

Explanation-Driven Case-Based Reasoning

Agnar Aamodt

University of Trondheim, Department of Informatics
N-7055 Dragvoll, Norway
agnar@ifi.unit.no

Abstract. Problem solving in weak theory domains should compensate for the lack of strong theories by combining the various other knowledge types involved. Such methods should be able to effectively combine general domain knowledge with specific case knowledge. A method is described that utilises a presumably extensive and dense model of general domain knowledge as explanatory support for case-based problem solving and learning. A generic reasoning method - captured in what is called the ACTIVATE-EXPLAIN-FOCUS cycle - is able to utilise a rich knowledge model in producing context-dependent explanations. A specialisation of this method for each of the main subprocesses of case-based reasoning is presented, and illustrated with examples.

1 Introduction

A growing part of the AI community is concerned with approaches that integrate several types of knowledge and reasoning methods (see for example [David *et. al.*, 1993]). Although case-based reasoning is a rather new addition to the current set of AI methods, a substantial amount of research address the combination of case-specific knowledge with explicit models of general domain knowledge [Koton, 1989; Bareiss, 1989; Aamodt, 1990; Branting, 1991; Althoff and Wess, 1991; Leake, 1993]. While some CBR methods emphasise the use of specific cases *instead of* general domain knowledge, these methods use cases *combined with* general knowledge. The work described here follows up on this line of research, but puts a stronger emphasis on the *active role* of general knowledge in the various subprocesses of case-based reasoning.

General domain knowledge may be combined with case-based reasoning in several ways. General knowledge may be used for an alternative problem solving method, e.g. a method that is applied if the case-based method fails, and/or it may be used within the case-based method itself. The former will be referred to as *horizontal integration* of general domain knowledge with case-based reasoning, and the latter as *vertical integration*. General knowledge may be of a shallow, associational type (e.g. a set of heuristic rules), or it may capture deeper and more principled knowledge (e.g. a model combining causal, functional and componential knowledge). This paper addresses the use of general domain knowledge *within* a case-based reasoning process, and presents a vertical integration method.

The general type of application we focus on is *knowledge-based decision-support* for *open* and *weak theory* domains. An open domain is a domain which cannot be realistically modelled unless the problem solver's relationships with the external, changing world are anticipated by the model. A weak theory domain is a domain in

which relationships between important concepts are uncertain [Porter, 1989]. Examples of open and weak theory domains are medical diagnosis, law, corporate planning, and most engineering domains. A counter-example would be a mathematical domain, or a well-understood technical domain [Richter and Wess, 1991]. The fact that a domain is open and has a weak theory does not necessarily imply that there is little general knowledge available. It implies that the general knowledge of the domain is theoretically uncertain, incomplete, and subject to changes. Inference and reasoning methods that rely on deductive proofs are not readily applicable, since general assumptions about universal truth of statements can not be made. Statements are more or less plausible, and stronger or weaker supported, rather than true or false. Although it may seem somewhat contradictory, weak theory domains therefore need to compensate for lack of strong knowledge with larger amounts of knowledge, represented in a *coherent* (rather than strictly consistent) knowledge model.

Knowledge of this kind should be interpreted and processed by methods that are able to draw plausible inferences by combining of the various knowledge types available. The main inference type involved is therefore *abduction* [Josephson and Josephson, 1994], and the core part of the reasoning process becomes an abductive *explanation process*¹, operating at various levels of reasoning. At the top level, for example, the corresponding explanation task could be to explain a patient's symptoms in terms of the physiological states that cause them. At a more detailed level it could be to explain why a particular diagnostic hypothesis should be preferred over another.

A strong motivation for a case-based approach is the need for future systems to have *adaptive* capabilities, i.e. to be able to learn from their own problem solving experiences. It is hard to see how they otherwise will be able to cope with the increased domain complexity and the increased user-interactiveness foreseen. To rely solely on general and global criteria and metrics will not work for the types of applications and domains described above. Instead, it is important that the reasoning processes are based on an understanding of the current problem within its *problem solving context*. Context focusing is the means by which problem solving in these domains can become tractable [Compton and Jansen, 1989]. It is in the tight coupling of case-based and generalisation-based approaches we find the strongest potential for realising the type of competent, flexible and interactive behaviour we would like future AI systems to exhibit. This is the high-level hypothesis we are investigating.

The method we will describe is based on the CREEK² architecture [Aamodt-91], and represents both an extension and a specialisation of that system. The next chapter introduces the notion of explanation-driven case-based reasoning, and presents a basic framework. This is followed by an outline of how case knowledge and general knowledge may be integrated in a unified representation system. In the subsequent chapter, a more detailed description of the explanation-driven case-based

¹Abduction is viewed as an "inference to the best explanation" (see, e.g. [Thagard, 1988]), and covers both the generation and evaluation of hypotheses.

²Case-based Reasoning through Extensive and Explicit general Knowledge

reasoning method is presented, and specified for each of the CBR tasks of case retrieval, case reuse, and learning. A simple example illustrates the method. Finally, the explanation-driven CBR approach is discussed by comparing it to related methods.

2 Explanations as support for case-based processes

The notion of an explanation has two different meanings related to knowledge-based systems. One is as the explanation that a system may produce for the benefit of the user, e.g. to explain its reasoning steps or to justify why a particular conclusion was reached. The other interpretation, and the one adopted here, is as the internal explanation that a system may construct for itself during problem solving. In this interpretation the construction of explanations becomes an integral part of the system's own reasoning process, as outlined in the previous section. Being able to produce explanations for internal use, however, is also a potential for good user explanations.

