes the *sample* of the current case as a pattern and searches the base of cases for experiments having at least that sample. In practice it will retrieve experiments with an identical sample since both pattern and experiments are graphs with the same links: protein, species, source and kingdom. `Reflect` task instantiates as solution to `purification` task the purification plan of the retrieved experiment. This method fails if there is no experiment having the same sample that the new one. This method is useful to solve routine purifications with commonly occurring samples and proteins and assures a correct solution.

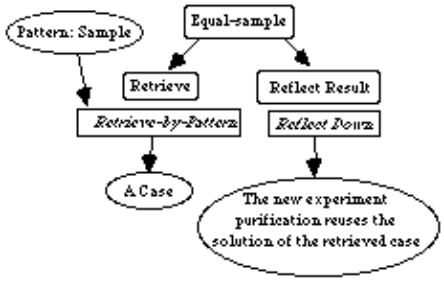

Fig. 3. Task-method decomposition of Equal-Sample method.

3.2.2 Analogy-by-Determination Method

The `analogy-by-determination` method is a case-based method constructed as a NOOS inference method for `purification` task . This method uses the domain knowledge embodied in a *determination* stating that the correct plan for purification task is determined by the protein to be purified. Determinations are functional dependencies that can be used to justify analogical reasoning (Russell, 1990)[1] . In the `analogy-by-determination` method the determination used states that the solution for `purification` task depends on the value of `protein` task. The method can then justify a solution purification plan from the fact that a precedent case with the same protein used that plan.

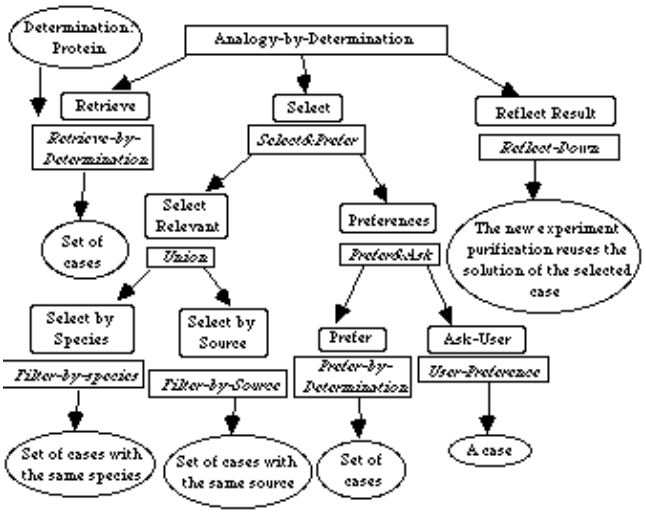

Fig. 4. Task-method decomposition of Analogy-by-determination method.

The `analogy-by-determination` method is composed by three tasks (figure 4): `retrieve`, `select` and `reflect`. The `retrieve` task uses a method named `retrieve-by-determination` that searches into the base of cases for those experiments purifying the same protein than the current problem. When it retrieves more than one experiment, the task `select` uses the method `select-&-prefer` in

order to select only one experiment. The `select-&-prefer` method has two subtasks: `select-relevant` and `preferences`. From the set of retrieved experiments, `select-relevant`'s goal is to select those experiments with a sample that has either the same species or the same source as the new experiment. The task `preferences` starts when there is more than one experiment purifying the same protein with the same species or source. If there are some experiments purifying the protein in the same species or source, `preferences` task has no effect because it is sure that the retrieved experiments belong to the same kingdom. Otherwise, experiments purifying proteins in species of the same kingdom are preferred. If more than one experiment is obtained all them are presented to the user who must to choose one of them. Finally, the `reflect` task instantiates as solution to the current experiment the purification plan of the selected case. The `analogy-by-determination` method fails if there is no experiment in the case base purifying the protein we want in our current experiment (i.e., it fails when there is no case fulfilling the protein determination).

3.2.3 Classify-by-Prototype Method

The `classify-by-prototype` method is a domain method that uses domain knowledge to classifies a sample in terms of a solution purification plan. In the Chromatography domain there is knowledge (molecular structure, source of the sample, etc.) that is implicitly used for the experts in order to search a good purification plan. We use the induction as an attempt to capture this implicit knowledge. The domain knowledge in the CHROMA application has the form of a set of prototypes (i.e. generalizations) which are created by an inductive method from the case-base of experiments. The `classify-by-prototype` method can be viewed as a case-based method that works with generalized cases (prototypes). As can be seen in Figure 5, a prototype is like an experiment with several values allowed for each slot in the sample.

