CASE-BASED REASONING

Bridging the Lesson Distribution Gap

David W. Aha[1], Rosina Weber[21], Héctor Muñoz-Avila[31], Leonard A. Breslow[1], Kalyan Moy Gupta[4]
[1]Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Code 5515
4555 Overlook Avenue, SW, Washington, DC 20375-5337, USA
[2]Department of Computer Science, University of Wyoming, Laramie, WY 82071-3682
[3]Department of Computer Science, University of Maryland, College Park, MD 20742-3255
[4]IIT Industries, AES Division, Alexandria, VA 22303
surname@aic.nrl.navy.mil

Abstract

Many organizations employ lessons learned (LL) processes to collect, analyze, store, and distribute, validated experiential knowledge (lessons) of their members that, when reused, can substantially improve organizational decision processes. Unfortunately, deployed LL systems do not facilitate lesson reuse and fail to bring lessons to the attention of the users when and where they are needed and applicable (i.e., they fail to bridge the *lesson distribution gap*). Our approach for solving this problem, named *monitored distribution*, tightly integrates lesson distribution with these decision processes. We describe a case-based implementation of monitored distribution (ALDS) in a plan authoring tool suite (HICAP). We evaluate its utility in a simulated military planning domain. Our results show that monitored distribution can significantly improve plan evaluation measures for this domain.

1 Introduction

Verified experiential lessons teach improvements about a work practice [Fisher *et al*., 1998]. Many large government (e.g., DOD, DOE, NASA) and private organizations develop *lessons learned* (LL) systems to assist with the knowledge management process of collecting, analyzing, storing, distributing, and reusing lessons [Davenport and Prusak, 1998; Weber *et al*., 2001a]. Lessons record *tacit* experiential knowledge from an organization's employees whose knowledge might be lost when they leave the company, shift projects, retire, or otherwise become unavailable. It is often crucial to record lessons; lives are sometimes saved by preventing recorded catastrophes from recurring [DOE, 1999]. Thus, sharing lessons, even if they are used infrequently, can be *very* important. LL processes and systems are needed to assist with lesson sharing, which can be complicated, especially for large organizations or large lesson databases.

Lessons are usually in unstructured text format, and distribution is commonly supported using standalone text or keyword retrieval tools that require users to "pull" lessons from a repository. Unfortunately, problems with text representations and with this approach to distribution negatively affect lesson reuse, which results in widespread underutilization [Weber et al., 2001a]. In particular, they are responsible for what we term the *lesson distribution gap*. This gap exists when an organization fails to properly promote lesson reuse and available lessons are not deployed when and where they are needed and applicable.

At least three approaches exist to eliminate this gap. First, identified lessons can be incorporated directly into *doctrine*, which defines the processes to be employed by an organization's members. The doctrine is updated to include the knowledge contained in the lesson. For example, the Army's CALL Center [CALL, 2001] deploys teams of lesson analysts and doctrine experts to perform such updates. However, not all lessons can be incorporated into rule-like doctrine (e.g., because they may be true exceptions), and not all organizations have close working relations between doctrine and lessons learned personnel.

A second way to bridge this gap involves "pushing" lessons to potential users, such as via list servers (e.g., [SELLS, 2000]) or intelligent spiders. For example, two of the DOE's sites already employ portals containing spiders [SELLS, 2000]. However, spiders are not integrated with the decision support processes that the lessons target. Thus, after retrieving lessons with a spider, users must characterize the situations for which they are useful, recall them when they encounter an applicable decision support context, and interpret them correctly so that they are properly reused. These are challenging tasks, requiring a high level of expertise and time that most users do not have.

We investigate a third approach to bridging the lesson distribution gap that involves tightly integrating the lesson repository with a decision support tool. Our approach, detailed in Section 2, requires inserting a monitor into the decision support process so that it can determine when a

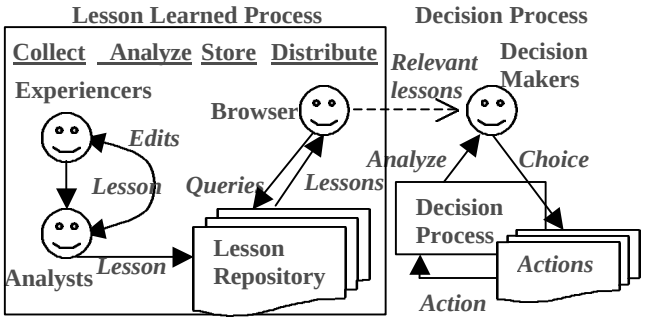


Figure 1. Most lessons learned processes are separated from the decision processes they support.

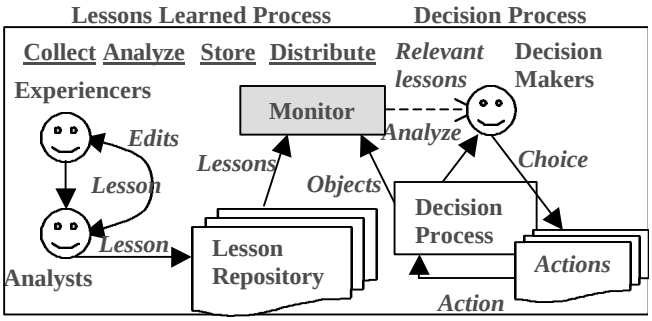


Figure 2. Monitored lesson distribution integrates the lessons learned process with lesson-targeted decision processes.

lesson's conditions are well matched by a decision context. In Section 3, we describe a case-based implementation of this *monitored distribution* approach in ALDS (Active Lesson Delivery System), a module of the HICAP plan authoring tool suite [Muñoz-Avila *et al*., 1999]. We describe ALDS's evaluation in Section 4, where we provide evidence that it can significantly improve performance measures for HICAP-generated plans for a simulated military planning domain (i.e., noncombatant evacuation operations (NEOs)).

Based on a recent survey [Weber *et al*., 2001a] and analysis of the AAAI'00 Workshop on Intelligent Lessons Learned Systems [Aha and Weber, 2000], we believe that monitored distribution is novel with respect to deployed LL systems, and has great potential for deployment. We discuss the implications of our findings and future research issues in Section 5.

2 Monitored Lessons Learned Processes

A **lesson** is a knowledge artifact that represents a validated (i.e., factually and technically correct) distillation of a person's experience, either positive or negative, that, if reused by others in their organization, could significantly improve a process in that organization. In particular, it identifies a specific design, process, or decision that reduces or eliminates the potential for failures and mishaps, or reinforces a positive result [Secchi *et al*., 1999]. The knowledge management process involving lessons (i.e., the **lessons learned process** (LLP)) implements strategies for collecting, analyzing, storing, distributing, and reusing a repository of lessons to continually support an organization's goals.

LLPs typically target decision-making or execution processes for various types of user groups (i.e., managerial, technical) and organizations (e.g., commercial, military). In this paper, we focus on managing lessons to support *planning* processes.

Flowcharts describing LLPs abound; organizations produce them to communicate how lessons are to be collected, analyzed, and distributed [SELLS, 2000; Fisher *et al*., 1998; Secchi, 1999]. Figure 1 displays a typical LLP, composed of the five sub-processes mentioned above, where reuse does not take place in the same environment as the other sub-processes.

Existing, deployed *LL systems* do not support all processes in a LLP. In particular, organizations typically do not develop software to support verification or reuse. Instead, they use electronic submission forms to facilitate lesson collection, and use a standalone retrieval tool for lesson distribution [Weber *et al*., 2001a]. Users interacting with this standalone tool are expected to browse the stored lessons, studying some that can assist them with their decision-making processes. However, based on our interviews and discussions with members of several LL organizations (e.g., in the Navy, Joint Warfighting Center, Department of Energy, and NASA), and many intended users, we found that they do not use available standalone LL systems, which are usually ineffective because (1) they force users to master a *separate* process from the one they are addressing, and (2) they impose the following unrealistic assumptions:

- Users are convinced that using an LL system is beneficial (e.g., contain relevant lessons).
- Users have the time and skills to successfully retrieve relevant lessons.
- Users can correctly interpret retrieved lessons and apply them successfully.
- Users are reminded of the potential utility of lessons when needed.

We believe that lessons should be shared when and where they are applicable, thus promoting their reuse. This motivated us to develop an architecture for proactive, integrated lesson distribution (Figure 2). In this *monitored distribution* approach, reuse occurs in the same environment as other sub-processes; the decision process and LLP are in the same context. This embedded architecture has the following characteristics/implications:

- The LLP interacts directly with the targeted decision-making processes, and users do not need to know that the LL module exists nor learn how to use it.
- Users perform or plan their decision-making process using a software tool.
- Lessons are brought to the user's attention by an embedded LL module in the decision-making environment of the user's decision support tool.

- A lesson is suggested to the user only if it is applicable to the user's current decision-making task and if its conditions are similar to the current conditions.
- The lesson may be applied automatically to the targeted process.

This process shifts the burden of lesson distribution from a user to the software, but requires an intelligent "monitoring" module to determine whether/when a lesson should be brought to a decision maker's attention.

3 Implementation

We implemented the monitored distribution process in ALDS, a module of HICAP [Muñoz-Avila *et al.*, 1999]. This section details HICAP and then ALDS.

3.1 Plan authoring using HICAP

HICAP (Hierarchical Interactive Case-based Architecture for Planning) is a multi-modal reasoning system that helps users to refine a planning hierarchy [Muñoz-Avila *et al.*, 1999]. A hierarchy is represented as a triple H = $\{T,\prec,:\}$, where T is a set of tasks, $\prec$ defines a (partial) ordering relation on T, and t_1:t_2 means that t_1 is a parent of t_2 in T. Task hierarchies are created in the context of a state **S**={<q,a>$^+$}, represented as a set of <question,answer> pairs.

HICAP provides three ways to refine tasks into subtasks. First, it supports manual task decomposition. Second, users can decompose a selected task using HICAP's *interactive* case retriever (NaCoDAE/HTN), which involves iteratively answering prompted questions that refer to state variables. Third, users can select a generative planner (SHOP) to *automatically* decompose *t*.

3.2 Monitored lesson distribution using ALDS

Planning tasks (e.g., for military operations) involve several decisions whose affect on plan performance variables (e.g., execution time) depends on a variety of state variables (e.g., available friendly forces). Without a complete domain theory, HICAP cannot be guaranteed to produce a correct plan for all possible states. However, obtaining a complete domain theory is often difficult, if not impossible. In addition to representing typical experiential knowledge, lessons can help fill gaps in a domain theory so that, when reused appropriately during planning, they can improve plan performance. This is the motivation for applying lessons while using HICAP.

Figure 3 summarizes the behavior of ALDS, the monitored distribution module. ALDS monitors task selections, decompositions, and state conditions to assess similarities between them and the stored lessons. When a stored lesson's applicable decision matches the current decision and its conditions are a good match with the current state, then the lesson is brought to the user's attention to influence decision-making. When a user implements a prompted lesson's task decomposition (i.e., reusing the lesson), the current task hierarchy is modified appropriately.

Abstractly, reusable lessons contain indexing and reuse components. Indexing components include the *target task* and the lesson's applicability *conditions*. The reuse components include a *suggestion* that defines how to reuse an experience and an *explanation* that records how the lesson was learned. This explanation can be used to justify the lesson's use in a new situation. In ALDS, a lesson is indexed by the (target) task that it can modify and a set of <question,answer> pairs defining its applicability conditions, and contains a suggestion (e.g., a task substitution) and the lesson's originating event (i.e., the explanation).

We use a case-based approach for lesson distribution primarily because the indexing components (i.e., task and conditions) must support a partial matching capability. Furthermore, the applicability of a lesson depends on the context of the task that it targets, which suggests using domain-specific similarity functions.

Thus, if both the task and the conditions are a "good" match to the current planning state, then the user should consider decomposing the current task into the lesson's suggested subtasks. We borrowed NaCoDAE/HTN's similarity function for cases, and used a thresholded version to define "good" (i.e., determine when a lesson should be prompted to a user).

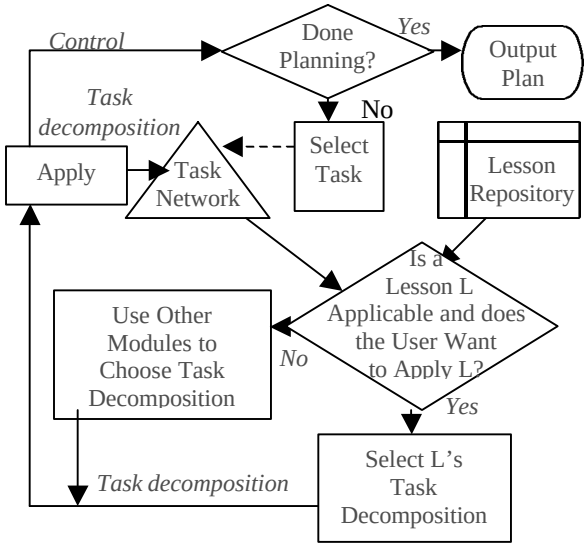


Figure 3. HICAP's lessons distribution sub-process, implemented in ALDS, during plan elaboration.

4 Evaluation

We wanted to evaluate the hypothesis that the monitored distribution approach (e.g., as implemented in ALDS) is superior to the traditional standalone approach for lesson distribution and promoting lesson reuse. For HICAP/ALDS, this hypothesis requires evaluating the plans created by operational users who use the two lesson distribution approaches in repeated planning tasks. Dependent variables

would include agreed-upon measures of plan quality, which depend on the planning domain.

Unfortunately, HICAP/ALDS has not yet been scheduled for testing in a military training exercise, which prevents us from working with operational planners. Therefore, we instead performed an evaluation using simulated users on a simulated NEO (noncombatant evacuation operations) domain. Sophisticated full-scale NEO simulators do not yet exist. Therefore, we constructed our own plan evaluator for a simulated NEO domain (Section 4.1). This allowed us to evaluate HICAP/ALDS's plan authoring and lesson distribution capability for an entire plan, rather than be limited to an evaluation on a single task decomposition task [Muñoz-Avila *et al.*, 1999].

Simulating how a user might benefit from a standalone lesson distribution tool is difficult. Therefore, we instead compared plan generation when using ALDS vs. not using it (Section 5.2), where our revised hypothesis is that *using lessons will improve plan quality*. This central hypothesis to LLPs, although simple, has ***not*** been previously investigated for lessons learned systems, and thus is appropriate for an initial evaluation focus.

4.1 Methodology

The plans authored by HICAP concerned performing a rescue mission where troops are grouped and move between an initial location (the assembly point) and the NEO site (where the evacuees are located), followed by evacuee re-location to a safe haven. 81 possible routes and 4 means of transportation were encoded. In addition, other conditions were determined during planning such as whether a communications expert was available and the method for processing evacuees. HICAP's plans had 18 steps, and its knowledge base included 6 operators, 22 methods, and 51 cases. We randomly selected 100 initial plan states (12 independent variables) and produced plans for each state with the simulated user interacting with HICAP. This user assigned, through task decomposition, an additional 18 variables (with from one to four values each) for each plan, which required HICAP an average of about 40 seconds to generate. The same set of initial states to produce plans in HICAP was used (to guide task decomposition) both with and without lessons. Each of the two sets of 100 plans (i.e., one set obtained using lessons, and the second set obtained without using lessons) authored by HICAP was input to the evaluator (Section 4.2). Due to the non-deterministic behavior of the evaluator, we executed each plan ten times.

The version of HICAP used in this paper is deterministic; given a state and a top-level goal (i.e., perform a NEO), it will always generate the same plan. A simulated user interacts with HICAP by choosing task decompositions to generate a plan, using the process shown in Figure 3. In NaCoDAE/HTN conversations, it always answers the top-ranking displayed question for which it has an answer, and it answered questions until either none remained unanswered or until one of the solutions exceeded a retrieval threshold, which we set to 50%.

We selected 11 lessons for our experiment, representing a subset of approximately 56 NEO-related lessons from the Active Navy lesson repository (containing 5120 lessons) from the November 2000 copy of the unclassified Navy Lessons Learned System. These were selected according to their relevance to NEOs and their clarity, so that we could recognize their relation to the plans authored using HICAP. For example, one lesson was defined as:

Task: Standard Medical Inventory
- Applicability Conditions: (<q,a> pairs)
- Is the medical inventory of standard size or is it standard minus 1/3? Yes
- Is the climate tropical? Yes

Suggestion: Add 1/3 to the medical inventory

4.2 Plan evaluation

We built a stochastic evaluator for NEO plans that take into account general knowledge of the NEO domain and computes the performance measures (described below). This evaluator is not a simulator because it does not use specific distributions for each type of event, but simply computes, according to a uniform distribution, what are the expected consequences of some choices in building a plan (i.e., the causal chain of events that are generated by these choices will influence each of the dependent variables differently). We built the evaluator and the HICAP knowledge base for mock NEOs based on available applicable lessons.

We defined plan quality based on official measures of NEOs, which are planning domain dependent. These measures are defined in the Universal Naval Task List [UNTL, 1996] under measures of performance suggested for Joint and Naval tasks. These measures primarily concern execution duration and casualty rates. To avoid a redundant evaluation, we have selected one measure for total duration of the operation, one for duration until evacuees receive medical assistance, and the percentage of casualties among evacuees, friendly forces, and enemies. These summarize the most important aspects suggested in the UNTL.