Note that the term "problem solving" is used in a general sense in this paper. For example, to solve a problem may be to find the fault of a car (solving a diagnosis problem), as well as to assess a legal situation in court (solving an interpretation problem). A problem is defined by a goal (what to achieve) which in turn sets up one or more tasks (what to accomplish). To distinguish between external, application related tasks of a problem solver, and tasks set up by the system's own reasoning process, the first will be referred to as *application tasks*, while the latter are named *reasoning tasks*. Hence, learning tasks are reasoning tasks. A task is in general accomplished by applying a method to it¹.

The main reasoning tasks of a case-based reasoner are:

- The identification of relevant features from a problem description
- The retrieval of a past case by assessment of similarity to the new problem
- The modification of a previous solution to fit within a new context
- The evaluation of a suggested solution
- The identification of what to retain from a case just solved
- The learning of indexes for memorising new cases

As indicated by the grouping of these tasks, they form three larger tasks, which will be referred to as *retrieve*, *reuse*, and *learn*.²

In the following, a method that utilises general knowledge extensively as an integrated part of case-based reasoning is described. The strong role that explanation processes have within this method has lead us to refer to the approach as *explanation-driven case-based reasoning*. A generic mechanism, called the "explanation engine", constitutes the fundamental reasoning scheme. It splits a reasoning task into the three subtasks ACTIVATE, EXPLAIN, and FOCUS - as illustrated in figure 1.

¹Our ontology of tasks, etc. is based on the Components of Expertise framework [Steels, 1990].

²The evaluation task is often regarded as a fourth task (see for example [Aamodt and Plaza, 1994], but being a task external to the system's reasoning, it is here de-emphasised.

The explanation engine takes a state description (the initial state) as input and returns an end state description (the goal state) as output. This constitutes a generic reasoning process. The goal is part of the task specification, and specifies what type of concept the outcome of the task should be. Constraints are requirements on the goal state, e.g. constraints on the solution as such (e.g. form, generality, or class priorities), or external constraints on the reasoning process (e.g. time or resource constraints). Input data are the source data for the process. For application tasks, input data are typically observations and other findings from the task environment, while the input data for internal reasoning tasks are descriptors characterising the current state of problem solving. The action of asking for a solution triggers the reasoning process.

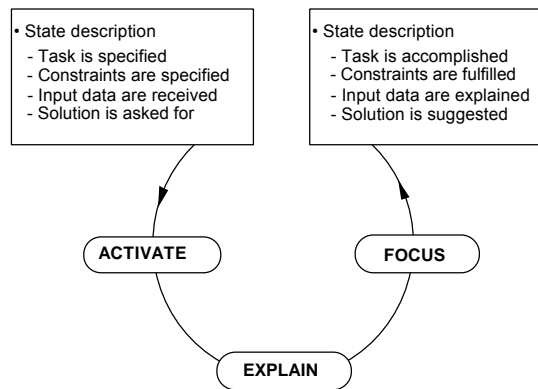


Figure 1. States and tasks of the explanation engine

The explanation engine gets a significant part of its inferencing power from the underlying representation system, a densely coupled semantic network where nodes as well as relations are represented as frame concepts. All knowledge, general as well as case specific, is represented as frame structures. At the low level, the frame interpreter makes use of basic inference methods such as property inheritance, relation-dependent spreading activation, low level frame matching, and constraint propagation. In addition to the system's own model of general domain knowledge, the explanation engine also assumes that there is a competent user at the terminal. Hence it will interact with the user in order to have conclusions confirmed, explanation conflicts solved, etc., whenever its own knowledge is insufficient to produce strong enough explanations.

The methods of **ACTIVATE**, **EXPLAIN**, and **FOCUS** operate briefly as follows:

ACTIVATE takes as input a problem description in terms of a task and goal specification, a set of solution constraints, and a set of input data (findings). It returns a set of activated concepts, assumed relevant for accomplishing the task. The main purpose of **ACTIVATE** is to establish a broad but limited context in which to conduct further reasoning. Two methods are used: *spreading activation* along a set of relations in the semantic network, and *direct reminding* by following links to past

cases. By spreading recursively along a suitable set of relations, starting with the goal concept, a sphere of concepts relevant to the goal is produced. A similar process starts out from the input findings. All concepts within the findings sphere that are also within the goal sphere, and all concepts on paths from this set to initial goals and findings, are marked as activated. The set of activated concepts constitute the part of the knowledge base to be used for further reasoning. Some spreading relations are pre-defined, while some are application-dependent, determined during the knowledge modelling process. An initial set of possible solutions is activated by retrieval of past cases from the given input findings or findings inferred by the spreading activation process.

EXPLAIN takes an activated knowledge structure as input and returns a set of solution hypotheses supported by explanations. It builds up support and justification for goals, subgoals, and intermediate states, that were identified by **ACTIVATE** or generated by **EXPLAIN** itself. Such concepts may be of various kinds, e.g. an hypothesised solution or solution class, an inferred state, an inference action to take, or a new case to be learned. The method of generating and evaluating explanations searches through activated parts of the semantic network, and follows the paths of cumulatively highest explanatory strength. In assessing this strength, it makes use of *default explanatory strengths* attached to each semantic relation and each meaningful combination of two successive relations¹. The strengths may have *contextual constraints* attached to them. Further, **EXPLAIN** has at its disposal a set of *explanation evaluation strategies* in the form of decision rules. An algorithm computes the resulting explanatory strength as the search proceeds. For application-oriented tasks, **EXPLAIN** will typically interact with the user, in order to get confirmations or rejections of hypotheses for which the system is unable to produce a sufficiently strong explanation for by itself.

FOCUS takes as input a set of hypothesised solutions, supported by sufficiently strong explanations, and returns a single - the presumably best - solution suggestion. It uses information about given *priorities* (e.g. try the least risky solution first), and possible *constraints of content or form* of the solution concept. These constraints are typically external, pragmatic constraints defined by the application environment and the current problem solving situation.