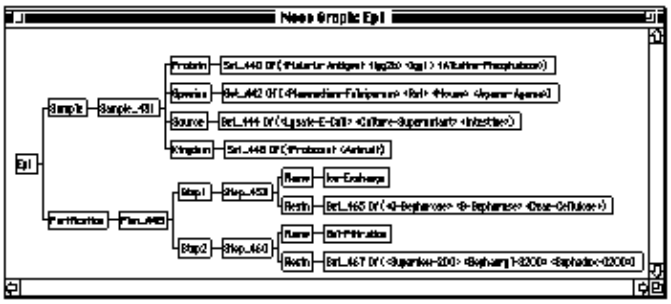

Fig. 5. A purification prototype induced from the base of cases.

The `classify-by-prototype` method is composed by four tasks (figure 6): `generate-prototypes, plausible-prototypes`, `select` and `reflect`. The `generate-prototypes` task accesses the domain knowledge required for the `classify-by-prototype` method in the form of purification prototypes. If this knowledge does not exist then the NOOS language generates an impasse that is solved using the `induce-prototypes` method (see next section).

Once retrieved all the purification prototypes the `plausible-prototypes` task selects, using `subsumption-matching` method, only those prototypes subsuming the new experiment sample. We saw in 3.2.1 the `retrieve-by-pattern` method that uses this same subsumption method to retrieve cases from memory. Subsumption in an object-oriented language like NOOS is similar to pattern-matching in rule-based systems[2] . An object A is subsumed by B, A < B when :

$\{F\}_A < \{F\}_B$,

for all F in $\{F\}_A$, F(A)=V1 , F(B)=V2 => V1 < V2

i.e., for all slots $\{F\}_A$ in A, B has all those slots ($\{F\}_A < \{F\}_B$) and the values of the slots in A (F(A)=V1) are subsumed by the corresponding values of the slots in B (F(B)=V2 and V1 < V2). For instance the following `experiment-3` is subsumed by the prototype `Ep1` shown in Figure 5.

```
(define experiment-3
    (sample (define (sample)
                (protein alkaline-phosphatase)
                (species rat)
                (source intestine)
                (kingdom animal))))
```

When more than one purification prototype is retrieved, the `select` task has the `user-preference` method that asks the user for the best purification plan: as in the previous method the final decision has to be the one of the user and NOOS supports the inspection of the alternatives. As in all inference methods, finally the `reflect` task instantiates in the current experiment the purification plan of the prototype. The `classify-by-prototype` method fails when there is no prototype subsuming the current experiment sample. In this situation the CHROMA application tries the `analogy-by-determination` method to search a case for solving the `purification` task.

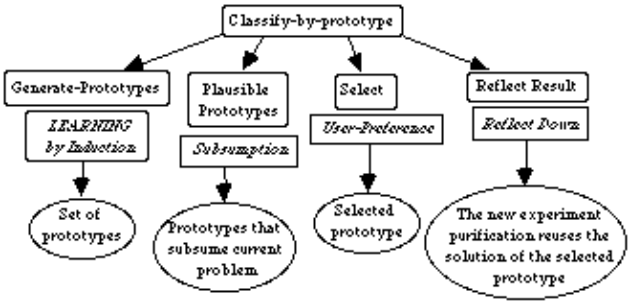

Fig. 6. Task-method decomposition of Classify-by-prototype method.

3.2.4 Induce-Prototype Method

As we have seen, the domain knowledge contains purification prototypes obtained by induction from the base of cases. These prototypes represent classes of experiments that have the same purification plan. Thus a prototype is like a rule of the type

situation => *plan*

where *situation* describes the general features that the *sample* of an experiment has to fulfil to be classified in the solution class that is the *plan*. The subsumption method we have seen acts as a pattern-matcher that compares a specific experiment to the generalized experiment (prototype); if they match, the classification recommends the prototype's purification plan as the plan for the current experiment. A purification prototype has the same slots that an experiment. The difference is that each sample slot of a purification prototype can have more than one value (see Fig. 5). The advantage provided by these purification prototypes is that sometimes a case retrieval method may propose to the user different experiments to choose but all they have the same purification. Using the purification prototypes, that summarize this information into one single (generalized) case, CHROMA can produce results without the user's intervention.