We defined bounds for variables based on actual NEOs. For example, we limited the percentage of casualties that occurred after a severe enemy attack takes place. Enemy attacks will only be possible in two planning segments (out of a total of five segments) and their likelihood increases when users choose land transportation and decreases when weather is troublesome. There is a small chance of a crash when helicopters are used that increases if the weather is not favorable; the resulting number of casualties is proportional to the number of passengers in each aircraft. A long planning segment flown by helicopter will have added the time and risks associated with in-flight refueling (e.g., [Siegel, 1991]).

4.3 Results

As summarized in Table 1, ALDS using lessons substantially improved the first four of five performance

variables. A brief examination of the results (i.e., the first run for each of the 100 plans), using a standard student's t test, revealed significant differences for both overall duration ($p<0.1$, $t=1.60$, df=99) and duration until medical assistance arrived ($p<0.1$, $t=1.39$). All lessons were used in generated plans, and an average of approximately three lessons were used per plan.

Table 1. Experimental results with the 100 plans.

	Without lessons	With lessons	% Reduction with lessons
mean duration	39h50	32h48	18
s.d.	16h51	16h12	-
mean duration until medical asst.	29h37	24h13	18
s.d.	11h13	10h26	-
mean % casualties:	11.48	8.69	24
to evacuees			
to friendly forces	9.41	6.57	30
to enemies	3.08	3.14	-2

The significance of an overall reduction of 24% in the percentage of casualties among evacuees was estimated in each plan based on the parameter *number of evacuees*, which was randomly set to *dozens*, *hundreds*, or *thousands*. Based on the number of evacuees selected for these simulated NEOs, using the lessons reduced the average number of casualties by **24**, from 100 to 76.

These results suggest that the monitored distribution approach can potentially generate better plans for realistic problem domains (e.g., planning for NEO operations). However, the experimental conditions were designed so that lessons were available for a reasonable percentage of the generated plans, and thus could be prompted to the simulated HICAP user so that, when applied, they could improve plan quality (with high probability). Nonetheless, we expect that similar improvements may yield benefits in plans for domains where safety issues and speed are paramount to success.

The capabilities of certain learning algorithms can be evaluated by varying dataset characteristics to determine when certain learning algorithms can be expected to perform well (e.g., [Aha, 1992]). Similarly, we plan to characterize the set of experimental conditions for which ALDS can use lessons to significantly improve plan evaluation performance measures.

5 Discussion

This paper proposes a technology (i.e., case-based reasoning) solution to part of a knowledge management (KM) problem (i.e., managing lessons learned). However, KM problems typically require challenging organizational dynamics issues, and these require precedence in the context of bridging the lesson distribution gap. Thus, monitored distribution can at most play only one part of a much larger solution.

Our evaluation of ALDS demonstrates how monitored distribution, when embedded in a decision-making (i.e., planning) process, can improve the results of that process. Although we used simulated users in our experiments to reduce human biases during the evaluation, we stress that this is a mixed-initiative approach, where humans interact with HICAP to generate plans. The unique aspect of ALDS is that it allows users to execute a lesson's suggestion (i.e., here, a task substitution), rather than limit them to simply browsing the suggestion.

HICAP's NaCoDAE/HTN module manipulates *cases* that represent task decompositions corresponding to either standard operating procedures or decompositions that were derived from decision making during training exercises and actual operations. In contrast, ALDS manipulates *lessons* that capture experiences that, if reused, can significantly improve the performance of subsequent plans. Unlike cases, lessons are not conceptually limited to representing task decompositions, but can be used to apply edits to *any* of HICAP's objects (e.g., resource assignments, resources, task durations).

Several workshops (e.g., organized by the Department of Energy, the European Space Agency, the Joint Warfighting Center, and each branch of the armed services) have now taken place on the topic of lessons learned. However, few efforts on lessons learned systems have examined the potential utility of AI (e.g., Vandeville and Shaikh [1999] briefly mention using fuzzy set theory to analyze elicited lessons), and there is a lack of closely related work to monitored distribution. However, one recent workshop brought attention to this area from an AI perspective [Aha and Weber, 2000], and a few of its contributors touched on issues related to proactive lesson distribution. For example, Leake *et al.*'s [2000] CALVIN system implements a task-oriented LLP that collects lessons about research topics and research results with an active distribution sub-process. Like ALDS, CALVIN prompts users with suggestions (i.e., alternative WWW pages to browse) that can be immediately executed. However, while CALVIN focuses on a diagnosis task, ALDS operates in the context of a synthesis task (i.e., planning), and can potentially update any of the planning scenario's objects. Like both of these systems, Watson [2000] also describes a case retrieval system, in this case for extending Cool Air to distribute trouble tickets. However, Cool Air does not operate in a mixed-initiative setting. Some KM approaches [Reimer *et al.*, 2000; Abecker *et al.*, 2000] also target distribution in the context of organizational knowledge, but use formats that do not support indexing.

Several limitations exist concerning our approach and its implementation in ALDS. For example, lessons can be complex, and suggest changes to a variety of objects in the planning scenario. Although HICAP represents several such objects (e.g., resources, resource assignments), it is currently limited to processing only task substitution lessons. In future implementations of HICAP and ALDS, lessons will be able to represent suggestions that, when applied, will not be limited to task substitution. For example, a lesson might suggest a task decomposition, or using an alternative resource assignment for a given task,

recommend changing some temporal orderings of tasks, or suggest edits to any of the objects used by HICAP to define plans.

In addition, to be useful, our approach assumes that the decision processes targeted by the lessons are managed by a software tool, thus allowing integration with ALDS. Furthermore, our approach requires identifying each lesson's indexing and reuse components, which requires significant knowledge engineering effort. We are currently developing lesson collection tools that reduce this effort. Weber *et al.* [2001b] describe interactive elicitation approaches that use taxonomies to guide lesson collection to populate ALDS's lesson repository.

In future work, we will conduct subject experiments that compare the monitored distribution approach vs. traditional keyword search tools for lesson distribution. We will also demonstrate how monitored distribution is not restricted to planning tasks.

6 Summary

We identified a problem with distributing lessons, called the l*esson distribution gap*, which is crucial to many lessons learned organizations. To address this problem, we introduced an approach called *monitored distribution*, which is characterized by a tight integration with a decision support tool that manages processes that the lessons can potentially improve. We implemented this approach in ALDS, a case retrieval system, and evaluated its capability in the context of a module for HICAP, a plan authoring tool. Our experiments with a simulated military planning domain (i.e., for noncombatant evacuation operations) showed that, by using lessons, monitored distribution can help to significantly improve plan performance measures. In summary, we demonstrated a technology that brings lessons to the attention of users when and where they are needed and applicable.

Acknowledgements

This research was supported by the Office of Naval Research and DARPA.

References

[Abecker *et al*., 2000] Abecker, A.; Bernardi, A.; Hinkelmann, K.; Kuehn, O.; and Sintek, M. Context-Aware, Proactive Delivery of Task-Specific Information: The KnowMore Project. Information Systems Frontiers 2:3/4, 253-276, 2000.

[Aha, 1992] Aha, D.W. Generalizing case studies: A case study. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 1-10). Aberdeen: Morgan Kaufmann, 1992.

[Aha and Weber, 2000] Aha, D.W., and Weber, R. (Eds.). *Intelligent Lessons Learned Systems: Papers from the AAAI Workshop* (Technical Report WS-00-03). Menlo Park, CA: AAAI Press, 2000.

[CALL, 2001] Center for Army lessons learned: Virtual research library. http://call.army.mil.

[Davenport and Prisak, 1998] Davenport, T.H., and Prusak, L. *Working knowledge: How organizations manage what they know.* Boston, MA: Harvard Business School Press, 1998.

[DOE, 1999] DOE *The DOE corporate lessons learned program* (Technical Report DOE-STD-7501-99). Washington, DC: U.S. Department of Energy.

[Fisher *et al*., 1998] Fisher, D., Deshpande, S., & Livingston, J. *Modeling the lessons learned process* (Research Report 123-11). Albequerque, NM: The University of New Mexico, Department of Civil Engineering, 1998.

[Leake *et al.*, 2000] Leake, D.B., Bauer, T., Maguitman, A., and Wilson, D.C. Capture, storage, and reuse of lessons about information resources: Supporting task-based information search. In (Aha & Weber, 2000).

[Muñoz-Avila *et al*., 1999] Muñoz-Avila, H., Aha, D.W., Breslow, L.A., & Nau, D. HICAP: An interactive case-based planning architecture and its application to NEOs. *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence* (pp. 870-875). Orlando, FL: AAAI Press, 1999.

[Reimer *et al*, 2000] Reimer, U., Margelisch, A., Staudt, M. A Knowledge-based Approach to Support Business Processes, In: Proceedings of the AAAI 2000 Spring Symposium Series: Bringing Knowledge to Business Processes, Stanford, CA, March 2000.

[Secchi *et al*., 1999] Secchi, P.; Ciaschi, R.; Spence, D. The ESA alert system. In P. Secchi (Ed.) *Proceedings of Alerts and Lessons Learned: An Effective way to prevent failures and problems* (Technical Report WPP-167). Noordwijk, The Netherlands: ESTEC, 1999.

[SELLS, 2000] SELLS *Proceedings of the SELLS Fall Meeting.* [tis.eh.doe.gov/ll/proceedings/proceedings1000.htm] [www.estec.esa.nl/CONFANNOUN/99c06], Fall, 2000.

[Siegel, 1991] Siegel, A. Eastern Exit: The Noncombatant Evacuation Operation (NEO) from Mogadishu, Somalia, in January 1991. Center for Naval Analyses, 1991.

[UNTL, 1996] UNTL *Universal Naval Task List* (OPNAVINST 3500.38). Arlington, VA: Navy Modeling and Simulation Office, September, 1996.

[Vandeville and Shaikh, 999] deville, J.V. and Shaikh, M. A. A structured approximate reasoning-based approach for gathering "lessons learned" information from system development projects. *Systems Engineering, 2*(4), 242-247, 1999.

[Watson, 2000] Watson, I. Lessons learned during HVAC installation. In (Aha & Weber, 2000).

[Weber *et al*., 2001a] Weber, R., Aha, D.W., and Becerra-Fernandez, I. Intelligent lessons learned systems. To appear in *Expert Systems with Applications, 20*(1), 2001.

[Weber *et al*., 2001b] Weber, R., Aha, D.W., Sandhu, N., & Muñoz-Avila, H. (2001). A Textual Case-Based Reasoning Framework for Knowledge Management Applications. Professionelles Wissenmanagement Erfahrungen und Visionen, 244-253. Aachen:Shaker Verlag.

Minimizing Dialog Length in Interactive Case-Based Reasoning

David McSherry
School of Information and Software Engineering
University of Ulster, Coleraine BT52 1SA
Northern Ireland

Abstract

Decision trees induced from stored cases are increasingly used to guide case retrieval in case-based reasoning (CBR) systems for fault diagnosis and product recommendation. In this paper, we refer to such a decision tree as an *identification* tree when, as often in practice, each of the faults to be identified, or available products, is represented by a single case in the case library. We evaluate common splitting criteria for decision trees in the special case of identification trees. We present simplified versions of those that are most effective in reducing the average path length of an identification tree, or equivalently, the average number of questions asked when the tree is used for problem solving. We also identify conditions in which *no* such reduction is possible with any splitting criterion.

1 Introduction

Interactive trouble-shooting, help-desk support, and internet commerce represent the majority of fielded applications of case-based reasoning (CBR) [Aha, 1998; Bergmann *et al*., 1999; Watson, 1997]. Increasingly, decision trees induced from stored cases are used to guide case retrieval with the aim of minimizing the number of questions required to identify a fault, or a product that meets the requirements of the user. A problem-solving session takes the form of an interactive dialog in which questions are presented in the order determined by the decision tree and the user's answers. Algorithms for building decision trees are usually evaluated in terms of predictive accuracy, whereas performance measures like efficiency and precision of retrieval are often more relevant in interactive CBR [Aha and Breslow, 1997; Breslow and Aha, 1997; Doyle and Cunningham, 2000].

The ability to solve problems by asking a small number of focused questions has been a major factor in the success of help-desk applications of CBR [Watson, 1997]. In on-line decision guides, minimizing the length of product-selection dialogs is important not only to avoid frustration for the user but also to reduce network traffic [Doyle and Cunningham, 2000]. Improving the clarity of explanations is another good reason for minimizing the length of problem–solving dialogs, without sacrificing solution quality, in interactive CBR [Breslow and Aha, 1997].

We will refer to the task of a CBR system as an *identification* task when, as often in practice, each of the faults to be identified, or available products, is represented by a single case in the case library. A decision tree for case retrieval in an identification task will be called an *identification* tree. Some researchers have questioned the usefulness of algorithms for building decision trees when instances in the data set are unlabelled; that is, not grouped according to the outcome class to which they belong [Aha and Breslow, 1997; Doyle and Cunningham, 2000]. However, an identification task can be regarded as a classification task in which each fault or product to be identified is a distinct outcome class. So, in principle, there is no reason why algorithms for top-down induction of decision trees cannot be used for building identification trees.

Most decision-tree algorithms, though, are designed for data sets with relatively small numbers of outcome classes, each of which is usually represented by many instances in the data set. In contrast, a data set for an identification task is *irreducible* in the sense that the deletion of a single instance means that the corresponding outcome class is no longer represented in the data set. An important question, therefore, is how the performance of decision-tree algorithms is affected by irreducibility in the data set.

Most algorithms for building decision trees differ in the criterion used to split the data set at a given node. In this paper, we evaluate splitting criteria such as the *information gain* criterion from ID3 [Quinlan, 1986] and the *Gini* criterion from CART [Breiman *et al.*, 1984] when used to build identification trees from irreducible data sets. The performance measure on which we focus is the average path length of the identification tree, or equivalently, the average number of questions asked when the tree is used for problem solving.

In Section 2, we show that a substantial reduction in average path length is possible in an irreducible data set

year = 71
    displacement = 100 to 149 : *mercury capri 2000*
    displacement = 200 to 249 : *plymouth satellite custom*
    displacement = 250 to 299 : *chevrolet chevelle malibu*
    displacement = 400 or more : *pontiac safari (sw)*
year = 73 : *oldsmobile vista cruiser*
year = 74
    acceleration = 10.0 to 14.9 : *opel manta*
    acceleration = 15.0 to 19.9 : *audi fox*
year = 75
    acceleration = 10.0 to 14.9 : *volkswagen rabbit*
    acceleration = 15.0 to 19.9 : *peugeot 504*
year = 76 : *ford granada ghia*
year = 77 : *ford mustang ii 2+2*
year = 78
    mpg = 15.0 to 19.9 : *ford futura*
    mpg = 20.0 to 24.9 : *buick century special*
    mpg = 30.0 to 34.9 : *dodge omni*
year = 79
    displacement = 300 to 349 : *dodge st. regis*
    displacement = 350 to 399 : *cadillac eldorado*
year = 80
    origin = 1 : *chevrolet citation*
    origin = 2 : *audi 5000s (diesel)*
year = 81
    horsepower = 50 to 99 : *plymouth champ*
    horsepower = 100 to 149 : *oldsmobile cutlass ls*

Figure 1. An identification tree for 20 automobiles selected at random from the AutoMPG data set.

with information gain (InfoGain) as the splitting criterion. We also identify conditions in which *no* such reduction is possible with any splitting criterion. In Section 3, we present simplified versions of the InfoGain and Gini criteria for irreducible data sets, and identify another splitting criterion that can never discriminate between attributes in an irreducible data set. In Section 4, we present an empirical comparison of splitting criteria for identification trees. We discuss related work in Section 5, and present our conclusions in Section 6.

2 Irreducible Data Sets

Whether a given data set is irreducible depends on which attribute is defined as the *class* attribute. For example, the class attribute in the AutoMPG data set from the UCI repository [Blake and Merz, 1998] is normally *miles per gallon*, which is not unique for every instance. However, with the identity of each automobile as the class attribute, the data set is irreducible. Most of the attributes in the data set, such as *year*, *acceleration* and *miles per gallon*, are features that one would expect to see in a CBR system for recommending previously-owned automobiles.

There are missing values for only one attribute, namely *horsepower*. To simplify our comparison of splitting criteria, the six instances that have missing values for this attribute were omitted from the data set in our experiments. However, we shall return to the issue of missing values in our discussion of related work. Continuous attributes in the data set were discretised by dividing their ranges into intervals that seemed most natural for the expression of user preferences. There is a trade-off in this process between preserving the ability of the attributes to discriminate between the available alternatives and limiting the number of options from which the user is required to select.

Following the discretisation of continuous attributes, the numbers of values of attributes in the data set are as shown below.

year	13
mpg, displacement, weight	8
cylinders, horsepower	5
acceleration	4
origin	3

Although most instances in the data set can be uniquely identified by these attributes, some have the same values for all the attributes. A possible solution to this problem, which is not uncommon in product data, is to present the user with a *list* of recommended products when no attribute can distinguish between those that remain [Doyle and Cunningham, 2000].

Figure 1 shows an identification tree for 20 automobiles selected at random from the 392 instances that have no missing values in the AutoMPG data set. A point we would like to emphasize is that the values of an attribute in a given subset of a data set are those that are actually represented in the subset. For example, some values of the attributes in the AutoMPG data set, such as *year* = 72, are not represented in the subset from which the example identification tree was constructed. Similarly, *displacement* has only two values in the smaller subset with *year* = 79.