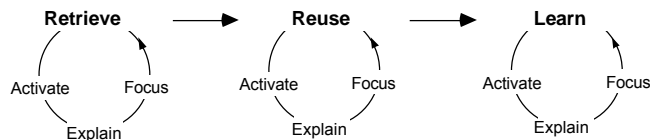


Figure 2. Generality of the Explanation Engine

¹Default strengths assigned to single relation is in principle similar to the pre-defined relational strengths in the PROTOS system [Bareiss, 1989]. The method for combined relational strengths is an extension that enables a more specific assessment of explanatory strengths of relations. This method, together with several spreading activation strategies, was implemented and tested in the KNOWIT system, a prototype system for knowledge-based information retrieval [Sølvberg *et. al.*, 1992].

Figure 2 illustrates the generic properties of the ACTIVATE-EXPLAIN-FOCUS cycle. It constitutes the basic reasoning scheme for all three processes of retrieval, reuse, and learning. In chapter 4 the generic methods described here will be instantiated and exemplified for each of the three CBR processes.

3 Knowledge Representation

The main representational concern of CBR research has been the representation of case knowledge, i.e. the contents and form of a case, the memory structures and indexes. For an explanation-driven approach, however, the representation of general knowledge is equally important.

The CREEK architecture combines case-based and generalisation-based knowledge within an integrated system design. All types of knowledge are represented within the same representation system, a frame-based language called CreekL (implemented in Common Lisp). Concepts are represented as 4-level structures of slots, facets, value expressions, and value fields. CreekL incorporates some features of CYCL [Lenat, 1990], such as the explicit representation of relations as concepts and inverses for all relations. It puts a similar emphasis on capturing the *contents* of knowledge as the CYC representation does. Each relation (corresponding to a slot name) and each value symbol is defined in its own frame. This results in a densely coupled knowledge model that integrates concept definitions, rules and cases, and object-level as well as control level knowledge. CreekL has an explicit model of its representational structures, described within the frame system itself (i.e. there is one frame for the concept "frame", one for the concept "slot", frames representing parts of frames and slots, etc.). This enables a reflective system that can reason about its own methods.

Below, a frame representing the concept car with its slots, facets and values is exemplified. It is shown as a pretty-printed Lisp structure with most parentheses removed. Facets represent value types (e.g. defaults, value-dimensions or ranges), constraints on values (e.g. legal value classes), and demons - attached Lisp functions that return a value or perform an operation. The latter is illustrated by the *if-needed* facet: When we want to know the age of a particular car, say *car#1* (an instance of *car*), the *has-age* slot is inherited, and the *time-difference* function is called with the two arguments shown. The dotted notation 'a.b' refers to the value of slot b in frame a, i.e. the value of the *has-production-year* slot of *car#1* (after substitution of self).

```

car
  has-instance      value      car#1
  subclass-of      value      vehicle means-of-transportation sporting-gear
  has-part         value      wheel fuel-system engine electrical-system
  has-number-of-wheels default 4
  has-colour       value-class colour
  has-fault        value-class car-fault
  has-age          value-dimension years
                  if-needed   (time-difference *current-year*
                           self.has-production-year)

```

The frames, interconnected through their slots, form a semantic network of concept nodes and bi-relational links. Figure 3 illustrates this perspective to the CreekL knowledge structure, and shows - for a small excerpt of a knowledge base -

the integration between general and case-specific knowledge. The example domain is car starting problems, and the figure highlights the interconnected, unified structure of a CreekL knowledge model. The direction of the relations are top-down. All relations have inverses (not shown), for example the inverse of has-subclass is subclass-of.

The grey areas at the upper left and lower right highlight regions of goal-related and findings-related concepts, and it is seen how the two concept types are connected via the partial model of general and specific domain knowledge. The relationships shown are basically taxonomic (has-subclass/subclass-of and has-instance/instance-of), componential (has-part/part-of), or related to the particular domain example of fault finding and testing.

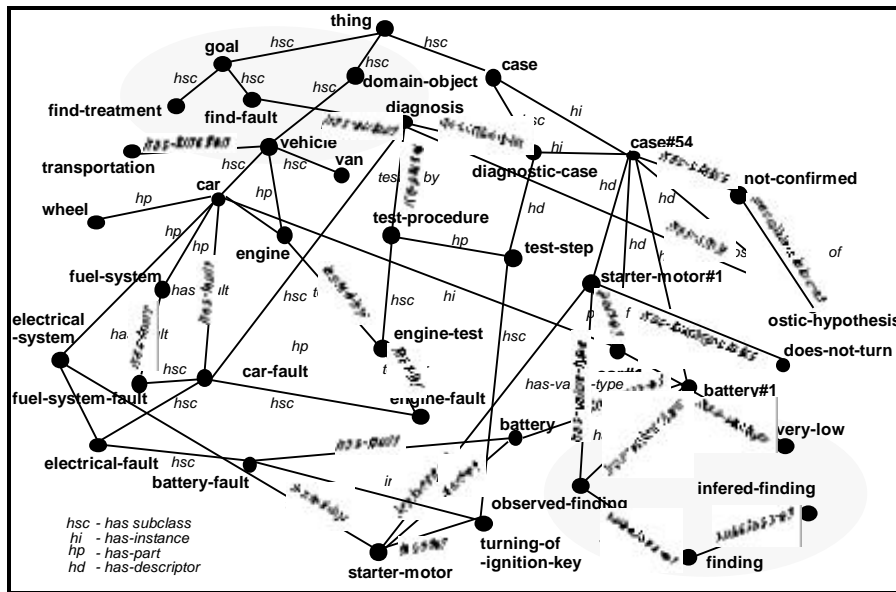


Figure 3: A tangled network of CreekL concepts

An explanation is a structure consisting of a single relationship - expressed as a (<concept><relation><value>) triplet - or a connected graph of such relationships. Explanations are stored within the frame of the concept to which the explanation belongs. The strength of an explanation is included as a numeric strength value. For example, an explanation for why a particular starter motor will not turn because of a low battery voltage, is represented as follows:

```

starter-motor#1
instance-of      value      starter-motor
part-of         value      car#1
has-turning-status expected-value turns
value          value      (does-not-turn
                        (0.9 (battery-1 has-voltage very-low)
                        (battery-1 instance-of battery)
                        ((battery has-voltage very-low) causes