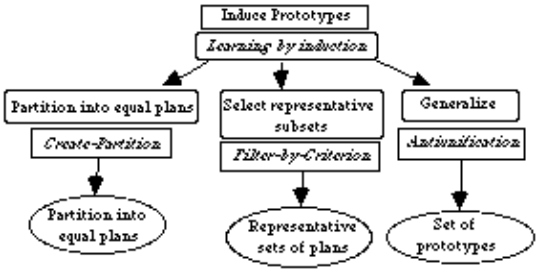

Fig. 7 Task-method decomposition of Induce-Prototypes method.

The `induce-prototype` method is activated by the impasse resulting from task `Retrieve-Prototypes` searching for the purification prototypes not yet computed. This method is composed by three subtasks (figure 7): `partition-into-equal-plans`, `select-representative-sets` and `generate-prototype-from-set`. The method of `partition-into-equal-plans` task divides the base of cases into sets containing experiments with the same purification plan. Because some of the formed sets may have few elements and it is not desirable to make induction, `select-representative-sets` obtains only those sets having a number of elements above a threshold. The method of `generate-prototype-from-set` task generates from each representative set a purification prototype. Its main concern is to generalize the samples of the experiments belonging to a representative set. The sample of a prototype is created with its slots having the union of the values of each slot of the experiment samples belonging to a representative set. The prototype purification plan is the common purification plan of all the experiments into that representative set.

3.2.5 Default-Plan Method

The `default-plan` method is a domain-based method that starts if all the other methods have failed. This occurs only when the protein of the current experiment has never been purified before and thus no case with that protein purification exists in memory. This default purification plan has been obtained from a statistical study where a lot of purification plans were analyzed. The conclusion of this study was that the most frequently used chromatographic technique in each step of a purification plan are the following: 1) Clarification, 2) Ion-Exchange, and 3) Gel-Filtration. As we said in section 3.1 there are several plans that can be used to purify a protein; although some plans are more adequate (more efficient or with a better degree of purity in the obtained protein) , when there is no literature on a particular protein subject, the expert uses general or default plans, less adequate but nonetheless appropriate for purification. The `default-plan` method captures this general domain knowledge useful for the CHROMA application.

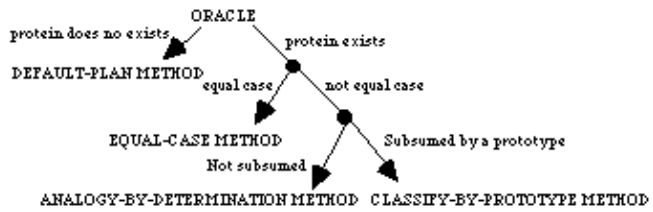

Fig. 8. Description of the Oracle method.

3.3 Integration

An advantage of the flexible integration provided by NOOS is that we may program different methods and experimentally evaluate the quality of their results, their efficiency, etc and then combine them in an efficient

way. We can evaluate our methods regarding their solution quality and their efficiency. The quality of a purification plan depends on the quantity of protein produced (some chromatographic techniques produce more and some less from a given source) and the degree of purity of the protein produced (it also depends on the chromatographic techniques). We will not discuss here quality of solution issues since this is a subjective and not numeric assessment of the domain expert.

Regarding efficiency, in some situations the use of several problem solving methods may produce a more or less efficient solution than using a single method. The sequential preference on methods turns out to be an inefficient combination since the failed methods increase the mean time used in solving a problem. Ideally, we would like to have some meta-knowledge that determines which of the possible methods is best to solve a specific problem, and try only that method, instead of trying them in a sequential order. Let us call this selection meta-knowledge an *oracle*. In the CHROMA application, all problems can be solved using together `default-plan` method (M*d*) and `analogy-by-determination` method (M*c*). Given a number Q*t* of new cases to be solved, and assuming that the *oracle* chooses one of them correctly, the necessary time to obtain a solution for all the cases is the following:

