Integrating Feature Extraction and Memory Search

Christopher Owens
Department of Computer Science
Yale University

Reasoning from prior experience depends upon having a large memory of prior cases and a system for retrieving them when they are relevant. Often, relevance means similarity to the current situation on the basis of abstract or thematic features other than the features used to initially describe the current situation. To retrieve cases relevant to some new situation, a system must be able to describe the new situation in abstract terms and use that description as a search key or as a means to judge the appropriateness of prior cases. Typically, the abstract description process has been considered as separate from the memory search process. This paper presents a scheme for integrating the feature extraction and memory search processes and argues in favor of such an approach on methodological and efficiency grounds. It presents a program that exploits parallelism to control some of the high processing costs associated with feature extraction and memory search.

RETRIEVAL AND ABSTRACT FEATURES

Recent work has suggested that a good approach to planning and problem–solving situations is for a system to get reminded of specific prior experiences and to reason based upon the similarities and differences between that prior experience and the current problem. For example, case-based reasoners such as those described by [Simpson, 85], [Hammond, 86], [Sycara, 87], [Kolodner, 87] and [Ashley and Rissland, 87] and analogical reasoners such as those described by [Carbonell and Veloso, 88], [Winston, 80] and elsewhere fundamentally rely upon a large and richly-indexed memory of experiences coupled with some mechanism for recalling the right memory at the right time.

For analogical reasoning or case-based reasoning systems to work well, they must be able to retrieve memories of prior experiences that bear some interesting similarity to a given new problem or situation. The key here is *interesting* similarity: recalling a prior case only helps a reasoner to the degree to which that prior case shares some important functional or causal characteristics with the current problem. If, for example, a shop scheduler is trying to expedite the production of a particular part by having two machines work on it at the same time, prior cases in which it successfully or unsuccessfully tried to speed up production might be useful as might prior cases in which it tried to use those two machines together. But prior cases involving the manufacture of class 7B flanges at 3:30 on Tuesday afternoons while it was 78 degrees in the shop and while machine 4 was working on a type 2C bracket assembly, although they have all these facts in common with the current situation, are likely to shed little light on the current problem. Although the similarities between those cases and the current one are numerous, they simply don't bear upon the problem that the scheduler is trying to solve.

How to characterize

The problem is describing or characterizing the current problem. Once a system is able to describe the situation presented above, for example, as "trying to speed up production by scheduling multiple agents to work on the same job at the same time," it might be reminded of some cases where this plan worked and some where it did not, perhaps because two machines were trying to perform incompatible tasks, or perhaps because they got in each others' way. Analyzing the differences between these past cases and the current situation might indicate whether or not the plan was a good idea; it might also suggest additional planning steps that might be necessary to anticipate and avoid failures. The machines' actions might be coordinated, for example, to prevent some bad interaction.

But how can a system derive that kind of description? Not only was this characterization of the situation not present in the original or "perceptual" description, but it is also impossible to derive from any boolean combination or weighting of raw perceptual features. If the features available to the shop scheduler consisted of a set of readings from instruments and sensors around the shop plus a list of what machine was working on what part, no weighting or boolean combination of these features would get us the remindings we wanted in the prior example. "Machine 4 and Machine 8 both working on **the same** part" is a description that cannot be so derived. "Two machines working on the same part" even more so.

So an effective case retriever must not only face the problem of choosing which features of a given situation description are relevant retrieval cues in the context of this situation, it also must extract or derive some abstract features that are not initially present in that description, so that those abstract features can be used as search keys or as part of similarity metrics. The task of extracting those features, or of characterizing the current situation, is an inseparable part of the task of memory search and should be so considered theoretically. Memory is not just the process of starting from some description of the input and using it to search. A theory of memory must include a theory of how that description is derived.

EXTRACTION AND SEARCH

Much work in AI memory has either explicitly or implicitly separated the task of deriving abstract descriptions from the task of actually searching memory for objects that match those abstract descriptions, and have focused on the latter. For example, much progress has been made in the memory-based reasoning paradigm (see [Stanfill and Waltz, 88]). Connectionist approaches, too, are very good at deciding how to weight features to measure case similarity, but they do not deal with the problem of how raw data gets turned into sets of features in the first place, nor do they deal with how new features can be learned. Some systems are built on the assumption that input cases come already described in the same representational language as was used to describe the cases already in memory, so that syntactic means can be used to measure similarity. This has been a necessary assumption to allow work to proceed on the mechanics of memory organization and search, but it begs a question that full-fledged memory-based systems will have to face: How are features extracted from raw input?