```

(starter-motor has-turning-status does-not-turn)
(starter-motor-1 instance-of starter-motor)))

Explanations are nested structures of relationships. The above concept is a temporary concept, instantiated, maintained and updated during the problem solving process, but not retained afterwards. Explanations that support the solution of the application problems are retained as part of stored cases. A stored case is a rich source of information, containing the following slot types (example from diagnostic domain):

problem goal	successful diagnosis	explanation of successful diagnosis
relevant findings	successful repair	explanation of successful repair
differential findings	failed diagnosis	explanation of failed diagnosis
differential cases	failed repair	explanation of failed repair

To produce explanations, the system may reuse a previous explanation from a retrieved case, or it may use the general domain knowledge. In an early state of system operation, where the general domain knowledge is still subject to frequent changes, explanations will typically be produced on the basis of the general model, containing the most updated knowledge. Explanations retrieved from cases are more effective, and enable a more precise focus, but will typically be used after the general domain knowledge has reach a relatively stable state. Three slot types of a stored case serve the role as index links for initial case retrieval: relevant findings (favours retrieval of the case), differential findings (favours retrieval of a case similar to, but also significantly different from, another past case), and solutions (successful and failed).

4 Explanation-driven case retrieval, reuse and learning

As previously stated, a case-based reasoning process may at the top level be represented by the three tasks Retrieval, Reuse, and Learning. In section two, the basic method of explanation-driven CBR was presented by outlining the ACTIVATE-EXPLAIN-FOCUS cycle. In the following, a more detailed description is given, by instantiating this cycle for each of the three CBR tasks. The main reasoning tasks and their methods are elaborated, and illustrated with an example from the domain of car starting problems. We will leave many details aside¹, and emphasise on the principles of explanatory support to the CBR processes.

Let us assume that the following input case description is given. (All facets are 'value' facets, and facet names are dropped for the sake of simplicity):

case#0	
instance-of	initial-state-case car-starting-case
of-car	N-VD-123456
has-task	find-car-starting-fault
has-electrical-status	electrical-system-ok
has-engine-status	engine-turns engine-does-not fire
has-ignition-status	spark-plugs-ok distributor-ok
has-weather-condition	low-temperature moisty normal-air-pressure
has-location-of-problem	inside-garage

¹See [Aamodt, 1991] for a detailed description of the underlying Creek system's methods for case storage and indexing, computation of combined remindings strengths, assessment of relevance factors, etc.

Retrieval

The goal of Retrieve is to return the best matching case from the case base.

ACTIVATE has here two subtasks. One is to *determine a relevant broad context* for the problem. We assume a large, multifunctional knowledge base, in this case of knowledge about cars and other vehicles, and in order to produce focused explanations later on, we want to activate just those parts and perspectives of this knowledge that is potentially relevant for the current task. The broad context is determined by the method of goal-focused spreading activation as previously described. The spreading activation method activates all input findings and goal relevant concepts. In our example the spreading starts out from the top-level application task, i.e. find-car-starting-fault, and the input features (has-electrical-status, etc.). Spreading-relations include general taxonomic ones (has-subclass, subclass-of, has-instance, instance-of), causal relations (causes, caused-by), associational relations (occurs-together-with, leads-to), and application-specific relations (tested-by, test-for). The other subtask of ACTIVATE uses the findings as indexes to the case base in order to *retrieve a set of cases* whose matching strength is above a certain threshold. A concept classified as a finding has a slot called relevant-finding-in which holds a list of cases and a computed relevance factor for the finding with respect to each case. For example:

```
spark-plugs-ok
  subclass-of      ignition-system-finding
  relevant-finding-in (case#19 0.7) (case#85 0.6) (case#366 0.95) (case#123 0.6)
  ....
```

Combined reminders from the findings to the cases they point to are computed. In our example this results in the following two cases being retrieved (partial case descriptions shown), i.e. their degree of match is above a certain threshold level:

```
case#85
  instance-of      solved-case car-starting-case
  has-task         find-car-starting-fault
  of-car          N-DD-234567
  has-solution     broken-carburettor-membrane
  has-electrical-status electrical-system-ok
  has-engine-status engine-turns engine-does-not fire
  has-ignition-status spark-plugs-ok
  has-weather-condition low-temperature sunny normal-air-pressure
  has-recent-driving-history hard-driving
  ....

case#123
  instance-of      solved-case car-starting-case
  has-task         find-car-starting-fault
  of-car          N-CC-345678
  has-solution     carburettor-valve-stuck
  has-electrical-status electrical-system-ok
  has-engine-status engine-turns engine-does-not fire
  has-ignition-status spark-plugs-ok distributor-ok
  has-weather-condition high-temperature moisty low-air-pressure
  has-location-of-problem inside-garage
  ....
```

EXPLAIN then *evaluates the match* between the problem case and the cases returned from ACTIVATE. This is necessary since the initial case retrieval was based

on a rather superficial matching process. The relevance to the problem has to be justified for findings that matches well, and mismatched findings has to be explained as not important. The input case differs from case#85 in the following findings:

	<i>Input case</i>	<i>Case#85</i>
has-ignition-status	spark-plugs-ok distributor-ok	spark-plugs-ok
has-weather-condition	low-temperature moisty	low-temperature sunny
	normal-air-pressure	normal-air-pressure
has-recent-driving-history	---	hard-driving
has-location-of-problem	inside-garage	---

First, EXPLAIN tries to 'explain away' findings that occur in one of the cases and not in the other. Next, it attempts to construct an explanation that shows the similarities of findings with syntactically different values in the two cases. For example, a finding of case#85 that is missing in the input case is hard-driving. In the knowledge model, hard driving and broken carburettor membrane are connected:

hard-driving *always-leads-to* extreme-engine-load *may-lead-to* abnormally-high-carburettor-pressure
causes broken-carburettor-membrane

Based on this explanation, has-recent-driving-history is regarded an important finding in case#85. The system therefore asks the user about the value of this finding in the input case. The user replies that the recent driving has been normal. The degree of match of case#85 is then reduced. In the knowledge model, there are no explanatory strong connections from weather conditions, the status of the distributor, or the location of the car, to the failure broken-carburettor-membrane. Hence, the difference in the two other findings does not change the degree of match. A similar procedure is then applied to case#123, resulting in no significant change in its degree of match.

FOCUS makes the *final selection of the best case*, or rejects all of them. The case with the strongest explanatory justification of its findings will normally be selected. In our example the reduced matching degree of case#85 leads to case#123 being returned as the outcome of the Retrieval task.

Reuse

The goal of Reuse is to use the solution of a previous case in solving a new problem, usually involving some kind of modification of the past solution.

ACTIVATE starts out from the solution of the best matching case, and spreads activation to concepts representing all *expected findings* given the solution of the retrieved case. The spreading relations used for this typically include causal and functional relations, as well as direct associations (e.g. *implies* and *co-occurs-with* relations). In addition, possibly *risky consequences* of applying the solution are also activated. Values of expected findings that are not known, is requested when needed, i.e. during the EXPLAIN step.

EXPLAIN has two major subtasks. One is to *evaluate the solution* proposed by the retrieved case. Expected findings have to be confirmed, or explained as irrelevant for the present problem. An attempt is made to infer expected findings before asking the user. If all relevant expectations are covered for, control is given to the FOCUS task. If not, the second EXPLAIN task, *modification of the solution*, is triggered. An attempt is made to produce an explanation structure that justifies a replacement or tweaking of the solution. For example: The solution in the retrieved case was stuck

carburettor valve. This is suggested to the user, but after inspection the carburettor turns out to be OK. The main explanation path from this solution to the findings is

carburettor-valve-stuck causes too-rich-gas-mixture-in-cylinder causes no-chamber-ignition causes engine-does-not-fire

See figure 4.

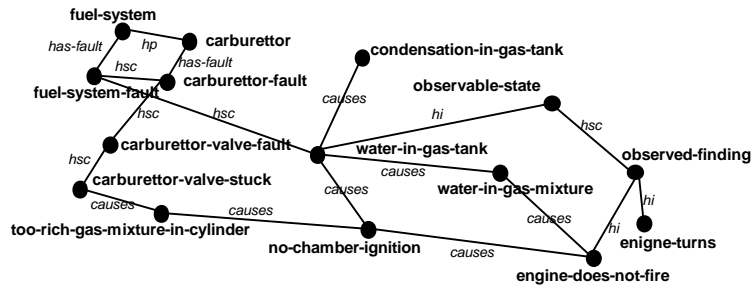


Figure 4. Partial knowledge structure in the car domain

Before looking for a better matching case, the system tries to modify its explanation of the findings. By searching backwards along the explanation path (lower part of figure 4) for other explanations of its states, it finds that engine-does-not-fire may also be caused by water-in-gas-mixture, in turn caused by water-in-gas-tank.

The fault water-in-gas-tank is also supported by the findings moisty and low-temperature, via their relations to condensation-in-gas-tank (not shown in the figure), and is therefore the solution suggested by EXPLAIN.

FOCUS is usually a small task, unless the explanation process returns several competing solutions. It checks whether a proposed solution confirms with *external requirements*, and proposes the suggested solution to the user. The water-in-gas-tank suggestion is checked and confirmed by the user. FOCUS also extracts a minimal, *sufficient explanation* for the case, based on the explanation structures produced by the EXPLAIN task. The result of this is given as input to the LEARNING task.

Learning

The goal of Learning is to capture the experience from the problem just solved, by constructing a new case and/or modifying parts of the knowledge base.

ACTIVATE works on the structure activated by Retrieve and Reuse, and extracts *potential concepts and structures* for learning, i.e. possible contents of a new case, and explanation structures that support the solution. New or modified concepts that may have been introduced by the user is also marked for the learning process.

EXPLAIN has three subtasks. The first is to justify whether a *new case* needs to be constructed or not. This is done if no similar past case was found, if a retrieved solution needed significant modification, or if the set of relevant problem findings are sufficiently different in the two cases. Explanations has to be produced in order to assess the significance of a modification or of feature differences. In our example a new case is added to the case base.

The second subtask is the justification step that determines what *structures to*

retain from a problem solving experience. Among these are the relevant findings with respect to the final solution. For each finding currently in the case, its relevance is explained. The strongest explanation from the solution to the findings, i.e. a structure of explanatory relationships, is retained in the learned case. If this does not constitute a sufficiently strong explanation for the solution (threshold value), additional 'parallel' explanation paths are added. Learning of generalisations does not play a strong role in our method, but a lazy generalisation of values for findings is done if justified by an explanation or by the user. Given that a new case is to be stored, the third subtask is to compute the importance (degree of necessity) and predictive strength (degree of sufficiency) of case findings, in order to determine their *relevance factors*.