The example identification tree was constructed with InfoGain as the splitting criterion. The average path length is only 1.9 compared with 4.2 for an identification tree constructed with attributes selected purely at random. On the other hand, it can be seen from the following proposition that no splitting criterion can reduce the average path length of an identification tree for a data set that is both irreducible and *complete*. A data set is complete if every combination of attribute values is represented in the data set [Cendrowska, 1987].

Proposition 1 *In any identification tree for a data set that is both irreducible and complete, the path length required to identify a given instance is never less than the number of attributes in the data set.*

However, real-world data sets are seldom complete. For example, the instances in the AutoMPG data set represent less than 1 in 5,000 of the possible combinations of attribute values. It is also worth noting that the problem does not usually arise in data sets that are not irreducible. For example, Cendrowska's [1987] Contact Lens data set is complete, but only a single test is required to reach one of

the leaf nodes in the decision tree induced from the data set with InfoGain as the splitting criterion.

With single instances at their leaf nodes, identification trees tend to be larger than decision trees for classification tasks. For data sets of realistic size, it is seldom feasible to identify an optimal tree by exhaustive search. The worst-case scenario occurs when the data set is complete, although in this case the average path length is always the same by Proposition 1. The number of possible trees is more easily determined if all attributes have the same number of values.

Proposition 2 *For a data set that is both irreducible and complete, and has k attributes each with r values, the number of possible identification trees is* $\frac{(k!)^r}{k^{r-1}}$.

3 Splitting Criteria for Identification Trees

Measures used to define splitting criteria for building decision trees include the entropy function:

$$-\sum_j q_j \log q_j$$

and the Gini index:

$$1-\sum_j q_j^2$$

where q_1, q_2,..., q_m are the proportions of the outcome classes in the data set. The InfoGain criterion [Quinlan, 1986] selects the attribute that maximizes the expected gain in information (or reduction in entropy) when used to split the data set. Similarly, the Gini criterion [Breiman *et al.*, 1984] selects the attribute that maximizes the expected reduction in the Gini index.

3.1 The Gain Ratio Criterion

To compensate for its bias towards attributes that have most values, the InfoGain criterion was replaced in later versions of ID3 and in C4.5 by the *gain ratio* criterion [Quinlan, 1993], in which the information gain for an attribute (with at least average information gain) is divided by:

$$-\sum_i p_i \log p_i$$

where p_1, p_2,..., p_r are the proportions of the attribute's values in the data set. As the following theorem and corollary show, all attributes in an irreducible data set are equally good according to the gain ratio criterion. Gain ratio can therefore be eliminated from consideration as a basis for attribute selection in an irreducible data set.

Theorem 1 *For any attribute in an irreducible data set, the expected information gain is:*

$$-\sum_i p_i \log p_i$$

where p_1, p_2,..., p_r are the proportions of the attribute's values in the data set.

Proof Since each outcome class is represented by a single instance, the entropy of the data set is:

$$-\sum \frac{1}{n}\log\frac{1}{n} = \log n$$

where n is the number of instances in the data set. Similarly, the entropy of any subset of size k is $\log k$. The expected information gain when an attribute is used to split the data set is therefore:

$$\log n - \sum_i p_i \log k_i = \log n - \sum_i p_i (\log p_i + \log n) = -\sum_i p_i \log p_i$$

where k_1, k_2,..., k_r are the frequencies of the attribute's values in the data set.

Corollary *The gain ratio is the same for any attribute in an irreducible data set.*

3.2 The InfoGain Criterion

Theorem 1 provides a simplified version of the InfoGain criterion for irreducible data sets. As Theorem 2 shows, it can alternatively be expressed in terms of the frequencies of an attribute's values in the data set.

Theorem 2 *In an irreducible data set, the InfoGain criterion is equivalent to selecting the attribute for which:*

$$\sum_i k_i \log k_i$$

is minimum, where k_1, k_2,..., k_r are the frequencies of the attribute's values in the data set.

Proof It follows from Theorem 1 that the expected information gain for any attribute in an irreducible data set is:

$$-\sum_i p_i \log p_i = -\sum_i \frac{k_i}{n}(\log k_i - \log n) = -\frac{1}{n}\sum_i k_i \log k_i + \log n$$

where n is the size of the data set and p_1, p_2,..., p_r are the proportions of the attribute's values in the data set.

3.3 The Gini Criterion

As the following theorem shows, the Gini criterion is equivalent in an irreducible data set to selecting the attribute with the largest number of values.

Theorem 3 *For any attribute in an irreducible data set, the expected reduction in the Gini index is* $\frac{r-1}{n}$*, where n is the size of the data set and r is the number of values of the attribute.*

Proof The Gini index for the data set is $1-\sum \frac{1}{n^2} = 1-\frac{1}{n}$. Similarly, the Gini index for a subset of size k is $1-\frac{1}{k}$. The expected reduction in the Gini index when an attribute is used to split the data set is therefore:

$$(1-\frac{1}{n})-\frac{1}{n}\sum_i k_i(1-\frac{1}{k_i}) = 1-\frac{1}{n}-1+\frac{r}{n} = \frac{r-1}{n}$$

where k_1, k_2,..., k_r are the frequencies of the attribute's values in the data set.

At first sight, it may seem to follow from Theorem 3 that the Gini criterion can be of no use as a basis for attribute selection in an irreducible data set if all attributes in the data set have the same number of values. However, as noted in Section 2, the number of *represented* values of an attribute decreases as the data set is partitioned. The Gini criterion does therefore discriminate between attributes that initially have equal numbers of values. In an irreducible data set in which all attributes are binary, the Gini criterion is equivalent to selecting any attribute that is capable of splitting the current subset of the data set; that is, does not have the same value for every instance in the current subset. As we show in Section 4, this simple criterion, which we call *Reduce*, is much better than selecting attributes purely at random. In the latter strategy, the effect of selecting an attribute for which all instances have the same value is to increase the path length without reducing the size of the data set.

4 Comparison of Splitting Criteria

We now present an empirical comparison of InfoGain and Gini as splitting criteria for building identification trees from irreducible data sets. Our experiments are based on the simplified versions of InfoGain and Gini presented in Section 3. A baseline for their evaluation is provided by a tree-building strategy in which attributes are selected purely at random. Also included in our evaluation is the Reduce criterion which is equivalent to Gini for irreducible data sets in which all attributes are binary. The strategies to be compared are:

InfoGain: Select the attribute for which $\sum_i k_i \log k_i$ is minimum, where k_1, k_2,..., k_r are the frequencies of the attribute's values in the current subset

Gini: Select the attribute that has most distinct values in the current subset

Reduce: Select any attribute that does not have the same value for every instance in the current subset

Random: Select an attribute purely at random

Our algorithm for building identification trees, called *Identify*, works in a similar way to algorithms for top-down induction of decision trees and can be used with any splitting criterion. It recursively partitions the data set until a subset is reached that contains only a single instance, or none of the remaining attributes can discriminate between the instances in the current subset, or all attributes have been used in the Random strategy.

4.1 The AutoMPG Data Set

Our first experiment compares the performance of the splitting criteria on randomly generated subsets of the AutoMPG data set ranging in size from 10 to 100 per cent. As noted in Section 2, the data set is itself only a small subset of the complete data set consisting of all combinations of the attribute values.

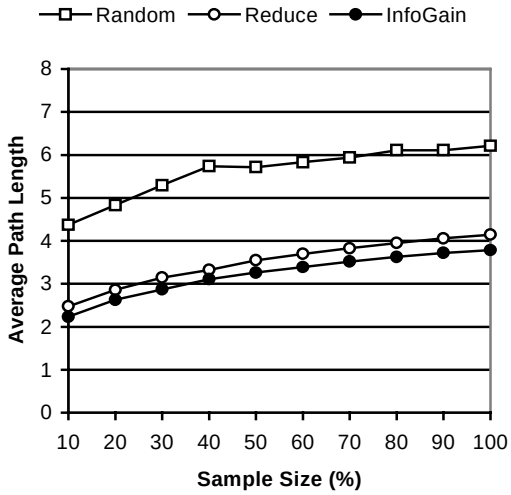


Figure 2. Comparison of InfoGain, Random and Reduce on subsets of the AutoMPG data set.

Average path lengths over 10 repeated trials for each sample size are shown in Figure 2. The results for Gini are not shown because they differ only slightly from those for InfoGain. While InfoGain performed slightly better than Gini for all sample sizes, its reduction in average path length relative to Gini was never more than one per cent. On the other hand, the Reduce criterion can be seen to provide a considerable reduction in average path length compared with selecting attributes purely at random. The additional reduction provided by InfoGain, though relatively small, is consistent over the range of sample sizes. While the average path length of the InfoGain trees increases as sample size increases, their efficiency is apparent even for the 100 per cent sample.

4.2 Which Criterion is Best for Binary Attributes?

Our second experiment compares the performance of InfoGain and Gini on irreducible data sets in which all attributes are binary. As noted in Section 3, Gini is equivalent in such data sets to the Reduce criterion. The data sets used in this experiment are randomly generated subsets of a complete, irreducible data set with 12 binary attributes. The class attribute is an integer that uniquely identifies each of the 4096 instances in the data set. Average path lengths for InfoGain and Gini over 10 repeated trials are shown in Figure 3 for subsets of the complete data set ranging in size from 10 to 100 per cent.

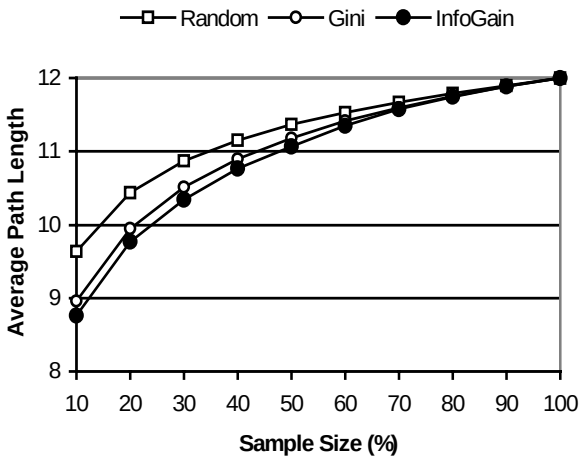


Figure 3. Comparison of InfoGain, Gini and Random on subsets of a complete data set with 12 binary attributes.

The effectiveness of both splitting criteria can be seen to decrease as sample size increases, with little reduction in average path length for sample sizes greater than 70 per cent. As predicted by our theoretical results, there is no reduction in average path length for the 100 per cent sample. However, for sample sizes less than 70 per cent, InfoGain now gives a noticeable improvement in comparison with Gini. The ability of Gini to almost match the performance of InfoGain on subsets of the AutoMPG data set may therefore be attributable to the absence of binary attributes in the data set.

4.3 How Close to Optimal is InfoGain?

We can attempt to answer this question only for data sets in which the number of attributes is small enough for optimal trees to be identified by exhaustive search. The data sets in our third experiment are randomly generated subsets of a complete, irreducible data set in which each of 3 attributes has 10 values. By Proposition 2, the number of possible trees for any such data set is at most 3,072.

Figure 4 compares the average path length of InfoGain trees and optimal trees for subsets ranging in size from 1 to 10 per cent of the complete $10 \times 10 \times 10$ data set. Based on averages over ten repeated trials, the results show that the InfoGain trees are close to optimal. Of the 100 InfoGain trees constructed in the experiment, 30 were optimal.

5 Related Work

Doyle and Cunningham [2000] applied a clustering algorithm to product data before using InfoGain to construct a decision tree. As shown by our results, however, InfoGain is often very effective in reducing average path length when applied *directly* to irreducible data sets. Doyle and Cunningham used a demand-driven or *lazy* approach to decision-tree induction, adapted from [Smyth and Cunningham, 1994], in which an explicit decision tree is not constructed. Instead, the user's answers are used to construct a single decision *path*. One advantage is that the user can select the attribute they consider most important instead of the one considered most useful by the system. In fault diagnosis, another advantage of a demand-driven approach to decision-tree induction is the system's ability to select the next most useful attribute to partition the case library when the user is unable to answer the most useful question [McSherry, 2001]. A demand-driven approach to the construction of identification trees is expected to provide similar benefits in a future version of Identify. The effects of incomplete data on retrieval performance, however, remain to be investigated.

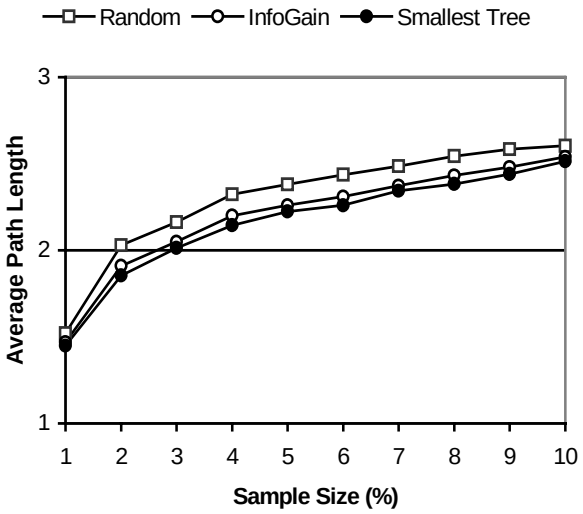


Figure 4. Comparison of InfoGain trees and optimal trees for subsets of a complete $10 \times 10 \times 10$ data set.

Aha and Breslow [1997] used decision trees as an intermediate representation to improve the quality of conversational case libraries. In conversational CBR, the problem of irreducibility addressed in this paper is often complicated by the use of different features to describe cases. In fault diagnosis, for example, certain attributes may not be relevant or even meaningful for every case. The result is a high frequency of unanswered questions, or missing values, in cases. The splitting criterion used by Aha and Breslow to address this problem selected the attribute with fewest missing values. In the absence of missing values, though, this criterion is at best equivalent to selecting any attribute that is capable of splitting the data set and therefore inferior to InfoGain as shown by our results. In this paper, we have assumed the absence of missing values to simplify our initial comparison of splitting criteria for irreducible data sets. However, the impact of missing values on retrieval performance is an important issue to be addressed by further research.

Algorithms for building decision trees are usually evaluated by dividing a data set into a *training set* that is used to build a decision tree and a *test set* that is used to assess its predictive accuracy [Breslow and Aha, 1997].

However, an identification tree has no predictive accuracy; that is, it can correctly identify only the instances from which it was constructed. Evaluation in terms of predictive accuracy is equally compromised by irreducibility in the case library in CBR systems that rely on similarity-based retrieval. Aha *et al.* [1998] propose a *leave-one-in* approach to evaluating retrieval precision for conversational case libraries in which most (or all) cases have unique solutions. In their approach, each case is used as a test case but without removing it from the case library during testing. Precision is measured by how often the solution for the most similar case matches the solution for the test case. We plan to use a similar approach to the evaluation of precision in an investigation of the effects of incomplete data and missing values on retrieval performance in interactive CBR.

6 Conclusions

Minimizing the number of questions required to identify a fault, or a product that meets the requirements of the user, is an important objective in interactive CBR applications such as fault diagnosis, help-desk support, and on-line decision guides. Often in these applications, each case has a unique solution or represents a unique product in the case library. We have examined the performance of common splitting criteria for building identification trees to guide the retrieval of cases from such irreducible case libraries.

Our results show that the InfoGain criterion [Quinlan, 1986] is often more effective than other splitting criteria in reducing average path length when used to build identification trees from irreducible data sets. While the Gini criterion [Breiman *et al*., 1984] appears to compete well with InfoGain for attributes with several values, our results suggest that InfoGain may be more effective for binary attributes. The effectiveness of both criteria decreases as a data set (or case library) approaches completeness, and no reduction in average path length is possible in an irreducible data set that is also complete.

We have shown that for irreducible data sets, InfoGain and Gini can be expressed in terms of the frequencies of an attribute's values, thus providing a reduction in computational effort that may be of particular benefit in on-line decision guides. Issues to be addressed by further research include the effectiveness of techniques for tolerating missing values in decision tree learning [Quinlan, 1989] in the special case of identification trees.

References

[Aha, 1998] David Aha. The omnipresence of case-based reasoning in science and application. *Knowledge-Based Systems*, 11:261-273, 1998.

[Aha and Breslow, 1997] David Aha and Leonard Breslow. Refining conversational case libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 267-278, Providence, Rhode Island, 1997. Springer-Verlag.

[Aha *et al.*, 1998] David Aha, Tucker Maney and Leonard Breslow. Supporting dialogue inferencing in conversational case-based reasoning. In *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 262-273, Dublin, Ireland, 1998. Springer-Verlag.

[Bergmann *et al*., 1999] Ralph Bergmann, Sean Breen, Mehmet Göker, Michael Manago and Stefan Wess. *Developing Industrial Case-Based Reasoning Applications: The INRECA Methodology*. Springer-Verlag, Berlin Heidelberg, 1999.

[Blake and Merz, 1998] Catherine Blake and Christopher Merz. *UCI Repository of Machine Learning Databases*. University of California, Irving, California, 1998.

[Breiman *et al*., 1984] Leo Breiman, Jerome Friedman, Richard Olshen and Charles Stone. *Classification and Regression Trees*. Wadsworth, Belmont, California, 1984.