Given that mechanisms are available to do retrieval and matching based on weighted vectors of features, it is tempting to say that some kind of parsing or feature extraction process should be

run over the data to extract the abstract features and weight them, and that then the original and derived features should subsequently be used as input to the retrieval process. But this approach is problematic for several reasons.

Complexity of "parsing"

One problem with this approach is that "parsing" or abstract feature extraction, can be arbitrarily complex when attempted bottom-up. Although features like "multiple machines working on the same part" as described above are quickly and easily calculable from input, others may be much less so. There may be an arbitrarily large number or abstract features that one might potentially want to derive from input, any of them potentially arbitrarily costly to infer. Since the feature extraction process does not know what is in memory and how memory is organized and searched, it might expend inferential cost on extracting features from the input that do not turn out to be particularly useful indices. It is clear that we don't want to extract all possible abstract features before searching memory, we just want to get some reasonable set of them.

Unfortunately, deciding what constitutes a reasonable set of abstract features from a particular episode requires having an abstract thematic understanding of that episode, which is the very problem we were trying to solve in the first place. There is a methodological circularity to this approach. What process can provide this abstract thematic understanding? One that relies on retrieving relevant cases?

Features not static

A second problem is that the set of abstract features that one might need to extract is not static, but depends upon the set of cases in memory. The features that one needs are the ones that describe the similarities and differences between the various cases in memory. As the set of cases changes, so must the set of abstract features. When new cases are added to memory, separate steps must be performed to select indexing features for discriminating among those new objects and to develop procedures for extracting those features from input. When a new indexing feature is learned, all objects in memory must be reexamined to determine whether or not they embody that feature; those that do must be appropriately re-indexed.

Expressing retrieval goals

A third important design consideration for case memories is that there is no one correct or closest match in memory to a given new experience. What constitutes a good match depends upon the goals of the system processing the new event. Different retrieval goals will yield different remindings, and it is important that a retrieval scheme be able to take into account the system's retrieval goals.

An example of this can be found in the discussion of the SWALE case-based explainer system ([Schank, 86], [Kass and Owens, 88], [Leake, 88]). One of the examples the system was called upon to explain was the death of Swale, a three-year-old race horse who died mysteriously, one week after winning the prestigious Belmont Stakes race.

What constitutes a satisfactory explanation of this kind of example depends upon the goals of the explainer. Therefore, the kind of reminding (and therefore the abstract description of Swale's

death) that is appropriate depends on the goals of the explainer as well. An insurance examiner, for example, might be reminded of the case of a valuable painting that mysteriously disappeared a year earlier, in what turned out to be a fake burglary staged by the owner to collect the insurance money. A veterinarian might be reminded of the cow that mysteriously died the previous week and begin investigation to see if the medical causes were the same. A racing examiner might be reminded of other cases of one competitor trying to disable another and might suspect the owners of Swale's competitors. A gambler might be reminded of other examples of an odds-on favorite suddenly being disabled or otherwise removed from competition. Each of these individuals will retrieve different remindings from memory because each has described Swale's death in different terms. Each description is equally correct, but each leads to a different path of explanatory reasoning.

It is difficult to account for these differences in retrieval with a system that separates feature extraction from the rest of memory. It is unreasonable to assume that veterinarians, insurance adjusters and gamblers have totally different processes for extracting features from situations. It is possible that the differences could be accounted for by some process that maps retrieval goals to predictive features, as discussed by [Stepp and Michalski, 86] or [Seifert, 88]. This could be used to weight the importance of features depending upon how relevant they were to the current set of retrieval goals. But this approach leaves unanswered the question of how retrieval goals and predictive features are linked together.

A more satisfying explanation of the differences in how the individuals above explained the same event is that abstract feature extraction is driven by the case libraries of each of these individuals. The veterinarian has a large case library of animal diseases and consequently describe the event in terms of features that can discriminate among these cases. Likewise an insurance examiner is discriminating among a second, different library of cases, and a gambler among a third. Each of these individuals extracts, from the story, the features necessary to discriminate among the cases in his own memory. Each individual can be using the same kind of mechanism to extract abstract features from concrete descriptions of situations, but that mechanism is being driven by a different case library in each case, and so results in a different set of features being extracted, a different case retrieved, and a different explanation generated.

INTEGRATING EXTRACTION WITH RETRIEVAL

Some of these issues can be addressed by integrating feature extraction and case retrieval as much as possible. Instead of trying to extract all possible abstract features from input and then using that set of features as a retrieval cue, a system can allow feature extraction and memory search to proceed incrementally. Each time a new feature is extracted from the input it changes the pool of candidate cases that might apply to the current situation; each time the pool of candidates changes it suggests different features that should be extracted from input to try to discriminate among the candidates.