FOCUS is the task of putting together the new structures into a case frame, and *actually storing* the new or modified case and its indexes. This also includes updating the relevant-finding-in slot of relevant findings. A finding may be relevant even if it is not an explicit part of an explanation path. The status of the electrical and ignition systems, for example, support the fact that a solution is found within the fuel system.

In our example, the explanation structure stored contains four paths that together explain the relevant findings. The following new case is stored (value facets shown):

```

case#461
instance-of          solved-case car-starting-case
of-car              N-VD-123456
has-process-status  solution-proposed
has-input-time      22/07/93 12:10
has-relevant-finding (has-electrical-status electrical-system-ok
                    has-engine-status (engine-turns engine-does-not fire)
                    has-ignition-status (spark-plugs-ok distributor-ok)
                    has-weather-condition (low-temperature moisty normal-air-pressure)

has-location-of-problem inside-garage
has-driving-history    normal-driving
has-solution           (water-in-gas-tank
                       (0.92
                        ((engine-does-not-fire caused-by water-in-gas-mixture)
                         (water-in-gas-mixture caused-by water-in-gas-tank))
                        (carburettor-fault has-status false)
                        (((low-temperature combined-with moist) leads-to
                          condensation-in-gas-tank)
                         (condensation-in-gas-tank causes water-in-gas-tank))
                        (water-in-gas-tank does-not-violate
                         (electrical-system-ok spark-plugs-ok distributor-ok
                          engine-turns normal-air-pressure)))

same-solution-in    case#06 case#88 case#388 case#401
difference-case     case#128

```

Following the updating of the case structure, a test is run by entering the initial problem once again. It is checked whether the case just learned is retrieved, or - if no new case is learned - whether the problem solving procedure is otherwise improved.

5 Discussion

Case-based reasoning has shown to be a powerful problem solving and learning paradigm for a variety of application domains. Characteristics of the domain type being addressed is important in determining the type of CBR method to use. In

closed and well-defined domains, the need for supportive general knowledge is much less than in open domains, and may often be compiled beforehand into global metrics of similarity and other general operators [Richter and Wess, 1991]. When explicit general knowledge is integrated into the CBR processes, two synergy effects are immediately seen: One is to provide explanation-based control and guidance to the case processes, by, e.g., focusing on particular goals and tasks, constraining search, and supporting proposed hypotheses. As previously stated, this is the synergy effect aimed at here. The other the kind of 'inverse' effect achieved by using the cases within the explanation process itself, i.e. a case-driven explanation process (as in the SWALE system [Schank and Leake, 1989]). Methods developed from the latter motivation may also be useful to obtain the former effect, but this is not presently part of our method.

Several methods have been developed that make use of explicit models of general knowledge in its case-based processes [e.g. Hammond, 1986; Kolodner, 1987; Koton, 1989; Porter *et. al.*, 1990; Branting, 1991]. However, although representing very important contributions to this research, methods that have been proposed typically focus on one or a small subset of the CBR tasks. An exception is CASEY [Koton, 1989], but that approach relies on a strong knowledge model, and leaves out the interactive co-operation with the user which is needed in open and weak theory domains. In a sense, our approach shares the widespread use of general knowledge with CASEY and the interactive role of the user with PROTOS [Bareiss, 1989]. Recent suggestions for integrated architectures [Althoff and Wess, 1991; Ram and Cox, 1992, Plaza and Arcos, 1993; Althoff *et. al.*, 1993] represent interesting work towards more unified methods, but so far the problem has been addressed only partly, at a high and abstracted level, or for closed domains. The MOLTKE system [Althoff and Wess, 1991] is an example of the latter.

The knowledge-intensive approach we have taken has forced us to pay a lot of attention to the knowledge representation problem. Related methods have also had to address this issue to some extent. CASEY [Koton, 1989] uses a pre-existing causal model of general knowledge, represented as a causal network augmented with probability estimates for mere associations between features and diagnostic states. Cases are held in a separate structure, organised as a Schank/Kolodner type of dynamic memory. PROTOS [Bareiss, 1989] has an integrated structure consisting of a semantic network of domain categories linked by a variety of relations (causal, functional, associational, etc.), in which cases are linked as exemplars of diagnostic categories. The CREEK approach is most similar to PROTOS in this respect, but there are significant differences in the way cases are integrated with general knowledge, as well as how they are indexed and used.

Modelling and representing knowledge for explanation-driven CBR is viewed as a general knowledge engineering task [Steels, 1990], subject to the problems, methods and tools addressed by the knowledge acquisition community. Unlike some other motivations, for example PROTOS', we do not advocate CBR as an alternative answer to the *initial knowledge modelling* problem, but as an approach to the problem of *continuous knowledge maintenance*. The view of knowledge modelling as basically a top-down modelling process [Chandrasekaran, 1992; Wielinga *et. al.*,

1992] is merged with the bottom-up oriented view provided by learning from experience [Van de Velde and Aamodt, 1992]. Hence, the often dominant role of top down modelling is weakened in favour of a more iterative development process for knowledge-based systems.

6 Conclusion and Further Work

The integrated method described here is under implementation in our department. An experimental evaluation will therefore have to wait. It's plausibility, however, is supported by the integration of two approaches which each has a lot of merit, and the fact that existing approaches to knowledge-intensive CBR in weak theory, open domains has shown promising results (e.g. [Bareiss, 1989; Hinrichs, 1989]).