$$T1 = Qd * Td + Qc * Tc + D * Qt \text{ where } Qt = Qc' + Qd$$

T*d* and T*c* are the mean execution time of M*d* and M*c* respectively, Q*d* and Q*c'* are the number of cases solved using M*d* and M*c* respectively, and *D* is the execution time of the *oracle*. `Classify-by-prototype` method (M*p*) can be used over a subset of cases that can also be solved using `analogy-by-determination` method, but its mean execution time (T*p*) is more efficient (T*c* >= T*p*). Using the three methods M*p*, M*d* and M*c* together the mean execution time if the *oracle* chooses correctly among them is the following:

$$T2 = Qd * Td + Qc * Tc + Qp' * Tp + D * Qt \text{ where } Qc' = Qc + Qp$$

Using these three methods instead of only two makes sense only if this combination is more efficient (i.e. T*1* >= T2). In fact this only requires that T*c* >= T*p*, and this is true in our application. However this is true assuming the oracle's meta-knowledge is perfect, i.e. it always chooses the right method. In a real situation, the *oracle* never fails when it chooses `default-plan` method[3] but it is not perfect when it has to choose between `classify-by-prototype` (M*p*) method and `analogy-by-determination` (M*c*) method. Let be Q*f* the number of cases over which the oracle chooses incorrectly between M*p* method and M*c* method, i.e. the *oracle* selects M*p* method but it fails. The necessary time to evaluate these cases is (T*p* + T*c*) * Q*f* , since over these Q*f* cases two methods will be applied: first the M*p* method (that fails) and then the M*c* method. Now the necessary time to evaluate Q*t* new cases is the following:

$$T3 = Qd * Td + (Qc + Qf) * Tc + Qp' * Tp + D * Qt \text{ where } Qp' = Qf + Qp$$

In this situation, the three-method combination is more efficient than the two-method combination if the condition T*1* >= T*3* holds. From this condition it can be obtained the following expression:

$$\frac{T_c - T_p}{T_p} \geq \frac{Q_f}{Q_p}$$

The efficiency of the three-method combination does not depend on the *oracle* time but on the number Q*p* of cases that can be solved using M*p* and the number of cases over which the oracle fails (Q*f*). In our domain T*p* = 20,937 seconds and T*c* = 27, 175 seconds; resulting the condition 0,298 Q*p* >= Q*f* . In particular, if we have 100 new cases that can be solved using M*p* less than a 23% of error in the decision of the oracle for the three method combination is acceptable in order to be more efficient.

We have shown how CHROMA can use an oracle to select a method. The oracle is easy to represent in the NOOS language because it is just a method solving the task of choosing among possible methods. This form of method combination is more efficient than the simple sequential combination in the chromatography domain. On the long run it would be interesting to learn to select the best methods based on features of the current problem and of the task environment. The important issue is that an integrated framework as NOOS allows us to freely implement, combine and experimentally evaluate the quality and efficiency of methods and method-combinations and tailor the final system to the task requirements in an application domain.

4 Related Work

We have presented a knowledge modelling framework and its supporting language, NOOS, that incorporate the notion of memory and impasse-driven learning for integrating case-based and inductive methods for learning. Other knowledge modelling frameworks, like components of expertise (Steels, 1990), KADS (Wielinga, 1992), PROTEGE-II (Puerta et al., 1991), etc, do not integrate learning methods. Our approach is closer to components of expertise. However, components of expertise lacks the two layers of NOOS: domain theories and methods and inference theories and methods, having only the domain layer. The basic building blocks in PROTEGE-II, are implemented in Lisp and thus new mechanisms require new programs in Lisp. The philosophy of NOOS is different: methods can be decomposed in a finer grain into elementary subtasks that use a set of elementary methods (e.g. conditionals, set intersection, etc.) provided by NOOS. There is a difference between our goals and the goals of COMMET, KADS, PROTEGE-II, etc., but they are complementary. The knowledge modelling community is interested in acquiring a wide library of components, while we are interested in integrating learning methods with those libraries of components.