[Breslow and Aha, 1997] Leonard Breslow and David Aha. Simplifying decision trees: a survey. *Knowledge Engineering Review*, 12:1-40, 1997.

[Cendrowska, 1987] Jadzia Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27:349-370, 1987.

[Doyle and Cunningham, 2000] Michelle Doyle and Pádraig Cunningham. A dynamic approach to reducing dialog in on-line decision guides. In *Proceedings of the Fifth European Workshop on Case-Based Reasoning*, pages 49-60, Trento, 2000. Springer-Verlag.

[McSherry, 2001] David McSherry. Interactive case-based reasoning in sequential diagnosis. *Applied Intelligence*, 14:65-76, 2001.

[Quinlan, 1986] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81-106, 1986.

[Quinlan, 1989] Ross Quinlan. Unknown attribute values in induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 164-168, Ithaca, New York, 1989. Morgan Kaufmann.

[Quinlan, 1993] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[Smyth and Cunningham, 1994] Barry Smyth and Pádraig Cunningham. A comparison of incremental case-based reasoning and inductive learning. In *Proceedings of the Second European Workshop on Case-Based Reasoning*, pages 151-164, Chantilly, November 1994. Springer-Verlag.

[Watson, 1997] Ian Watson. *Applying Case-Based Reasoning*: *Techniques for Enterprise Systems*. Morgan Kaufmann, San Francisco, California, 1997.

SiN: Integrating Case-based Reasoning with Task Decomposition

Héctor Muñoz-Avila,[1] David W. Aha,[2] Dana S. Nau,[1] Rosina Weber,[2]
Len Breslow[2] & Fusun Yamal[1]
[1]Department of Computer Science, University of Maryland, College Park, MD 20742-3255
[2]Navy Center for Applied Research in Artificial Intelligence,
Naval Research Laboratory (Code 5515), Washington, DC 20375
[1]lastname@cs.umd.edu [2]lastname@aic.nrl.navy.mil

Abstract

This paper describes SiN, a novel case-based planning algorithm that combines conversational case retrieval with generative planning. SiN is provably correct, and can generate plans given an incomplete domain theory by using cases to extend that domain theory. SiN can also reason with imperfect world-state information by incorporing preferences into the cases. Our empirical validation shows how these preferences affect plan quality.

1 Introduction

Generative planners traditionally require a complete domain theory, which provides a clear semantics for the planner's inferencing mechanism. This allows a planner to be used in different domains. However, in many planning domains, developing a complete domain theory is infeasible.

In this paper we present a case-based planning algorithm called *SiN* (SHOP integrated with NaCoDAE), which integrates the *SHOP* generative planner [Nau *et al.*, 1999] with *NaCoDAE, a* conversational case retriever [Breslow & Aha, 1997]. SiN is a provably correct algorithm that does not require a complete domain theory nor complete information about initial or intermediate world-states.

In addition to describing SiN, which has been implemented in HICAP [Muñoz-Avila *et al.*, 1999], we present sufficient conditions to ensure its correctness, show how SiN represents preferences in cases to generate plans in the context of imperfect world state information, and describe an empirical analysis that demonstrates the impact of the preferences on plan quality.

In the following sections, we introduce some terminology, detail SiN, including theoretical results on its semantics with respect to incomplete domain theory, present SiN's empirical evaluation to show the role of preferences to handle incomplete world state information, and discuss the implications of these results.

2 Motivation

SiN's design was partly motivated by the following characteristics of military planning operations.

- Military operations are strongly hierarchical [Mitchell 1997; Muñoz-Avila *et al.*, 1999]. Thus, we chose to represent plans using Hierarchical Task Networks (HTNs) [Erol *et al.*, 1994].
- There is an incomplete domain theory, in the form of general guidelines (doctrine) and standard operating procedures (SOPs). However, neither doctrine nor SOPs can be used to derive detailed tactical plans, which often require knowledge about previous experiences. Thus, SiN uses SHOP to perform first-principles reasoning and NaCoDAE to employ previous experiences.
- Military planners do not have complete information about the current situation; part of the planning includes dynamic information gathering, typically to assess enemy capabilities and/or deployment. In SiN, NaCODAE is used to plan with an incomplete world state using *preferences*, which we define in Section 4.

3 Notation and definitions

An ***HTN*** is a set of tasks and their ordering relations, denoted as $N=(\{t_1,\ldots,t_m\},<)$ $(m\geq 0)$, where $<$ is a binary relation expressing temporal constraints between tasks. Decomposable tasks are called ***compound***, while non-decomposable tasks are called ***primitive***.

A ***domain theory*** consists of methods and operators for generating plans. A ***method*** is an expression of the form $M=(h,P,ST)$, where h (the method's ***head***) is a compound task, P is a set of ***preconditions***, and ST is the set of M's (children) ***subtasks***. M is ***applicable*** to a task t, relative to a ***state*** S (a set of ground atoms), iff matches(h,t,S) (i.e., h and t have the same predicate and arity, and a consistent set of bindings È exists that maps variables to values such that all terms in h match their corresponding ground terms in t) and

the preconditions *P* are ***satisfied*** in *S* (i.e., there exists a consistent extension of È, named È', such that $\forall\ p \in P$ {*p*È'$\in S$}), in which case *M*(*t*,*S*)=*ST* È'.

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

A ***planning problem*** is a triple (*T*,*S*,*D*), where *T* is a set of tasks, *S* is a state, and *D* is a domain theory. A ***plan*** is the collection of primitive tasks obtained by decomposing all compound tasks in a planning problem (*T*,*S*,*D*).

4 Cases in SiN

In many domains it is impossible to assume that a complete domain theory of the world is known. For example, this is true when planning for non-combatant evacuations (NEOs). However, a partial domain theory exists for NEOs, and it can be elicited from doctrine and standard operating procedures [DOD, 1997].

Reasoning about parts of the domain for which no domain theory is available is done through cases. A ***case*** *C* is an instance of a method, denoted by *C=(h,P,ST,Q)*, where *h*, *P*, and *ST* are defined as for methods and *Q* is a set of <question,answer> pairs. *Q* defines ***preferences*** for matching a case to the current state. Preferences are useful for ranking cases in the context of incomplete world states and/or domain theories because, as we will show, they focus users on providing relevant additional state information.

5 SiN mixed-initiative planner

SiN integrates SHOP and NaCoDAE's task decomposition algorithms. A single (current) state *S* is maintained in SiN that is accessible to and updateable by both SHOP and NaCoDAE. Answers given by the user during an interaction with NaCoDAE are added to *S* (i.e., each question has a translation into a ground atom). Changes to the state that occur by applying SHOP's operators are also reflected in *S*.

SHOP generative planner. At any point during the planning process, SHOP is refining a task list *T'* relative to a state *S* and a domain theory *D*. Initially, *T'* is the set of tasks *T* in the planning problem (*T*,*S*,*D*). SHOP performs *ordered task decomposition* [Nau *et al.*, 2000], meaning that the tasks must be totally ordered (i.e., the < relation on HTNs is a total order). SHOP also maintains the partial solution plan *p* being derived (i.e., the primitive tasks in *T'*). Initially *p* is empty. SHOP selects the first task *t* in *T'* and continues as follows:

- If *t* is primitive and has an applicable operator *O*, then *O* is applied to *t*, *S* is updated accordingly, *t* is removed from *T'* and added to the end of *p*.
- Else if *t* is compound and has an applicable method *M* (that has not yet been applied to *t*), then *M* is applied, which replaces *t* in *T'* with *M*'s subtasks.
- Else if *T'* is not empty, then SHOP backtracks.
- Else SHOP fails.

SHOP terminates when *T'* is empty, in which case *p* is the solution, or when SHOP tries to backtrack on a compound task *t* whose applicable methods have been exhausted.

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

SiN integrated planning algorithm. SiN receives as input a set of tasks *T*, a state, *S*, and a knowledge base $I \cup B$ consisting of an incomplete domain theory *I* and a collection of cases *B*. The output is a solution plan *p* consisting of a sequence of operators in *I*. Both SHOP and NaCoDAE assist SiN with refining *T* into a plan. As does SHOP, SiN maintains the set of tasks in *T'* that have not been decomposed and the partial solution plan *p*. At any point of time, either SHOP or NaCoDAE is in control and is focusing on a compound task $t \in T'$ to decompose. SiN proceeds as follows:

- **Rule # 1**: If SHOP is in control and can decompose *t*, it does so and retains control. If SHOP cannot decompose *t*, but NaCoDAE has cases for decomposing *t*, then SHOP will cede control to NaCoDAE.
- **Rule # 2**: If NaCoDAE is in control, it has cases for decomposing *t* whose pre-conditions are satisfied. If the user applies one of them to decompose *t*, then NaCoDAE retains control. If NaCoDAE has no cases to decompose *t* or if the user decides not to apply any applicable case, then if *t* is SHOP-decomposable, NaCoDAE will cede control to SHOP.

If neither of these rules applies, then SiN backtracks, if possible. If backtracking is impossible (e.g., because *t* is a task in *T*), this planning process is interrupted and a failure is returned.

By continuing in this way, and assuming that the process is not interrupted with a failure, SiN will eventually yield a plan *p* (i.e., consisting only of primitive tasks).

6 Correctness of SiN

In this section we will assume that SiN performs ordered task decomposition. That is, we assume that all tasks are totally ordered and at each iteration, when refining a set of tasks *T'*, SiN will start by decomposing the first task in *T'*. A relaxation of this condition should be possible once we modify SiN to include the extended version of SHOP that can represent partial-order task relations [Nau *et al*, 2001].

If *I* is an incomplete domain theory and *B* is a case base (i.e., a set of cases), then a domain theory *D* is ***consistent*** with $I \cup B$ 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,P,ST,Q)* in *B*, there is a method M=(*h',P',ST'*) in *D* such that *h*, *P*, and *ST* are instances of *h', P'* and *ST'* respectively. Although many different domain theories might be consistent with $I \cup B$, in general we will not know which of these is the one that produced *I* and *B*. However, we can prove that SiN is correct in the sense that, if it succeeds in outputting a plan, then that plan could have been generated by SHOP using any domain theory consistent with $I \cup B$.

Proposition (Correctness of SiN). Let *T* be a collection of tasks, *S* be an initial state, *I* be an incomplete domain theory, and *B* be a case base, and let SiN(*T,S,I,B*) represent the invocation of SiN with those items as inputs. Suppose that SiN performs ordered task decomposition. Then:

(1) If SiN(*T,S,I,B*) returns a plan *p*, then for every domain theory *D* consistent with $I \cup B$, *p* is a solution plan for the planning problem (*T,S,D*).

(2) If SiN(*T,S,I,B*) cannot find a plan, then there is a domain theory *D* consistent with $I \cup B$ such that no solution plan exists for (*T,S,D*).

The proof is done by induction on the number of iterations of the SiN algorithm. The proof shows that each SiN task decomposition in ($T,S,I \cup B$) corresponds to a SHOP task decomposition in (*T,S,D*). This is sufficient to prove correctness because SHOP is known to be correct [Nau *et al*., 1999]. We omit the details of the proof due to space limitations.

This proposition suggests that cases in SiN supply two kinds of knowledge: first, they provide control knowledge, similar to the knowledge encoded in cases using derivational replay when a complete domain theory is available [Veloso, 1994; Ihrig & Kambhampati, 1994]. Because cases are instances of methods, applying a case is comparable to a replay step in which the method selected to decompose a task is the one in the case's derivational trace. The main difference is that, while cases in replay systems correspond to a complete derivational trace, cases in SiN correspond to a single step in the derivational trace. Second, cases in SiN augment the domain theory and, thus, provide domain knowledge as do cases in many case-based planners (e.g., [Hammond, 1986]).

7 Imperfect World Information

SiN uses NaCoDAE to dynamically elicit the world state, which involves obtaining the user's preferences. Depending on the user's answers, cases will get re-ranked. When solving a task, the user can choose any of the cases, independent of their ranking, provided that all their preconditions are met. The preferences play a pivotal role in determining plan quality due to the absence of a complete domain theory.

Consider the following two simplified cases:

Case 1:

Head: selectTransport(ISB,NEOsite)
Preconditions: HelosAvailable(ISB)
Questions-Answer pairs: Weather conditions? Fine
Subtasks: Transport(ISB,NEOsite,HELOS)

Case 2:

Head: selectTransport(ISB,NEOsite)
Preconditions: groundTransportAvailable(ISB)
Questions-Answer pairs:

- Weather conditions? Rainy
- Imminent danger to evacuees? No

Subtasks: Transport(ISB,NEOsite,GroundTransport)

These cases both concern the selection of transportation means between an intermediate staging base (ISB) and the NEO site (NEOsite). The first case suggests using helicopters provided that they are available at the ISB. The second one suggests using ground transportation provided that the corresponding transportation means are available at the ISB. If the two cases are applicable, because both preconditions are met, the answers given by the user will determine a preference between them. For example if the weather is rainy and there is no immediate danger for the evacuees, NaCoDAE would suggest the second case. The rationale behind this is that flying in rainy conditions is risky. Thus, selecting ground transportation would be a better choice.

8 Evaluation

Our experiments focused on the role of preferences in the plan generation process in the context of incomplete world state information and how they affect the quality of the resulting plans. Towards this goal, we developed two planning domains where the contents of the world state can significantly impact choices during planning.

For these experiments, we encoded an automatic user that dynamically provided preferences when asked by NaCoDAE. The user provided preferences with a pre-defined bias (defined in the following sections) towards certain kinds of solutions. The automatic user will always

select the case with the highest similarity. In situations where several candidate cases had the same highest similarity, the automatic user selected one of them randomly. The purpose was to observe whether the resulting plans reflect the user's bias despite the incomplete information about the world state.

8.1 The Personal Travel Domain

The first domain was the *personal travel domain*. Its plans concern traveling from locations in Washington, DC to downtown New York City (NYC). We encoded 7 transportation methods (3 inter- and 4 intra-city). Plans consist of 3-5 planning segments. States indicate different locations, whether connections between locations exists and by which means, weather conditions, etc. The knowledge base consists of 10 methods and 1 operator for SHOP and 40 cases for NaCoDAE.

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

We selected ten goals, corresponding to ten pairs of departing and arrival locations in Washington, DC and downtown NYC, respectively. For each goal, we generated 10 random world states, thus yielding 100 total planning problems. SiN was then used to generate a plan for each problem, thus yielding 300 plans. Each was executed 10 times by the plan evaluator, for a total of 3000 runs.

Table 1: Results for the personal travel domain.

Preference	*Duration*	*Price*	*Success*
Bus	676	85	92.2%
Train	466	176	94.3%
Plane	375	338	77.0%

Table 1 summarizes the results. When the user's bias was given towards taking the Bus for the intercity part of the trip. This preference yields maximal (676 minutes) durations, but has the cheapest price. On the other extreme, when the Plane is the preferred means of transportation, the duration is the shortest but the price is the most expensive. The lowest success rate occurs for plane trips; it reflects missing connections due to external factors such as the weather. Although a bias expresses a preference for a certain transportation mode, it does not imply that it was always selected; world state conditions may prevent the use of some travel modes for particular situations.

8.2 The NEO Planning Domain

The second domain was the *Noncombatant Evacuation Operations Domain*. Its plans involve performing a rescue mission where troops are grouped and transported between an initial location (the assembly point) and the NEO site (where the evacuees are located). After the troops arrived at the NEO site, evacuees are re-located to a safe haven. Planning involves selecting possible pre-defined routes, consisting of 4 segments each. The planner must also choose a transportation mode for each segment. In addition, other conditions were determined during planning such as whether communication exists with State Department personnel and the type of evacuee registration process. SiN's knowledge base included 6 operators, 22 methods, and 51 cases.

As with the personal travel domain evaluator, the NEO planning evaluator can generate a different output each time it is given a plan and a world state because of its non-deterministic execution. For each run, it outputs the plan execution duration, the time it took to reach the evacuees, and the evacuee casualties. Similar to the personal travel domain, we applied a delay function that is influenced by world state conditions. The Neo Planning evaluator is more complex than the personal travel evaluator because there are more conditions that can affect the output variables and these conditions may interact. For example, a small-sized force will incur fewer delays because embarking troops in the transportation means will take less time than for large-sized forces. However, smaller force increase the chances of hostile attacks, which if they occur will delay the operation.

We had a single task, to perform a NEO, and generated 100 random world states, thus yielding 100 planning problems. SiN was then used to generate a plan for each problem, thus yielding 200 plans, and each was executed 10 times by the plan evaluator for a total of 2000 runs.

Table 2: Results for the NEO planning domain.

Preference	*Duration*	*Time to Reach Evacuees*	*Evacuee Casualties*
Helicopter	38.5	28.7	11%
Ground Vehicle	48.1	34.8	16%

Table 2 summarizes the results. The helicopter transport preference yields plans that have shorter execution durations (38.5 hours) and require less time to reach the evacuees (28.7 hours). In addition, average casualties among evacuees is less (11%), due mainly to the shorter time to reach them, and land travel is generally riskier than air travel. Still the number of casualties among evacuees is high even with helicopters. This is due to the simple bias encoded

in the simulated user. Human users could yield better (and worse) plans by dynamically providing more sophisticated preferences depending on the world state conditions.

8.3 Discussion

The experiments show the capabilities of SiN in allowing the user to guide the planning process towards their preferences while dynamically capturing world-state conditions. Despite our use of simplistic simulated users, the quality of the plans reflect the user's bias.