All CBR systems use indexes in one way or another for case retrieval, but there are a lot of unresolved problems here. Research issues include what type of indexing terms to use, the actual index vocabulary, the way indexes are linked to cases, and the combination of indexes during retrieval. A problem with too heavy reliance on indexing, however, is that indexes are a kind of pre-compiled knowledge. A characteristic of case-based learning is that the generalisation process is not made when learned knowledge is stored, but when this knowledge is used, i.e. during problem solving. Indexing works in the opposite direction to this, since it anticipates and pre-sets the future use of case knowledge. The alternative approach to case retrieval is search, which is time consuming, and difficult to guide in the wanted direction. Elaborate reasoning within an extensive and rather deep model of general knowledge, is a cost demanding process. The problem, then, is to find a suitable balance between the two. An explanation-driven approach enables a search procedure which is constrained by general domain knowledge related to the context of the actual problem. We have started to study how this may allow a system to weaken its reliance on abstract indexes, in favour of making abstractions within the context of the actual problem.

Currently, the ACTIVATE-EXPLAIN-FOCUS engine uses only its general knowledge to produce explanations. A case in the CreekL representation is a rich structure, and the utilisation of the cases themselves in the explanation process (e.g. [Leake, 1993]) is an extension we are looking into.

Acknowledgements

Arnt Lockert has improved the CreekL representation system and is implementing a case adaptation (solution modification) method as part of his M.Sc. study. Pinar Ozturk is studying explanation- and case-driven context focusing in her Ph.D research. They both provided valuable comments to earlier drafts of the paper.

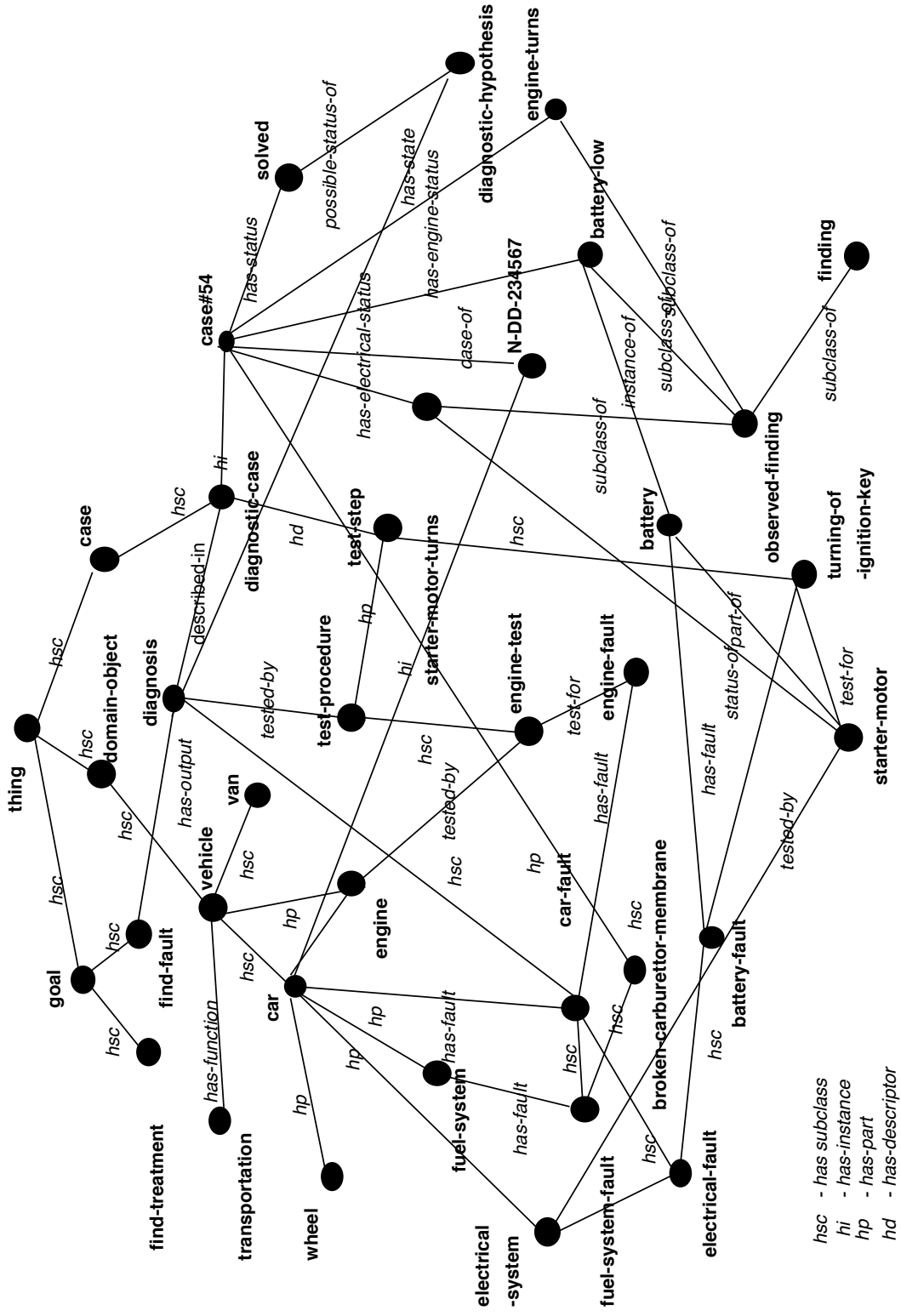
References

- [Aamodt, 1990] Agnar Aamodt. Knowledge intensive case-based reasoning and sustained learning. In: *ECAI-90*; Proceedings of the ninth European Conference on Artificial Intelligence, Stockholm, August, 1990.
- [Aamodt, 1991] Agnar Aamodt. A knowledge-intensive approach to problem solving and sustained learning. Ph.D. Dissertation, University of Trondheim, Norwegian Institute of