There is some work dealing specifically with the combination of case-based methods and knowledge produced by inductive methods. The KATE system (Manago, 1989) induces a decision tree and combines decision-tree classification with case-based classification. The combination of methods is fixed: decision-tree classification is tried first and if it "fails" (because of a missing attribute) then a case-based method is used. The INRECA project (Manago, 93) follows a similar approach integrating the induction of decision trees (using KATE) and case-based reasoning using PATDEX . Currently KATE and PATDEX are able to interchange results via the format of the CASUEL language, and their combination is again fixed. The integration of the multiple inference methods is easily realized by the support given by NOOS language to the knowledge modelling of the CHROMA application. As we have seen all methods (case-based, inductive, knowledge-based) can be described in our knowledge modelling framework and then implemented using the NOOS language. This is a tighter integration than other proposals for integrating inductive and case-based methods; for instance the INRECA project is based on the establishment of the syntax of an interchange format called CASUEL. Different modules (CBR, induction) read from and write to this format but each module uses a different representation language.

5 Conclusions and future work

In this paper, we have presented a knowledge modelling framework for KBS modelling and its implementation into the NOOS language. A more detailed explanation of NOOS and its case-based capabilities can be found in (Arcos and Plaza, 1993). We have focused on the integration of case-based and inductive methods in our knowledge modelling framework. The basis for this integration is the capability of NOOS for ascribing several alternative methods to any task, selecting among them based on a language of preferences, and automatically backtracking to other alternative methods. Learning methods, case-based or inductive, are integrated in an impasse-driven manner. This general setting allows us to ascribe appropriate learning methods to specific tasks, as shown in the CHROMA application. The multiplicity of methods raises the issue of how to combine them in an efficient way. We have shown that meta-knowledge about their applicability range can be incorporated in the form of an oracle that selects the most adequate method for every case to be solved. Other learning methods, like induction of decision trees, could be integrated in our framework if it is useful to solve specific tasks.

Our current work involves several systems where CBR methods are integrated with other methods. For instance, a classification-based system for sponge identification is being developed in NOOS using a method of top-down classification called `recognize-prototype-and-refine`. This work is a test for comparing our approach to a classical rule-based expert system on the same domain developed at our Institute and may help us understand the differences and the utility of our knowledge modelling framework for KBS design and the utility of learning methods for different subtasks. Moreover, research on planning as a general form of problem solving is currently under way. We focus on case-based planning, non-linear search-based planning, and interleaving case-based/search-based planning. Comparing it to other systems, like derivational analogy (Veloso, 1992) NOOS supports its two main capabilities: non-linear planning and interleaving planning and learning. Comparison of planning domains in KBS knowledge modelling frameworks will be pursued, specifically using the workbench project Sisyphus on floor planning, used as a standard for comparison by the KBS knowledge acquisition community.

Acknowledgements

The research reported on this paper has been developed at the IIIA inside the ANALOG Project funded by CICYT grant 122/93 and a MEC fellowship. We would like to thank Dr. Lluis Bonamusa for kindly providing his expertise on the Chromatography domain.

References

Arcos, J. L., and Plaza, E. (1993). A Reflective Architecture for Integrated Memory-based Learning and Reasoning. In *First European Workshop on Case-based reasoning*. Kaiserslautern: Germany.

Napoli, A. (1992). Subsumption and Classification-based Reasoning in Object-based Representations. *Proceedings of the European Conference on Artificial Intelligence (ECAI'92), Vienna, Austria*, p.425-429.

Manago, M. (1989). Knowledge Intensive Induction, *Proceedings 6th International Machine Learning Workshop*. Morgan Kaufman.

Manago, M., Alhoff, K. D., Auriol, E., Traphöner, R., Wess, S., Conruyt, N., Maurer, F. (1993). Induction and reasoning from cases. *First European Workshop on Case-based reasoning*. Kaiserslautern: Germany..

Newell, A. (1990). *Unified Theories of Cognition.* Cambridge MA: Harvard University Press

Plaza, E. (1992). Reflection for analogy: Inference-level reflection in an architecture for analogical reasoning. *Proceedings of IMSA'92 Workshop on Reflection and Metalevel Architectures*, Tokyo, November 1992, p. 166-171.

Puerta, A., Egar, J., Tu, S., Musen, M. A. (1991). A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools. In *Proceedings of AAAI-KAW*.

Russell, S. (1990). *The Use of Knowledge in Analogy and Induction*. Morgan Kaufmann.

Steels, L. (1990). The Components of Expertise, *AI Magazine*, 11(2):30-49.

Veloso, M., (1992). *Learning by analogical reasoning in general problem solving*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Wielinga, B., Schreiber, A., Breuker, J. (1992). KADS: A modelling approach to knowledge engineering. Knowledge Acquisition 4(1).