9 Related Work

Table 3 compares seven different features of SiN to those of other planning systems.

Table 3: Comparisons between different systems. Conventions: Gen=Generative; CBP=Case-Based; M-I=Mixed-initiative; I=Interleaved control structure; DK=Cases are used to supply domain knowledge; CK=Cases are used to supply control knowledge.

System	Gen	CBP	M-I	I	DK	CK
SiN	√	√	√	√	√	√
CHEF		√			√	
MI-CBP	√	√	√			√
NaCoDAE		√	√		√	
Prodigy/ Analogy	√	√				√
SHOP	√					
SIPE II	√		√			

We first discuss the features shown in columns 2–5 of Table 3. SHOP [Nau *et al.*, 1999], as is typical of generative planners, requires a complete domain theory. CHEF [Hammond, 1989] and DIAL [Leake *et al*., 1997] are case-based, but do not have a generative component, and thus need a large case base to perform well across a wide variety of problems. Prodigy/Analogy [Veloso, 1994], DerSNLP [Ihrig & Kambhampati, 1994], and Paris [Bergmann & Wilke, 1995] integrate generative and case-based planning, but require a complete domain theory and are not mixed-initiative. SIPE II [Wilkins, 1998] is a mixed-initiative generative planner, but does not use cases. NaCoDAE [Muñoz-Avila *et al.*, 1999] is a mixed-initiative case-based planner, but does not employ generative planning.

At least three other integrated (case-based/generative), mixed-initiative planners exist. MI-CBP [Veloso *et al.*, 1997], which extends Prodigy/Analogy, limits interaction to providing it with user feedback on completed plans. Thus, it must input, or learn thru feedback, a sufficiently complete domain theory to solve problems. In contrast, SiN gathers information it requires from the user through NaCoDAE conversations, but does not learn from user feedback. CAPlan/CbC [Muñoz-Avila *et al.*, 1997] and Mitchell's [1997] system use interaction for plan adaptation rather than to acquire state information.

Among integrated case-based/generative planners, SiN's interleaved control structure is unique in that it allows both subsystems to equally control the task decomposition process. In contrast, other approaches either use heuristics (Prodigy/Analogy; MI-CBP) or order case-based prior to generative planning [DerSNLP; Mitchell, 1997], although Paris does this iteratively through multiple abstraction levels. Distinguishing the relative advantages of these control strategies is an open research issue.

The final two columns of Table 3 refer to the types of contributions made by cases in CBP systems. CHEF and NaCoDAE both use cases to provide domain knowledge, while Prodigy/Analogy uses cases for control knowledge (i.e., determining which planning constructs to apply). In contrast, SiN uses cases to both provide domain knowledge (i.e., instances of methods) and control knowledge (i.e., it allows the user selects which of these instance methods to apply).

CaseAdvisor [Carrick *et al*., 1999], like SiN, integrates conversational case retrieval with planning. While CaseAdvisor applies pre-stored hierarchical plans to gather information to solve diagnosis tasks, SiN instead uses its case retriever to gather information and applies cases to refine hierarchical plans.

Planning with incomplete information has been the subject of frequent research in planning (e.g., [Golden *et al*., 1996]). Typically, a distinction between sensing and planning actions is made, where the former involve queries to external information sources and the latter involves inferencing steps. This is comparable to querying using NaCoDAE and task refinement using SHOP.

10 Final Remarks

We presented the SiN algorithm for case-based HTN planning. SiN was motivated by three requirements for planning military operations: plans are hierarchical, there is no complete domain theory explaining all possible courses of action, and planners do not have complete information about the current situation.

Our work includes the following contributions:

- SIN is a provably correct algorithm for case-based planning with incomplete domain theories.
- SiN can tolerate incomplete world state information by representing preferences in the cases. Our experimental results show that a user can dynamically guide SiN by giving preferences to it as part of the user's normal interaction with SiN during the planning process.
- SiN provides a bridge between two classical approaches for case-based planning, in which cases either provide control knowledge or domain knowledge. In SiN, cases provide both kinds of knowledge.

- SiN's ability to combine both experiential and generative knowledge sources can be beneficial in real-world domains where some processes are well known and others are obscure but recorded memories exists on how they were performed. Planning for NEOs is a typical example of this type of domain. We have integrated SiN into HICAP [Muñoz-Avila *et al.*, 1999], a system designed to support these kinds of operations.

Our creation of SiN was made possible because of the similarity between NaCoDAE's cases and SHOP's methods. For our future work, we want to further exploit this similarity to ease knowledge acquisition for plan generation. To this end, we have started working to create algorithms for learning HTN methods automatically from cases.

Acknowledgements

The authors would like to thank the reviewers for their helpful comments. This research was supported by funding from DARPA, ONR, AFRL (F306029910013 and F30602-00-2-0505), ARL (DAAL0197K0135), and the University of Maryland General Research Board. Opinions on SiN expressed in this paper are those of the authors and do not necessarily reflect the opinions of the funding agencies.

References

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

[Breslow & Aha, 1997] Breslow, L. A. & Aha, D. W. NaCoDAE: Navy Conversational Decision Aids Environment (Technical Report AIC-97-018). NCARAI, Washington, DC, 1997.

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

[DOD, 1997]. *Joint tactics, techniques and procedures for noncombatant evacuation operations* (Joint Report 3-07.51). Washington, DC: Department of Defense, Joint Chiefs of Staff, 1997.

[Erol *et al.*, 1994] Erol, K., Nau, D., & Hendler, J. HTN planning: Complexity and expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 123-1128). Seattle, WA: AAAI Press, 1994.

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

[Ihrig & Kambhampati, 1994] Ihrig, L.H., & Kambhampati, S. Derivational replay for partial order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (992-997). Seattle, WA: AAAI Press, 1994.

[Golden *et al*., 1996] Golden, K., Etzioni, O., and D. Weld. "*Planning with execution and Incomplete Information*," Technical Report, Department of Computer Science, University of Washington, TR96-01-09, February, 1996.

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

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

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

[Muñoz-Avila & Weberskirch, 1996] Muñoz-Avila, H., & Weberskirch, F. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. *Proceedings of the Third International Conference on AI Planning Systems*. Edinburgh: AAAI Press, 1996.

[Nau *et. al.*, 2000] Nau, D., Aha, D.W., & Muñoz-Avila, H. Ordered task decomposition. In Y. Gil & K. Myers (Eds.) *Representational Issues for Real-World Planning Systems: Papers from the AAAI Workshop* (Technical Report WS-00-18). Menlo Park, CA: AAAI Press, 2000.

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

[Nau *et. al.*, 2001] Nau, D., Muñoz-Avila, H., Lotem, A., Mitchell, S. & Yue, C. Total-order planning with partially ordered subtasks. To appear in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*.

[Veloso, 1994] Veloso, M. *Planning and learning by analogical reasoning.* Berlin: Springer-Verlag, 1994.

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

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

A Distributed Case-Based Query Rewriting

Maurizio Panti Luca Spalazzi Loris Penserini
Istituto di Informatica, University of Ancona, via Brecce Bianche, 60131 Ancona, Italy
{panti,spalazzi,pense}@inform.unian.it

Abstract

In literature, the mediator architecture has been proposed for taking information from distributed, heterogeneous, and often dynamic sources and making them work together as a whole. In this paper we propose a distributed case-based approach for the main problem of a mediator, i.e. rewriting queries according to mediator's schema. According to this approach we use a case memory as mediator's schema. Therefore, such a schema is not static (as in other systems) but is dynamically updated through the cooperation with information sources and other mediators, strongly influenced by the queries submitted by a consumer. From the analysis of different cooperation strategies arises that it is more efficient and effective for a mediator to directly cooperate with information sources, when the sources are few. Otherwise, it is more efficient to cooperate with other mediators.

1 Introduction

Nowadays, information systems can be thought as collections of *information sources* and *information consumers* that are often *distributed* in world-wide networks. This means that sources and consumers can be autonomous and thus are often *heterogenous* and *dynamic*. Information sources are heterogenous due to discrepancies at the physical level (different DBMSs, software and hardware platforms), logical level (different data models), and conceptual level (different schemas, concept and relation names).

Example 1.1 Figure 1 shows an example of three information sources about articles on computer science and their authors. Let us suppose that the three sources receive the query $Q_0 \doteq \forall pub.ai$. w_3 can answer to Q_0 (it finds a view that matches with Q_0), whereas w_1 and w_2 can not. Indeed, the schema of w_1 is different (*heterogeneity of schemas*). Notice that, if we rewrite the query as $Q_0' \doteq \forall pub.\forall keyword.\{\text{“}AI''\}$, w_1 is able to answer since $\forall pub.\forall keyword.\{\text{“}AI''\}$ is part of its schemas. For what concerns w_3, it is not a matter of heterogeneity, w_3 simply does not have the required information.

Information consumers are heterogeneous as well, since each consumer may have different customization needs due to distinct business objectives. Moreover autonomous information sources are dynamic, since they may be added to the system, may become (temporarily or definitively) unavailable, or, sometimes, may also vary their conceptual schemas. Information consumers are also dynamic. Indeed new consumers can be inserted or removed. Furthermore, their information needs can change very often. Our work deals with information integration in distributed, heterogeneous, and dynamic information systems, i.e., constructing answers to query from consumers. Our primary goal consists on rewriting a consumer's query into queries to specific information sources. In [M. Panti, L. Spalazzi, A. Giretti, 2000] we proposed a distributed case-based approach to the problem of rewriting queries. According to this approach we have a dynamic mediated schema (the case memory) instead of a static one (as in other systems). Indeed, our mediator's schema is dynamically updated through the cooperation with information sources and other mediators, strongly influenced by the queries submitted by a consumer. In this paper we focus on the cooperation strategies of the distributed case-based reasoner. Indeed, we compare different strategies from a theoretical point of view and discuss the results. The paper is organized as follows. Section 2 reports the related work. An overview of our work is provided in Section 3. Section 4 summarizes results about local query rewriting (i.e. when a query can be reformulated by a single mediator). Section 5 describes the distributed query rewriting (i.e., when a query can be reformulated only through cooperation among mediators) and discuss some theoretical results. Finally, some conclusions are given in Section 6.

2 Related Work

Intelligent Information Integration (I^3). The most significative approach to information integration relies on the so called *mediator architecture* [Wiederhold, 1992]: an architecture with three-layers. A layer is devoted to information consumers, another layer is devoted to information sources, the middle layer (i.e. the mediator) deals with the reformulation of a given query in a set of queries, each targeted at a selected source. Intuitively, such a reformulation would be equivalent to the original query (i.e., denote the same set of instances). Nevertheless, often this is not possible. Therefore, a query

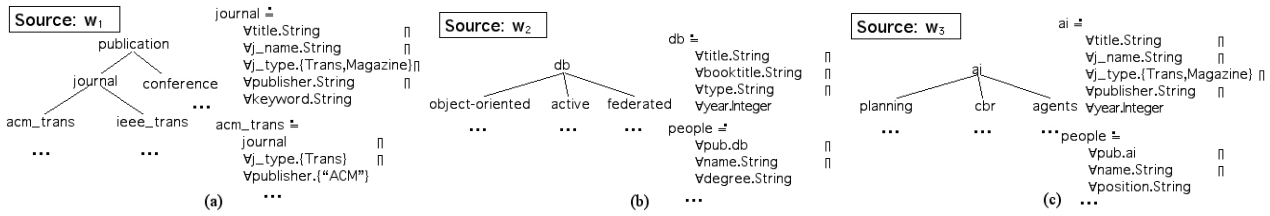


Figure 1: An example of information sources.

reformulation *contained in* (instead of *equivalent to*) the original query is considered adequate (see for example [Ullman, 1997; Beeri *et al.*, 1997]). Furthermore, when a *description logic* (e.g. [Borgida and Patel-Schneider, 1994]) is used as data modeling and query language (e.g. [Beeri *et al.*, 1997; Kirk *et al.*, 1995; Ullman, 1997]), the problem of query containment corresponds to the subsumption problem.

There are two basic approaches to intelligent information integration: the procedural approach and the declarative approach. In the *procedural* approach (e.g., TSIMMIS [Garcia-Molina *et al.*, 1997]), mediators integrate information from sources through ad-hoc procedures defined with respect to a set of predefined information needs. When such needs or sources change (i.e., we have a dynamic information system), a new mediator must be generated. In the *declarative* approach (e.g., SIMS [Arens *et al.*, 1993], Information Manifold [Kirk *et al.*, 1995], and Infomaster [Duschka and Genesereth, 1997]), mediators use suitable mechanisms to rewrite queries according to source descriptions. Usually, the proposed systems do not allow automatic adaptation of such descriptions when something changes.

Distributed Case-Based Reasoning (DCBR). A *distributed case-based reasoner* [Prasad *et al.*, 1996; Plaza *et al.*, 1997] learns by experience and cooperation how problems can be solved. Therefore, it is appropriate for distributed and dynamic application domains, when it is impossible to have predefined solutions. Nevertheless, as far as we know, its application to information integration is a novelty. In DCBR, there are several agents; each agent has its own case memory since it could have acquired its own independent problem-solving experiences. A new problem is solved through agent cooperation. Indeed each agent reuses the local past case that best contributes to the overall case. Description logic is applied to case-based reasoning (and thus DCBR) as well (e.g. [Koehler, 1994]). Indeed, in case-based reasoning, subsumption becomes a powerful tool to automatically derive case hierarchies that can be used in case retrieval and case retention.

3 Work Overview

Our goal is to build an information system capable of interconnecting information consumers and sources. Previous approaches solve several problems concerned with distributed, heterogeneous information systems. Their main limitation is related to the capability of evolving according to dynamic information systems. We propose (see also [M. Panti, L. Spalazzi, A. Giretti, 2000]) a multi-agent system which is based on a mediator architecture, i.e. mediators, sources, and consumers are agents. All the agents adopt a description logic called C–CLASSIC [Borgida and Patel-Schneider, 1994] as data modelling and query language. Indeed, subsumption in C–CLASSIC can be computed in polynomial time.

In our work, the capability of a **mediator** agent to face heterogeneous and dynamic systems relies on a thesaurus and a distributed case-based reasoner. Each mediator has its own *thesaurus* in order to solve name heterogeneity. A thesaurus is composed by a set of classes of synonyms. Each element of a class has the reference to information sources that use it as term. Every time a new query arrives, its terms are translated in standard terms by means of the thesaurus. The reverse process occurs when the rewritten query must be sent to information sources. The thesaurus must be dynamically updated when a change in the system occurs. The classes of the thesaurus are modified by means of clustering. The explanation of this technique is out of the scope of the paper, for more details we remind to [Diamantini and Panti, 1999]. Therefore, in the rest of the paper we assume that *no name heterogeneity occurs* (i.e., the thesaurus has a maximum precision). Each mediator has also its own *case-based reasoner* in order to solve heterogeneity of schemas and consumer's needs. Each case contains a query and the corresponding reformulation, and it is stored in the mediator's case memory. When a new query from the consumer arrives, the mediator looks for past queries subsumed by the new one and adapts the corresponding solutions in order to obtain a reformulation of the new query. This means that the mediated schema is strongly influenced by the queries submitted by consumers (mediator's experience). When this experience does not help, the mediator cooperates with other mediators and/or information sources. This approach allows the mediator to update its schemas and therefore to take into account changes in information sources and consumer's needs. This can be considered a kind of distributed case-based reasoning.

In our system, **information source** agents[1] are able to cooperate with mediators through a query reformulation based on local schemas. Indeed each source is able to find a view over its schema that is subsumed by an input query.

Example 3.1 Let us consider the distributed information system of Figure 1 (Example 1.1). According to the above assumption, we consider that all the (concept and role) names have been already translated by a thesaurus and thus no name heterogeneity occurs. Nevertheless, conceptual schemas are different (as showed in Example 1.1).

[1]Notice that when a source is a non-agent software, we need a wrapper as interface between the source and the rest of the system.

Query	Rewriting of the query	Information Sources
H	$(\forall pub.journal \sqcap \forall pub.db), (\forall pub.ai \sqcap \forall pub.db)$	$(\forall pub.journal, w_1), (\forall pub.db, w_2), (\forall pub.ai, w_3)$
I	$\forall pub.\forall publisher.\{\text{“}ACM\text{''}\}$	$(\forall pub.\forall publisher.\{\text{“}ACM\text{''}\}, w_1), (\forall pub.\forall publisher.\{\text{“}ACM\text{''}\}, w_3)$
J	$(\forall pub.\forall keyword\{\text{“}AI\text{''}\}), (\forall pub.ai)$	$(\forall pub.\forall keyword\{\text{“}AI\text{''}\}, w_1), (\forall pub.ai, w_3)$
K	$(\forall pub.\forall keyword.\{\text{“}Agents\text{''}\} \sqcap \forall pub.db)$, $(\forall pub.agents \sqcap \forall pub.db)$	$(\forall pub.\forall keyword.\{\text{“}Agents\text{''}\}, w_1), (\forall pub.db, w_2), (\forall pub.agents, w_3)$

Figure 2: An example of case memory.

Let us suppose that source w_1 receives a query $Q_1 \doteq \forall j\_type.\{Trans\}$. w_1 finds that Q_1 subsumes *acm_trans* and *ieee_trans* and thus returns the corresponding instances (denoted by $I_{w_1}(acm\_trans) \cup I_{w_1}(ieee\_trans)$).