- Technology, May 1991. (University Microfilms PUB 92-08460)
- [Aamodt and Plaza, 1994] Agnar Aamodt, Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, Vol. 7, No.1 March 1994, pp. 39-59.
- [Althoff and Wess, 1991] Klaus-Dieter Althoff, Stefan Wess. Case-based knowledge acquisition, learning and problem solving for real world tasks. *Proceedings EKAW-91*, European Knowledge Acquisition Workshop, 1991.
- [Althoff et. al., 1993] K.-D. Althoff, R. Bergmann, F. Maurer, S. Wess, M. Manago, E. Auriol, N. Conruyt, R. Traphöner, M. Braeuer, S. Ditttrich, (1993). Integrating Inductive and Case-Based Technologies for Classification and Diagnostic Reasoning. In: E. Plaza (ed.), *Proc. ECML-93 Workshop on "Integrated Learning Architectures"*, Vienna, Apr 1993.
- [Bareiss, 1989] Ray Bareiss. *Exemplar-based knowledge acquisition*. Academic Press, 1989.
- [Branting, 1991] Karl Branting. Exploiting the complementarity of rules and precedents with reciprocity and fairness. In: *Proceedings from the Case-Based Reasoning Workshop 1991*, Washington DC, May 1991. DARPA. Morgan Kaufmann, 1991. pp 39-50.
- [Chandrasekaran, 1992] B. Chandrasekaran. Task-structure analysis for knowledge modeling. *Communications of the ACM*, Vol. 35, no. 9, September 1992 (Special issue on modeling), pp. 124-137.
- [Compton and Jansen, 1989] P. Compton, R. Jansen. A philosophical basis for knowledge acquisition. *EKAW 89, Third European Workshop on Knowledge Acquisition for Knowledge-Based systems*. Ed. by J. Boose, B. Gaines, J.G Ganascia. ECCAI/AFCEC/ARC, Paris 1989. pp.75-89.
- [David et. al., 1993] J-M. David, J-P. Krivine, R. Simmons (eds.), *Second generation expert systems* (Springer, 1993).
- [Hammond, 1986] Kristian J. Hammond. CHEF; a model of case-based planning. *Proceedings of AAAI-86*. Morgan Kaufmann, 1986. pp 267-271.
- [Hinrichs, 1992] Thomas R. Hinrichs. *Problem solving in open worlds: A case study in design*. Lawrence Erlbaum Associates, 1992.
- [Josephson and Josephson, 1994] J. Josephson, S. Josephson. *Abductive inference, computation, philosophy, technology*. Cambridge University Press, 1994.
- [Kolodner, 1987] Janet Kolodner. Extending problem solver capabilities through case-based inference. *Proc. 4th workshop on Machine Learning*, UC-Irvine, June 22-25 1987. pp 167-178.
- [Koton, 1989] Phyllis Koton. Using experience in learning and problem solving. Massachusetts Institute of Technology, Laboratory of Computer Science (Ph.D. dissertation, October 1988). MIT/LCS/TR-441. 1989.
- [Lenat, 1990] Doug Lenat, R. Guha. Building Large Knowledge-Based Systems; Representation and Inference in the CYC Project. Addison-Wesley, 1989.
- [Leake, 1993] D.B. Leake. Focusing construction and selection of abductive hypotheses. In *IJCAI-93, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, 1993. Vol.1. pp 24-29.
- [Plaza and Arcos, 1993] Enric Plaza, Josep-Lluís Arcos. Reflection, memory, and learning. Institut d'Investigació en Intel·ligència Artificial, CSIC/CEAB, Report de Recerca IIIA 93/2. Also in *Proceeding from MSL-93 Workshop on Multistrategy Learning*.
- [Porter et al., 1990] Bruce Porter, Ray Bareiss, Robert Holte. Concept learning and heuristic classification in weak theory domains. *Artificial Intelligence*, vol. 45, no. 1-2, September 1990. pp 229-263.
- [Porter, 1989] Bruce Porter. Similarity assessment; Computation vs. representation. In *Proceedings from the Case-Based Reasoning Workshop*, Pensacola Beach, Florida, May-June 1989. Sponsored by DARPA. Morgan Kaufmann, 1989. pp 82-84.
- [Ram and Cox, 1992] Ashwin Ram, Michael Cox. Multistrategy learning with introspective meta-explanations. In D. Sleeman, P. Edwards (eds.), *Machine Learning, Proceedings of the*

- International Workshop (ML 92)*, Aberdeen, 1992. pp 123.128.
- [Richter and Wess, 1991] A.M. Richter, S. Weiss. Similarity, uncertainty and case-based reasoning in PATDEX. In R.S. Boyer (ed.): *Automated reasoning, essays in honour of Woody Bledsoe*. Kluwer, 1991, pp. 249-265.
- [Schank and Leake, 1989] Roger Schank, David Leake. Creativity and learning in a case-based explainer. *Artificial Intelligence*, Vol. 40, no 1-3, 1989. pp 353-385.
- [Steels, 1990] Luc Steels. Components of expertise. *AI Magazine*, 11(2), 1990. pp. 29-49.
- [Sølvberg *et al.*, 1992] Ingeborg Sølvberg, Inge Nordbø, Agnar Aamodt. Knowledge-based information retrieval. *Future Generation Computer Systems*, Vol.7, 1991/92, pp 379-390.
- [Thagard, 1988] Paul Thagard. Explanatory Coherence. *Behavioural and Brain Sciences* 12 (1989) 435-467.
- [Van de Velde and Aamodt, 1992] Walter Van de Velde, Agnar Aamodt: Machine learning issues in CommonKADS. KADS-II Report, KADS-II/TII.4.3/TR/VUB/002/3.0, Free University of Brussels -VUB, 1992.
- [Wielinga *et al.*, 1992] Bob Wielinga, Walter van de Velde, Guus Schreiber, Hans Akkermans. Towards a unification of knowledge modelling approaches. In *Proceedings of JKAW-92, Japanese Knowledge Acquisition Workshop*. 1992

Fig 3



hsc - has subclass
 hi - has instance
 hp - has part
 hd - has-descriptor