4 Local Query Rewriting

Generally speaking, a case is an arbitrary set of features (attribute-value pairs). Some features are devoted to represent a problem (in our application the query to be reformulated) in order to make easier the retrieval of a past problem similar to the current situation. The rest of features are devoted to represent the problem solution (the reformulated query and information sources where the reformulated query has been sent) to reuse it in the current problem. Furthermore, the problem of rewriting queries has a fundamental condition that must be satisfied: *the rewritten query must be contained in the original query* [Beeri *et al.*, 1997; Ullman, 1997]. We use C–CLASSIC as a language for representing queries and thus cases. The subsumption relation is used as query containment and thus for case retrieval. According to above considerations, a **case** is a structure as the following: $\langle Q, Sol(Q), \langle (Q_1, w_1), \ldots (Q_n, w_n)\rangle\rangle$ where $w_1, \ldots, w_n$ are information sources and $Q_1, \ldots, Q_n$ are queries to $w_1, \ldots, w_n$ respectively. Q is a given query and $Sol(Q)$ is its reformulation, i.e., a set of arbitrary combinations of $Q_1, \ldots, Q_n$ such that each element of $Sol(Q)$ is subsumed by Q. A collection of cases is called **case memory**. A **terminology** (i.e., a set of concept definitions and their relations) is always associated to a case memory. $\top, \bot$ represent the *top* and *bottom* concepts, respectively. The initial case memory can be built as static mediated schemas of traditional mediators (e.g. SIMS [Arens *et al.*, 1993]). In our system, the added value is that the DCBR works to maintain updated such a schema.

Example 4.1 Figure 2 and Figure 3 show the case memory and the terminology of a mediator of Example 3.1.

Query Retrieval. Every time a new query arrives, it is inserted in the terminology and classified by means of subsumption. If there exists a past problem equal to the new query, then its solution can be used as solution for the new query. Otherwise, the solutions of past problems that are subsumed by the new query can be used as solution for the new problem. The closest to the new query the retrieved cases are, the best the solutions are. This drives us to the notion of problems maximally contained in the new query. This notion allows the definition of the retrieval function $M_{inf}(Q, \mathcal{P})$ as a set of conjunctions of concepts ($\sqcap_{i\geq 1} X_i$) of the given terminology ($\mathcal{P}$). For each conjunction of concepts there does not exist another conjunction of concepts of the same terminology ($\sqcap_{j\geq 1} Y_j$) such that $(\sqcap_{i\geq 1} X_i) \sqsubseteq (\sqcap_{j\geq 1} Y_j) \sqsubseteq Q$.

Example 4.2 Let us suppose to have $Q_2 \doteq \forall pub.(ai \sqcap db)$ as input query in Example 4.1. Applying the retrieval function we obtain: $M_{inf}(Q_2, \mathcal{P}) = \{K, H \sqcap J\}$

Query Reuse. The basic reuse algorithm ($\mathcal{S}_{inf}(Q, \mathcal{P})$) simply copies past solutions of past problems returned by the retrieval function.

Example 4.3 Applying the basic reuse algorithm to Example 4.2, we obtain: $\mathcal{S}_{inf}(Q_2, \mathcal{P}) = Sol(K) \cup (Sol(H) \sqcap Sol(J))$[2]

Now we have all the building blocks to define the local reuse algorithm ($\mathcal{D}(Q, \mathcal{P})$). Such an algorithm decomposes each consumer's query in basic components, applies the basic reuse algorithm to each component, and finally assembles the corresponding results (a more detailed description can be found in [M. Panti, L. Spalazzi, A. Giretti, 2000]).

Example 4.4 Let us suppose to have $Q_3 \doteq \forall pub.ai \sqcap \forall pub.acm$ as input query in Example 4.1. Applying the reuse algorithm we obtain:

$$\begin{aligned}\mathcal{D}(Q_3, \mathcal{P}) &= \mathcal{S}_{inf}(\forall pub.ai, \mathcal{P}) \sqcap \mathcal{S}_{inf}(\forall pub.acm, \mathcal{P}) \\ &= (Sol(K) \sqcap Sol(I)) \cup (Sol(H) \sqcap Sol(J) \sqcap\ Sol(I))\end{aligned}$$

Notice that this is just one possible answer, another mediator with a different case memory may rewrite the query in a different way and thus return a different answer.

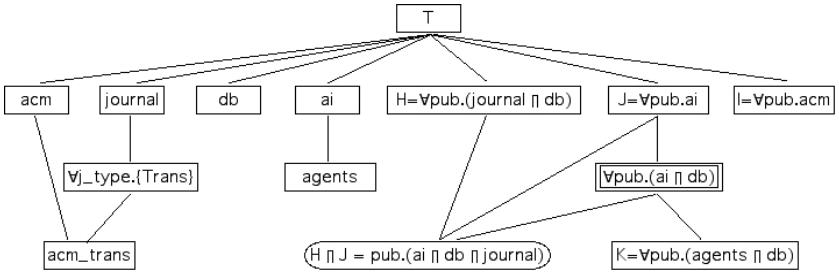
It is easy to prove that each element of $\mathcal{S}_{inf}(Q, \mathcal{P})$ and $\mathcal{D}(Q, \mathcal{P})$ is subsumed by Q. Therefore, a reformulation is contained in the original query as required in the definition of case. Finally, notice that in general $\mathcal{S}_{inf}(Q, \mathcal{P}) \neq \mathcal{D}(Q, \mathcal{P})$.
Query Evaluation. When the mediator has a reformulation of the received query, it must send such a reformulation to related information sources and wait for their answers.

Example 4.5 The evaluation of $Sol(H)$ consists on querying sources w_1, w_2, and w_3 with $\forall pub.journal$, $\forall pub.db$, and $\forall pub.ai$ respectively, and integrating the related answers. The semantics of concepts helps us on integrating the answers. Let $I_{w_1}(\forall pub.journal)$, $I_{w_2}(\forall pub.db)$, and $I_{w_3}(\forall pub.ai)$ be answers from w_1, w_2, and w_3 respectively, then the answer to $\forall pub.(journal \sqcap db)$ is: $(I_{w_1}(\forall pub.journal) \cap I_{w_2}(\forall pub.db)) \cup (I_{w_3}(\forall pub.ai) \cap I_{w_2}(\forall pub.db))$.

Failures. We have a *local failure in query reuse* when a mediator is not able to rewrite a given query Q, i.e., $Sol(Q) = \bot$[3]. This means that the mediator's case memory contains no past cases that can be used to reformulate Q. Usually this is due to the fact that it is the first time that the consumer formulates such a query, i.e., the consumer has a new information need.

[2] $Sol(H) \sqcap Sol(J)$ means that each element of $Sol(H)$ is conjuncted with each element of $Sol(J)$.

[3] Notice that $I(\bot) = \emptyset$.



$\{\ldots journal \dot{\leq} \top\ ,\ db \dot{\leq} \top\ ,\ ai \dot{\leq} \top\ ,\ agents \dot{\leq} ai\ ,\ acm \doteq \forall publisher.\{\text{“}ACM\text{”}\}\ ,\ acm\_trans \doteq acm \sqcap \forall j\_type.\{Trans\}\ ,$
$H \doteq \forall pub.(journal \sqcap db)\ ,\ I \doteq \forall pub.acm\ ,\ J \doteq \forall pub.ai\ ,\ K \doteq \forall pub.(agents \sqcap db)\ \ldots\}$

Figure 3: An example of terminology.

Example 4.6 Let us consider the case memory of Example 4.1 and the query $Q_4 \doteq \forall pub.ai \sqcap \forall affiliation.\{\text{“}Stanford\text{”}\}$. The mediator is not able to rewrite the query. Indeed, we have

$$\mathcal{D}(Q_4, \mathcal{P}) = (Sol(K) \sqcap \bot) \cup (Sol(H) \sqcap Sol(J) \sqcap \bot) = \{\bot\}$$

Furthermore, we have a *local failure in query evaluation* when a mediator sends a rewritten query to related sources and receives at least an empty answer. This means that the case memory of the mediator is not updated. Typically, an information source has been removed from the system or changed its schema.

Example 4.7 Let us suppose that source w_1 has been (perhaps temporarily) removed, then the evaluation of the reformulated query in Example 4.4 fails. Indeed $I_{w_1}(\forall pub.\forall keyword.\{\text{“}Agents\text{”}\}) = I_{w_1}(\forall pub.journal) = I_{w_1}(\forall pub.\forall publisher.\{\text{“}ACM\text{”}\}) = I_{w_1}(\forall pub.\forall keyword.\{\text{“}AI\text{”}\}) = \emptyset$

5 Distributed Query Rewriting

Even if a local failure (in query reuse or evaluation) occurs, the distributed case-based reasoning has still a possibility of solving the problem by means of cooperation with other mediators and sources. Notice that this provides the system the most important way of learning. Indeed, if the query is reformulated, the new rewritten query and/or the new sources can be stored as a new case. In such a way the mediator can support dynamic information systems. Furthermore, it does not need to maintain consistent its case memory every time something changes, but only when a consumer sends a query that fails. Therefore, the distributed information system is not overloaded with consistency maintenance operations, that usually are time consuming. On the contrary, traditional mediation systems can only update mediated schema with expensive hand-made procedures.

Cooperation strategies can be classified according to three parameters: partners, queries, and answers. In the following we describe such strategies, we also analyse them (from a theoretical point of view), since the choice of the right cooperation strategy is crucial for effectiveness and efficiency, and discuss the results.

Partners. When a mediator (say M) fails, it can cooperate with other *mediators* asking them to rewrite a query according to their own case memories. If they succeed, M can store the result as a new case in its case memory. The mediators involved in this strategy can be all the mediators of the system, just mediators that have never been in touch with M, or mediators that successfully cooperated with M in the past. The mediator M can directly cooperate with *information sources* as well. This strategy can involve all the sources (it is very expensive), sources responsible of the local failure (this strategy takes into account changes of schemas), recently added sources (this strategy takes into account new sources), or the sources that successfully cooperated with M in the past (when a user has a new information need, the application of this strategy supposes that the new need involves the same sources of the past). Each source classifies the query and returns the subsumed concepts. Let us analyse such strategies focusing on recall and precision ratios in order to evaluate the effectiveness of the strategy. We also evaluate the information redundancy that such strategies produce.

First, let us consider a mediator M that cooperates with other mediators, sends them the original query, and asks for receiving a rewritten query. This strategy has a possible disadvantage that another mediator can send a wrong rewriting. This fact has harmful consequences on the precision, as stated by the following theorem.

Theorem 5.1 *Let $S_1, \ldots, S_n$ be n information sources. Let $\mathcal{V}$ be a view of $S_1, \ldots, S_n$. $\mathcal{V}$ is represented as a case memory that does not change. Let M, N be two mediators such that M interacts with N when M fails. Let $\mathcal{C}_n(M)$ be the case memory of M after n interactions with N. Then*

Recall:	$\lim_{n\to\infty} \frac{card(\mathcal{C}_n(M) \cap \mathcal{V})}{card(\mathcal{V})}$	≤ 1
Precision:	$\lim_{n\to\infty} \frac{card(\mathcal{C}_n(M) \cap \mathcal{V})}{card(\mathcal{C}_n(M))}$	*is indeterminate*

In the above theorem, $\mathcal{C}_n(M)$ denotes the case memory of the mediator M (e.g. see Figure 2). $\mathcal{V}$ denotes the information need of a given consumer, i.e., it is the schema to which $\mathcal{C}_n(M)$ must converge. Such a theorem states that when the mediator N sends a rewriting that is part of $\mathcal{V}$ the recall and the precision grow, otherwise the recall does not change and the precision decreases. This means that is crucial the choice of what and how many mediators to cooperate with. Another possible disadvantage of such a strategy is that asymptotically all the mediators may have the same case memory, i.e. this strategy produces a certain redundancy. This result is a consequence of the following theorem.

Theorem 5.2 *Let M, N be two mediators such that M interacts with N when M fails. Let $\mathcal{C}_n(M)$ be the case memory of M after n interactions with N. Let $\mathcal{C}(N)$ be the case memory of N such that it does not change while N interacts with M. Then*

$$\lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{C}(N))}{card(\mathcal{C}(N))} = 1$$

The above theorem states that the case memory of M converges to the case memory of N when M cooperates with N. This seems to suggest that when the mediator cooperates with other mediators it would directly ask data instead of the rewritten query. This would allow any mediator to maintain its expertise and avoid redundancy.

As a second strategy, let us consider a mediator M that directly cooperates with information sources, sends them the original query, and receives the rewritten query (and eventually the data). This strategy guarantees a given consumer that the mediator M converges to the consumer's information need. Indeed it is possible to prove the following theorem.

Theorem 5.3 *Let $S_1, \ldots, S_n$ be n information sources. Let $\mathcal{V}$ be a view of $S_1, \ldots, S_n$. $\mathcal{V}$ is represented as a case memory that does not change. Let M be a mediator such that M interacts with $S_1, \ldots, S_n$ when it fails. Let $\mathcal{C}_n(M)$ be the case memory of M after n interactions with $S_1, \ldots, S_n$. Then*

$$\textit{Recall:} \quad \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{V})}{card(\mathcal{V})} = 1$$

$$\textit{Precision:} \quad \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{V})}{card(\mathcal{C}_n(M))} = 1$$

The above theorem states that the mediator will asymptotically satisfy the information need of a given consumer (denoted by $\mathcal{V}$). Indeed, under the hypothesis that no name heterogeneity occurs (see Section 3), it is impossible that the source has wrong schemas. Therefore, the information source has either the right answer or no answer at all. This means that the choice of sources to cooperate with is crucial only for efficiency of the system, i.e. how fast the mediator's case memory converges to user's needs.

Example 5.1 In the situation of Example 4.7, let us suppose that the mediator chooses to look for new sources and cooperate with them. Let us suppose that w_4 is a new information source which contains ACM publications and has the following concepts in its schema:

$$acm\_tods \doteq \forall j\_name.\{\text{“}TODS''\} \sqcap \forall publisher.\{\text{“}ACM\text{”}\}$$
$$acm\_tocl \doteq \forall j\_name.\{\text{“}TOCL''\} \sqcap \forall publisher.\{\text{“}ACM\text{”}\}$$
$$acm\_tois \doteq \forall j\_name.\{\text{“}TOIS''\} \sqcap \forall publisher.\{\text{“}ACM\text{”}\}$$

The mediator decomposes the query Q_3 and sends each component to w_4. For example, when w_4 receives $\forall pub.\forall publisher.\{\text{“}ACM\text{”}\}$, it looks for maximally contained concepts and obtains $M_{inf}(\forall pub.\forall publisher.\{\text{“}ACM\text{”}\}, \mathcal{P}) = \{\forall pub.acm\_tods, \forall pub.acm\_tocl, \forall pub.acm\_tois\}$. When the mediator receives the answer, retains as new case:

$$\langle\ \forall pub.acm\ ,\ \langle \forall pub.acm\_tods\ ,\ \forall pub.acm\_tocl\ ,\ \forall pub.acm\_tois\rangle\ ,\ \langle\ (\forall pub.acm\_tods, w_4)\ ,\ (\forall pub.acm\_tocl, w_4)\ ,\ (\forall pub.acm\_tois, w_4)\ \rangle\ \rangle$$

Queries. Cooperation strategies can also be classified according to queries. Indeed M can send the query as received by the consumer (this is a strategy that takes into account the new user's need). It can send the retrieved query (if any), i.e. the problem returned by the retrieval function (this is a strategy that takes into account the need of maintaining consistent the mediator's case memory). Finally, the mediator can send the reformulated query (if any). In the latter case the “approximation of the solution grows”, as stated by the following theorem.

Theorem 5.4 *Let Q be the original query. Let $Q' \in M_{inf}(Q, \mathcal{P})$ be one of the past queries retrieved by M. Let $Sol_M(Q')$ be one of the reformulations of Q' used by M as rewriting of Q. Let N be the mediator or source that cooperates with M. Let $Sol_N(x)$ be the query rewritten by N if N receives x as input query. Then*

$$Sol_N(Q) \sqsubseteq Q \quad , \quad Sol_N(Q') \sqsubseteq Q' \sqsubseteq Q$$
$$Sol_N(Sol_M(Q')) \sqsubseteq Sol_M(Q') \sqsubseteq Q' \sqsubseteq Q$$

As a consequence, the solution returned by N is more distant from Q than Q' when M does not send the original query.

Cooperation strategies can also be classified depending on the fact that the query can be sent as it is or be decomposed in basic components. This second strategy is applicable since it is straightforward to prove that the following theorem holds when we use C-CLASSIC:

Theorem 5.5 *Let $Q_1, \ldots, Q_n$ be queries. Let $Q = f(Q_1, \ldots, Q_n)$ be the user's query. Let N be the mediator or source that cooperates with M. Let $Sol_N(x)$ be the query rewritten by N if N receives x as input query. Then*

$$Sol_N(f(Q_1, \ldots, Q_n)) = f(Sol_N(Q_1), \ldots, Sol_N(Q_n))$$

The theorem above states that M does not need to decompose a query before its sending, since the query is directly decomposed by partners. This suggests that decomposition is convenient (efficient) only if M failed to reformulate / evaluate just some components of the query.

Answers. Finally, cooperation strategies can be classified according to answers. M can ask for rewriting the query. Its goal is to update its case memory with a rewritten query obtained from its collaborators. M can also ask for data that answer the query. In such a situation, it stores in the case memory the addresses of the mediators/sources that answered. We can analyse such strategies depending on partners. Table 1 reports the results of cooperation with only one agent. We assume that such an agent (say a source S or a mediator N) does not change while the strategy is in progress. We also suppose that N has $\langle Q, Sol_S(Q), S\rangle$ in its case memory. For each strategy, Table 1 reports the new case that M stores in its case memory, the set of instances related to the reformulation, the number of messages to obtain instances of Q (when M receives Q the first time and the next time), and the results about redundancy (see Theorem 5.2). When we consider cooperation with mediators, it seems more efficient to ask for a rewriting, since M interacts with N only the first time, the next time M directly interacts with S. On the other hand, the strategy of requiring data does not produce redundancy, even if it is less efficient. Notice that, for what concerns such evaluation parameters, the strategy of cooperating

Partner	Answer	New Case	Data	No. of messages.		Redundancy
				1st. time	Next time	
Source (S)	Rewriting	$\langle Q, Sol_S(Q), S\rangle$	$I_S(Sol_S(Q))$	4	2	No
"	Data	$\langle Q, Q, S\rangle$	$I_S(Sol_S(Q))$	2	2	No
Mediator (N)	Rewriting	$\langle Q, Sol_S(Q), S\rangle$	$I_S(Sol_S(Q))$	4	2	Yes
"	Data	$\langle Q, Q, N\rangle$	$I_S(Sol_S(Q))$	4	4	No

Table 1: A comparison of different cooperation strategies.

with sources requiring data seems to be the most appropriate when the number of sources is comparable to the number of mediators.

6 Conclusions

We considered the problem of rewriting queries by means of distributed case-based reasoning. As far as we know, this is a novel approach. This allows us to have dynamic mediated schemas instead of static ones. We showed that a real dynamic mediator is obtained by means of distributed query reformulation, i.e., cooperation with other mediators and sources. Thanks to this approach, the mediator is able to be updated when sources and consumers are added/removed or change their schemas/needs. This operation is performed only when a consumer sends a query that fails and not every time something changes. This allows us to avoid of overloading the distributed information system with expensive consistency maintenance operations. In distributed case-based reasoning, the choice of the right cooperation strategy is crucial. We hinted several possible strategies and discussed their advantages. From such an analysis arises that we can obtain good results for effectiveness and efficiency when a mediator directly cooperates with information sources when the number of information sources is small. Otherwise, the most efficient strategy is the cooperation with other mediators. Finally, notice that the choice of C–CLASSIC is appropriate since the algorithm complexity for query retrieval and reuse is polynomial.

Acknowledgements

We thank J. Mylopoulos for suggestions and useful discussions about the topics presented in this paper. We also thank P. Coupey and S. Gianfelici that have participated to the development of the C–CLASSIC reasoner.

References

[Arens *et al.*, 1993] V. Arens, C. Y. Chee, C-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[Beeri *et al.*, 1997] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting Queries Using Views in Description Logics. In *Proc. of the 16th ACM Symposium on Principles of Database Systems (PODS)*, pages 99–108, New York, NY, May 12–14, Tucson, Arizona 1997. ACM Press.

[Borgida and Patel-Schneider, 1994] A. Borgida and P. F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994.

[Diamantini and Panti, 1999] C. Diamantini and M. Panti. A Conceptual Indexing Method for Content-Based Retrieval. In A. M. Tjoa, A. Cammelli, and R. R. Wagner, editors, *Tenth International Workshop on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, 1–3 September 1999. IEEE Computer Society.

[Duschka and Genesereth, 1997] O. M. Duschka and M. R. Genesereth. Infomaster - An Information Integration Tool. In *Proc. of the International Workshop on Intelligent Information Integration*, Freiburg, Germany, September 1997.

[Garcia-Molina *et al.*, 1997] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, 8:117–132, 1997.

[Kirk *et al.*, 1995] T. A. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *AAAI Spring Symposium on Information Gathering*. AAAI, 1995.

[Koehler, 1994] J. Koehler. An Application of Terminological Logics to Case-based Reasoning. In P. Torasso, J. Doyle, and E. Sandewall, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 351–362, Bonn, Germany, May 1994. Morgan Kaufmann.

[M. Panti, L. Spalazzi, A. Giretti, 2000] M. Panti, L. Spalazzi, A. Giretti. A case-based approach to information integration. In *Proceedings of the 26th International Conference on Very Large Databases*, Cairo, Egypt, 10–14 September 2000.

[Plaza *et al.*, 1997] E. Plaza, J. L. Arcos, and F. Martín. Cooperative Case-Based Reasoning. In G. Weiss, editor, *Distributed Artificial Intelligence meets Machine Learning*, Lecture Notes in Artificial Intelligence, Berlin, 1997. Springer Verlag.

[Prasad *et al.*, 1996] M. V. N. Prasad, V. R. Lesser, and S. E. Lander. Retrieval and reasoning in distributed case bases. *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries*, 7(1):74–87, March 1996.

[Ullman, 1997] J. D. Ullman. Information Integration Using Logical Views. In *Proceedings of the International Conference on Database Theory*, pages 19–40, 1997.

[Wiederhold, 1992] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer Magazine*, 25:38–49, March 1992.

Using Case-Base Data to Learn Adaptation Knowledge for Design*

Jacek Jarmulak and **Susan Craw**
The Robert Gordon University
Aberdeen, AB25 1HG, Scotland, UK
s.craw@scms.rgu.ac.uk

Ray Rowe
AstraZeneca, Silk Road Business Park
Macclesfield SK10 2NA, UK

Abstract

One advantage of Case-Based Reasoning (CBR) is the relative ease of constructing and maintaining CBR systems, especially as a number of commercial CBR tools are available. However, there are areas of CBR that current tools have not yet addressed. One of these is easing or automating the acquisition of adaptation knowledge. Since tasks like design or planning typically require a significant amount of adaptation, CBR systems for these tasks still do not fully benefit from CBR's promise of reducing the development effort. To address this, we have developed several "knowledge-light" methods for learning adaptation knowledge from the cases in the case-base. These methods perform substitutional adaptation, for both nominal and numerical values, and are suitable for decomposable design problems, in particular formulation and configuration. Tests performed on a tablet formulation domain show promising results. The automatic adaptation methods we present can easily be incorporated in general-purpose CBR tools, thus further contributing to reducing the cost of CBR systems.

1 Introduction

The popularity of Case-Based Reasoning (CBR) can be attributed to a number of factors. One of these is CBR's ability to cope with ill-defined problem domains by basing solutions to new problems on solutions that worked in the past. Given a new problem, a CBR system retrieves a set of similar problems from a case-base of previously solved problems. The retrieved problems provide ballpark solutions to the new problem; if needed, they can be adapted to fit the new problem better. This final solution can be reviewed and stored in the case-base to *improve* future performance if appropriate. This ability to learn is an important advantage of CBR.

A further advantage of CBR is the relative ease of constructing and maintaining CBR systems, especially as a number of commercial CBR tools are available. However, this advantage applies mainly to CBR systems *relying* on retrieval for finding solutions, e.g., many systems for classification and estimation tasks. This is because current CBR tools concentrate only on certain aspects of building a CBR system: automating construction of the case-base, defining retrieval, and implementing the user interface. Although the tools usually also provide some basic facilities for coding adaptation rules, they do not provide means of automating the task of acquiring adaptation knowledge. This means that CBR systems for more complex tasks, like planning or design, can be developed relatively easily only if their need for automatic adaptation can be eliminated (or reduced) by either having a large number of cases or by using the system in a decision-supporting role where adaptation is performed by an expert.

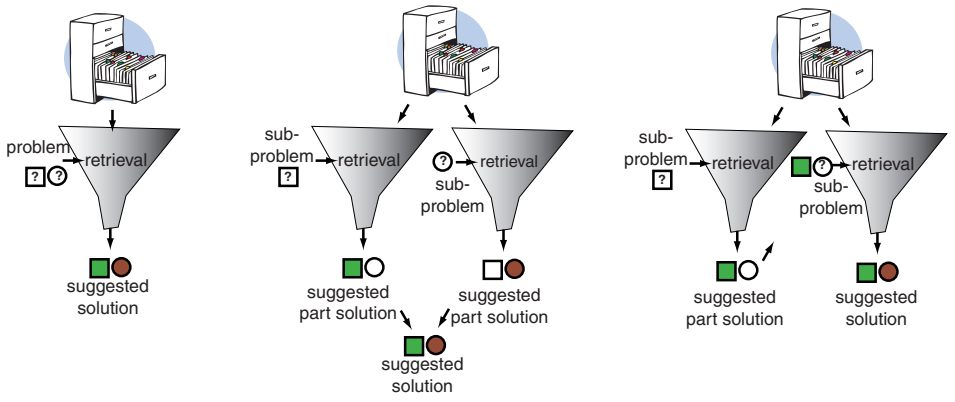
Recent developments in CBR tools seem to have concentrated on increasing their possible field of application: improving the way they interface with data-base systems, facilitating their deployment in web environments, increasing portability, and improving case representation. However, so far, commercial systems provide few facilities for automating the process of constructing competent case-bases, optimising retrieval with the goal of retrieving most useful cases, and, in particular, automating the acquisition of adaptation knowledge. These issues are now being addressed in recent research: Smyth and McKenna [1999] aim to build compact competent case-bases, Jarmulak *et al.* [2000] introduce ways of automatically optimising retrieval, and Wilke *et al.* [1997] consider knowledge-light approaches for learning adaptation suitable for incorporation into CBR tools.

In this paper we present a selection of generic methods suitable for automating the acquisition of adaptation knowledge for configuration and formulation tasks, a subclass of design tasks. We begin by placing our work in the context of related research in CBR. Then, we describe the methods that we have implemented for automated acquisition of adaptation knowledge from cases in the case-base, and show how the acquired knowledge is used in adaptation. After that, we present our test domain and the results of our experiments, before drawing conclusions and discussing further research.

2 Context & Related Work

We are interested in automating the acquisition of adaptation knowledge for case-based design, in particular for configuration or formulation tasks, where the final solution consists of a more-or-less constant number of components with relatively limited interactions between them. The solution is

*This work is supported by EPSRC grant GR/L98015.



a) single retrieval b) simultaneous multiple retrieval c) sequential retrieval

Figure 1: Solving a decomposable configuration problem.

usually determined by finding the correct "fillers" for components, as well as deciding the values of "filler" parameters. Examples of this type of task are: tablet formulation [Craw *et al.*, 1998], formulation of rubber compounds for tires [Bandini and Manzoni, 2000], and PC configuration [Stahl and Bergmann, 2000].

In general, finding a solution for a design problem is made more tractable if the task can be decomposed. Figure 1 contrasts a) a single step retrieval with two multi-step retrievals where, either b) multiple retrievals are merged, or c) the solution retrieved in one cycle informs the retrieval in the next cycle. Decomposition has special advantages for CBR design, because it allows the use of multiple cases. This results in better problem space coverage, and is important for the sparse case-bases typical of CBR design systems. This is demonstrated in the CBR systems Déjà Vu [Smyth and Cunningham, 1992] and EADOCS [Netten and Vingerhoeds, 1996]. Configuration and formulation tasks are in particular suitable for solving by decomposition; e.g., our CBRTFS uses decomposition for tablet formulation [Craw *et al.*, 1998]. CBRTFS makes use of the fact that the interaction between solution components is not strong, and that there is a difference in the degree to which the choice of the different components is subject to constraints. Thus it performs a sequential retrieval/adaptation (Figure 1c), proceeding from the most constrained to the least constrained components. Using the solutions to "earlier" subproblems, when searching for solutions to "later" subproblems, achieves good results, while keeping the complexity manageable.

The difficulty of acquiring adaptation knowledge was identified early in CBR research [Kolodner, 1991]. However, overall, little research has been done on automating acquisition or learning adaptation knowledge. Hanney and Keane [1997] have implemented a CBR system that estimates property values, and learns adaptation rules from cases stored in its case-base. DIAL [Leake *et al.*, 1996] also learns adaptation knowledge, in this case for a CBR planning system. It stores previous successful adaptation strategies that combine domain independent abstract adaptation rules with search procedures for domain specific information.

There are simple adaptation methods suitable for adjusting a single numeric solution. These methods make use of lazy learning to acquire their adaptation knowledge, i.e., at the point of retrieval they look at neighbouring cases to determine the amount of adaptation required. The simplest such method is weighted voting. Another example of simple adaptation is the difference heuristic of McSherry [1998]. This heuristic compensates for differences in feature values between the problem and the best matching case by searching for case pairs with the same difference in corresponding features. The major restriction of this method is the assumption that the estimated value is approximated well by an additive function. This severely limits its application area.

Wilke *et al.* [1997] present a framework for learning adaptation knowledge using *knowledge-light* approaches. Simple algorithms, like optimisation or inductive learning, can make use of the knowledge contained in the case-base, the similarity measure, and possibly already existing adaptation knowledge, to learn or improve adaptation knowledge. Wilke *et al.* suggest that one can consider optimisation of the k parameter in weighted voting as an example of learning adaptation knowledge; we view this as learning retrieval knowledge [Jarmulak *et al.*, 2000]. Such knowledge-light methods can not only be used for estimation or other simple CBR tasks, but can also be usefully applied for design tasks, especially if the design can be decomposed into subtasks, as is often the case with configuration or formulation.

Because we are interested in methods of learning adaptation knowledge that are suitable for implementation in general-purpose CBR tools, simpler, knowledge-light methods, like those described by Hanney and Keane and Wilke *et al.* seem more suitable, as opposed to the more powerful, but also more complex, methods used in DIAL. In the following, we use several knowledge-light approaches to acquire adaptation knowledge for a formulation task.

3 Adaptation

Adaptation methods in CBR are typically classified in the following 3 categories: substitutional adaptation substitutes or modifies a parameter in the solution; transformational adap-

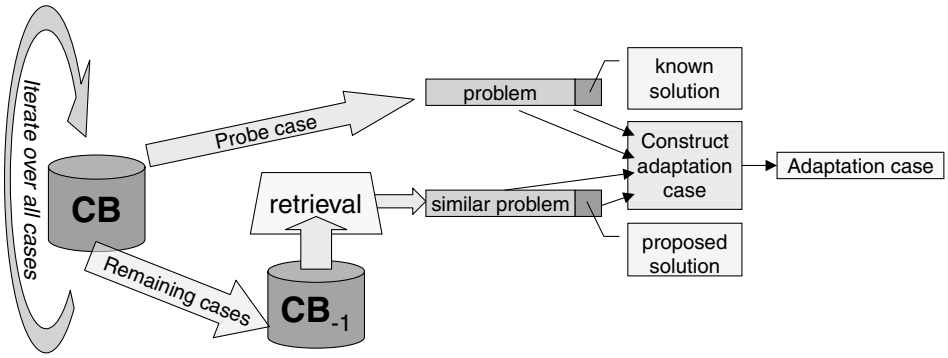


Figure 2: Constructing an adaptation case.

tation involves structural changes to the solution; and generative adaptation redoes the reasoning process that led to a solution [Smyth and Cunningham, 1993]. In this paper we are interested in *substitutional* adaptation, which lends itself best for automatic knowledge acquisition. The other two methods, often used in design tasks, present more difficulties as far as knowledge acquisition is concerned. However, a decomposable design task allows the adaptation to be decomposed into simpler adaptation subtasks, often suitable for substitutional types of adaptation.

3.1 Learning Adaptation Knowledge

The adaptation learning methods we present extract adaptation knowledge from the case data. We propose methods suitable for predicting values for nominal as well as numeric features; we call these targets. Both are needed in configuration or formulation tasks where often the nominal-valued types of components are determined first, and then their numeric parameters are estimated.

Constructing Adaptation Cases

One method we use to extract knowledge that is then used for adaptation, performs leave-one-out retrieval experiments and constructs "adaptation cases". Figure 2 illustrates how each of the cases is used as a probe case. After retrieving the most-similar case(s) from the reduced case-base, the probe and similar cases are analysed to construct an adaptation case that will be used to adapt the *proposed solution* from the similar case into the correct *known solution* for the probe case.

The way that adaptation cases for numeric targets are constructed is illustrated in Figure 3. We store differences between the problem features of the probe and retrieved cases, and the difference between their solutions. This is sufficient if we expect a uniform dependency between the solution difference and the problem differences over the whole problem space. If, however, the dependency differs in various parts of the problem space, then it is advisable to include also the problem features of the probe and the proposed solution from the retrieved case, as shown in Figure 3.

For nominal targets our adaptation cases are used to predict whether the solution to a probe case proposed by a retrieved case is a good solution. The construction of an adaptation case for nominal targets is illustrated in Figure 4. Firstly, the

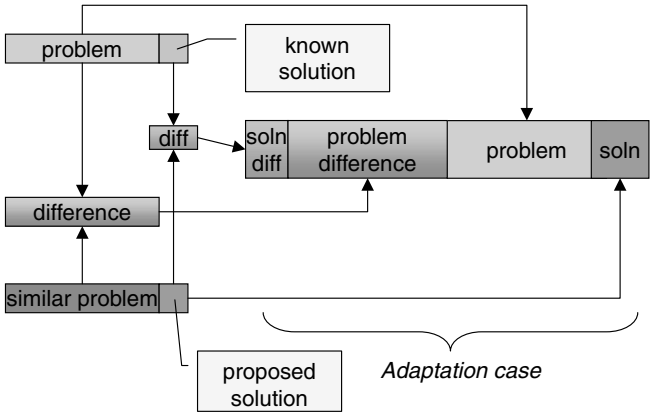


Figure 3: Adaptation cases for numeric targets.

adaptation case must contain data describing the problem features of the probe and the solution proposed by the retrieved case. Because we are dealing with a design task, it may also be possible to include two other types of data in the adaptation case. One type of information is an estimate (`est`) of important properties of the final design that make use of the proposed solution. We estimate these properties using CBR_E, a CBR system with the same case-data but with a different target and a more instantiated probe. The final type of information that can be included in the adaptation case is simpler to obtain. Given the predicted solution we determine feature(s) from the problem description (`rel`) which are related to that particular solution. An example from the tablet formulation domain would be chemical stability of a drug (part of problem description) with the chosen component (proposed solution).

Constructing Preference-Profiles

In the method of constructing adaptation cases shown in Figure 4, we estimated values of some design properties (`est`) and found some features (`rel`) related to the proposed solution. When placed in the adaptation case they are used to determine the correctness of the solution. But we can also use these values without constructing adaptation cases. We assume that the values of `est` & `rel` actually occurring in cases correspond to an acceptable range of values for designs,

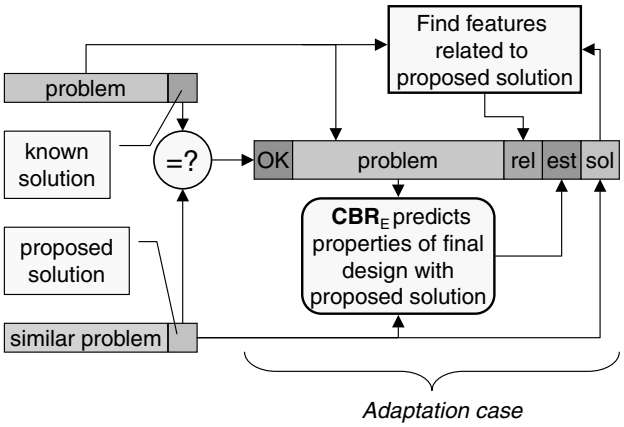


Figure 4: Adaptation cases for nominal targets.

and construct "preference profiles" for these features which reflect the distribution of `est` & `rel` in the case data. Figure 5 shows an example of a profile for the estimated feature tYP: a "frequency distribution" of the values of tYP in the case-data. Such profiles can be also interpreted as membership functions for a fuzzy concept "preference". Profiles, once constructed, are then used to estimate the "quality" of suggested solutions. The assumption is that values occurring often in the actual case-data are preferable. Looking at the profile in Figure 5, suppose two nearest neighbours, whose similarity to the problem case are virtually equal, have different solutions, one resulting in an estimated tYP of 130, and the other with equal to 530. We would choose the first solution with the higher preference tYP.

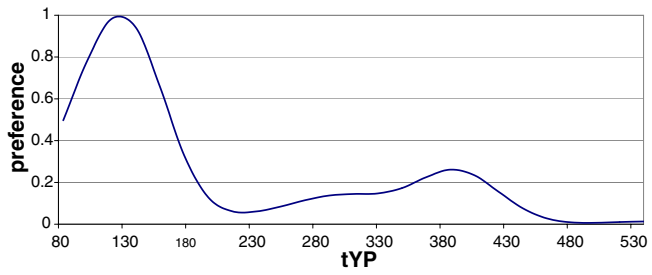


Figure 5: Preference profile.

3.2 Applying Adaptation Knowledge

Now we describe how adaptation cases or preference profiles are used in adaptation. Adaptation cases are applied in a CBR system to determine the adaptation actions. Another possibility would be to induce decision trees to decide the correctness of a nominal solution or to suggest an approximate adjustment for a numeric solution. We could also use other methods, e.g., induce adaptation rules [Hanney and Keane, 1997].

Numeric Targets

Adapting values of numeric targets, is conceptually simple, Figure 6. Given the differences between the problem and the retrieved case(s) we use CBR_A, a CBR system with adaptation cases like Figure 3, to predict adjustments to the proposed solutions. We can use just a single suggested correction, or k updates from the nearest neighbours or several CBR_A systems and combine the adjustments.

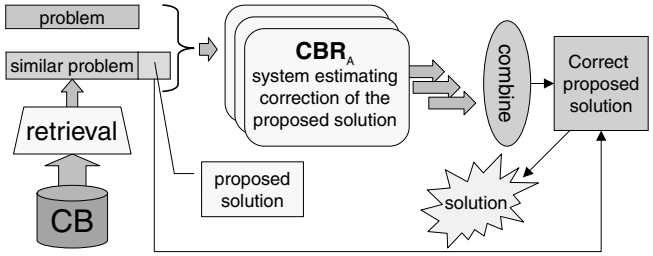


Figure 6: Problem differences to adapt numeric solutions.

Nominal Targets

Figure 7 shows how we can use information about the estimated correctness/quality of the solution in adaptation. The working cycle of the presented system is as follows: (1) given a new problem we retrieve cases from the case-base, these cases give us suggested solutions; (2) we estimate the correctness of the suggested solutions applied to our problem; (3) if the estimate of correctness suggest a high possibility of the solution being correct we return this solution; (4) otherwise, we store this solution with its correctness estimate, and (5) repeat retrieval, but this time ignoring all cases with the solution values which have already been tried out; (6) if all possible solutions have been tried without finding a really good one we will return the one that had the highest estimated correctness, giving preference to the solutions that were retrieved first (the nearest neighbours of the problem case). This approach might look like a search, however, the aspect distinguishing it is the order in which the solutions are tried – this order is determined by case similarity as defined by retrieval.

In this "iterative-adaptation" method the quality of a solution can be estimated using preference profiles; PPROF in Figure 7. To use the profiles, first, we have to obtain values of `est` & `rel` as we did to construct nominal adaptation cases in Figure 4. If several profiles are used then their values can be combined using a weighted sum. If we consider "preference profiles" as fuzzy membership functions then their values can be combined using a fuzzy AND operator, e.g., a minimum or a product.

Figure 7 also shows solution correctness being estimated using a CBR_A system containing adaptation cases like those in Figure 4, instead of PPROF. In this method, we use a combination of the case similarity and the accuracy of the leaves of the case-base index to give us a numeric estimate of the correctness of the solution.

4 Experiments and Results

We have tested these adaptation methods on a simple design task – tablet formulation [Craw *et al.*, 1998]. The design of a new tablet involves identifying inert substances called *excipients* to balance the properties of the drug so that the

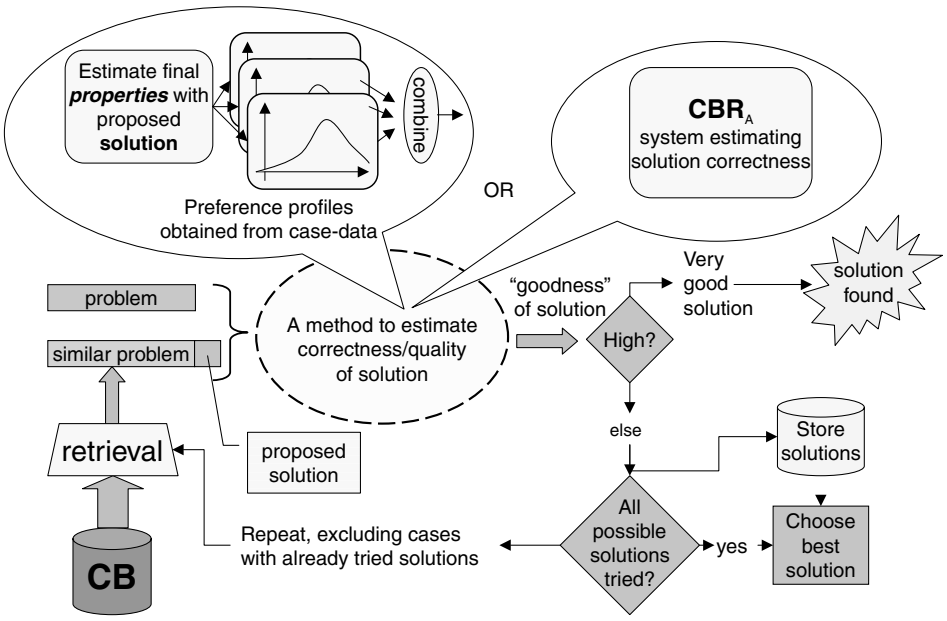


Figure 7: Adaptation of nominal values by iterative "search" for solution.

tablet is manufactured in a robust form, and the desired dose of drug is delivered and absorbed by the patient. Excipients play the role of fillers, binders, lubricants, disintegrants and surfactants in the tablet. Furthermore, a formulation specifies the quantity of each of the added excipients. The difficulty of the formulation task arises from the need to select a set of excipients compatible with the drug, whilst at the same time satisfying a variety of soft and hard constraints. Current tablet formulation practice suggests that this design task is decomposable into subtasks consisting of the choice of excipient component plus determining the excipient amount. It is possible to find a sequence in which the components will be determined, from the most constrained, filler, to the least constrained, surfactant, so that most tablets can be formulated without the need to backtrack in the formulation process.

We conducted adaptation experiments on a case-base containing 156 formulations. Table 1 shows the results for all tablet formulation components, except those where retrieval with *no adaptation* already achieves 100% accuracy. Entries of the form "–" indicate there was no gain from adaptation. The table shows the accuracy as measured by the percentage of acceptable solutions for default CBR retrieval and our automatically optimised retrieval [Jarmulak *et al.*, 2000]. It also shows the gains from using adaptation cases (CBR$_A$). For nominal targets, the gains from preference profile adaptation (PPROF) are shown in brackets. The accuracies and gains are averaged over repeated cross-validation experiments that separate the available data into case-data and test problems.

We can see that for several tablet formulation subtasks learned adaptation did indeed bring improvements. We were especially interested in improving the results for the three tasks for which optimised retrieval with no adaptation had accuracies below 90%. Therefore, we are pleased with the improvements in the tabletSRS and binder prediction. For

CBR$_A$/ (PPROF)	retrieval only (%)	adaptation gain (%)	optimised only (%)	adaptation gain (%)
filler	68.7	— (—)	80.8	— (—)
binder	30.1	5.4 (11.1)	39.2	4.9 (9.9)
disinteg.	93.5	1.9 (2.4)	96.7	0.7 (—)
filler	85.8	9.8	99.9	0.1
binder	95.9	—	99.4	—
lubricant	88.0	0.8	97.4	0.6
tYPfiller	79.7	1.1	96.2	0.3
tSRS	70.3	—	76.4	3.8

Table 1: Percentage Accuracy and Adaptation Gain

the latter task, we notice that the PPROF method gives much better results than CBR$_A$. Preference profiles capture the notion of optimality directly, representing the distribution of feature values for a given prediction; CBR$_A$ relies on more local feedback. Moreover, preference profiles are less susceptible to data over-fitting since their knowledge is based on larger training sets.

Because no gains were achieved for the filler excipient task, we analysed this task more closely by looking at the index trees of the CBR$_A$ system, see Figure. 8. Our expert confirmed that parts of the index reflected correct domain knowledge. All *est. tSRS* nodes correctly capture the preference for tablets with low SRS. However, other nodes were incorrect, mainly due to the lack of sufficient training examples (adaptation cases). For example, the second level nodes under CCA and MAN are obviously incorrect, and the nodes labelled *filler_related* represent the opposite of the domain knowledge. It is encouraging that the knowledge contained in the case-base index was so easy to interpret. It is also easy

to make corrections to the index. This suggests that learning adaptation knowledge should be followed by a review and refinement of that knowledge by a domain expert. This would still require considerably less effort than acquiring all the adaptation knowledge from the expert.

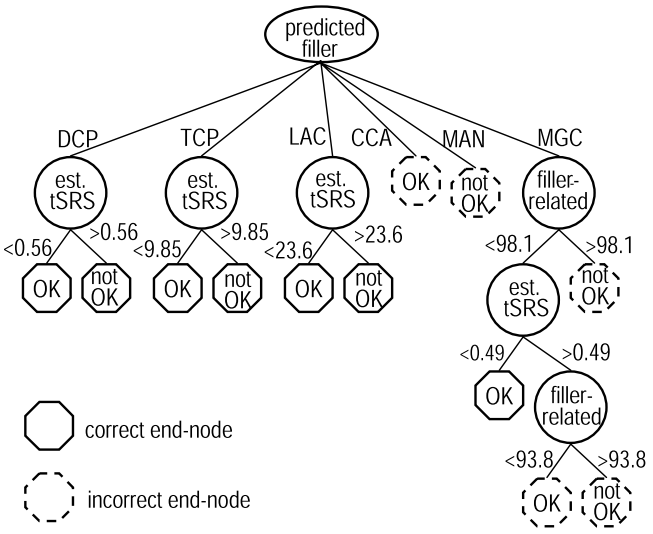


Figure 8: Example case-base index for CBR_A.

5 Conclusions & Future Work

Our research has identified several "knowledge-light" methods of learned adaptation suitable for implementation in general purpose CBR tools. The tool's adaptation phase must allow the preference profile or CBR_A systems to be invoked and repeated retrieval to be undertaken. Since both adaptation methods are based around cases, the CBR tool itself could be the adaptation platform.

The methods described are suitable for decomposable design tasks as illustrated by our tablet formulation problem. For these adaptation subproblems we have developed effective methods for learning adaptation knowledge. These methods are generic and particularly suited to component-based design because they use constraints and optimality of the components. Tasks other than design may also benefit; features describing the suitability of a proposed solution would be used as the estimated feature.

In our experiments, we found that, depending on the task, different methods, like PPROF or CBR_A, may bring better results. This is in part due to the inherent differences in the design tasks: some may be guided mainly by a search for optimal components while others may use constraints to reject unsuitable components. We therefore think that CBR tools should be equipped with a choice of learned adaptation methods, from which the user selects those giving better results.

An important advantage of the methods presented here is that the acquired knowledge is relatively easy to interpret, e.g, the adaptation cases, the CBR_A systems that use them, and the adaptation profiles. Our further research will concentrate on refining acquired knowledge. We intend to study methods of presenting the learned adaptation knowledge to an expert, in order to obtain, and then process, expert feedback with the goal of further improving the CBR adaptation stage.

References

[Bandini and Manzoni, 2000] Stefania Bandini and Sara Manzoni. A support system based on CBR for the design of rubber compounds in motor racing. In *Proc. 5th European Workshop on CBR*, pages 348–357, 2000. Springer.

[Craw *et al.*, 1998] Susan Craw, Nirmalie Wiratunga, and Ray Rowe. Case-based design for tablet formulation. In *Proc. 4th European Workshop on CBR*, pages 358–369, 1998. Springer.

[Hanney and Keane, 1997] Kathleen Hanney and Mark T. Keane. The adaptation knowledge bottleneck: How to ease it by learning from cases. In *Proc. 2nd International Conference on CBR*, pages 359–370, 1997. Springer.

[Jarmulak *et al.*, 2000] Jacek Jarmulak, Susan Craw, and Ray Rowe. Genetic algorithms to optimise CBR retrieval. In *Proc. 5th European Workshop on CBR*, pages 136–147, 2000. Springer.

[Kolodner, 1991] Janet Kolodner. Improving human decision making through case-based decision aiding. *AI Magazine*, 12(2):52–68, 1991.

[Leake *et al.*, 1996] David B. Leake, Andrew Kinley, and David Wilson. Acquiring case adaptation knowledge: A hybrid approach. In *Proc. 13th AAAI Conference*, 1996.

[McSherry, 1998] David McSherry. An adaptation heuristic for case-based estimation. In *Proc. 4th European Workshop on CBR*, pages 184–195, 1998. Springer.

[Netten and Vingerhoeds, 1996] B. D. Netten and R. A. Vingerhoeds. Incremental adaptation for conceptual design in EADOCS. In *ECAI Workshop on Adaptation in CBR*, Budapest, Hungary, 1996.

[Smyth and Cunningham, 1992] Barry Smyth and Padraig Cunningham. Déjà Vu: A hierarchical case-based reasoning system for software design. In *Proc. ECAI92 Conference*, pages 587–589, 1992. Wiley.

[Smyth and Cunningham, 1993] Barry Smyth and Padraig Cunningham. Complexity of adaptation in real-world case-based reasoning systems. In *Proc. 6th Irish Conference on Artificial Intelligence & Cognitive Science*, 1993.

[Smyth and McKenna, 1999] Barry Smyth and Elizabeth McKenna. Building compact competent case-bases. In *Proc. 3rd International Conference on CBR*, pages 329–342. Springer, 1999.

[Stahl and Bergmann, 2000] Armin Stahl and Ralph Bergmann. Applying recursive CBR to the customisation of structured products in an electronic shop. In *Proc. 5th European Workshop on CBR*, pages 297–308, 2000. Springer.

[Wilke *et al.*, 1997] Wolfgang Wilke, Ivo Vollrath, Klaus-Dieter Althoff, and Ralph Bergmann. A framework for learning adaptation knowledge based on knowledge light approaches. In *5th German Workshop on CBR*, 1997.