Playing "20 questions"

The model for this approach is that, rather than the front-end or feature extraction portion of the system telling memory what the input case looks like and then memory coming up with a match,

memory is now playing a game of "20 questions" with the feature extraction process, asking for features as it needs them to discriminate among its known cases. Since feature extraction is expensive, memory is trying to ask as few "questions" as possible; seeking maximum payoff for its inferential cost with each question. The issue to resolve is how memory should decide what question to ask next.

The simplest and least interesting way to play "20 questions" is via a discrimination tree. the tree is balanced, the program can choose the correct match from among n cases by asking about log(n) features. But the problem with using this approach as a model for case retrieval is that a discrimination tree is a static object. It must be set up at the time the system is built, and, although it can be modified by adding cases and new discriminating features, it is computationally expensive to reorganize. Reorganizing a tree dynamically for each query is prohibitively expensive, so taking into account changeable items like retrieval goals is difficult. Furthermore, a tree-based retrieval algorithm will have difficulty if it asks the feature extraction process for a particular feature and receives the answer "I don't know" or "That's too expensive to calculate." What an integrated approach to feature extraction and memory search needs is a more flexible and dynamic way of playing "20 questions".

The object is to ask about the feature that offers the most information content for the least inferential cost, subject to the current retrieval goals of the system. Part of this problem is difficult: there is no good way of calculating *a priori* the difficulty of inferring any given feature. The best one can do is to remember how difficult it has been in the past to determine the presence or absence of that feature and use that cost as an estimate. Barring any other information, one can make the erroneous but necessary simplifying assumption that all abstract features are equally difficult to extract.

Fortunately, the likely utility of a feature as a retrieval cue is more easily estimable. The simplest basis is that of information content. A feature that is present in or absent from all of a given set of candidates is not worth extracting from new input, because it does not narrow down the space of candidate matches. On the other hand, a feature that is present in about half of the candidate cases is worth examining because knowing whether or not it is present in the new input case cuts the pool of remaining candidate matches in half. Accordingly, an important part of an incremental retrieval algorithm is a scheme for suggesting, given a set of cases, features that, if known to be present or absent in the input case, would discriminate among them. These are the features that the system should try to extract from input. Each time a feature is extracted it can be used to change the members of the current candidate set of cases; each time the set of candidate cases changes it would suggest a new set of discriminating features.

Parallel implementation

The ANON program is an attempt to integrate feature extraction and memory search. It plays the role of a memory in service of an overarching case-based planning or explanation system. Its behavior is to suggest features to an (external) feature extraction process, and, based upon whether each suggested feature is found to be present, absent or too expensive to infer, to continually narrow a set of candidate cases until either it finds either one case or a group of cases that do not differ in their causal implications a propos the current problem. Its case library is a set of abstract knowledge structures characterizing stereotypical plan failure situations. These

cases correspond to common advice-giving proverbs like *too many cooks spoil the broth.* (See [Dyer, 82] for a discussion of the relationship of proverbs to stereotypical situations.)

ANON's memory contains about 1000 of these proverbial cases. In a full planning system they would be represented in much more causal detail than has been done to date, but the purpose of this system is to explore retrieval strategies in a large case library. A form for deeper causal representation for these proverbial knowledge structures is similar to the Explanation Patterns (XPs) described by [Schank, 86] or [Kass and Owens, 88].

Each case is represented on one processor of a Connection Machine parallel processor as is each known indexing feature. The system operates in two alternating modes: retrieval and feature suggestion.

Retrieval mode consists of moving from a set of features to a set of cases that embody those features. This is done by instructing the desired features to broadcast to the cases that embody them, the cases can then be ordered according to how many of the desired features they embody. The mechanism is in place here to assign weights to the features based on any of the criteria discussed above; the program currently assumes equal weights. This kind of retrieval is discussed in more detail by [Stanfill and Kahle, 86].

Feature suggestion mode is the more important mode; it is the means whereby the system picks the next feature to try to extract from input. The key to being able to suggest features is to be able to examine any group of cases and to suggest a feature that will discriminate among them. This can be done with any two cases just by comparing the features that participate in their causal structure. If the cases suggest different causal conclusions then there must be a feature in their causal structure that discriminates between them.

With larger groups of cases that cannot be compared individually with each other, the parallel implementation approximates this feature suggestion behavior by means of calculating *representativeness*. To find out how representative a given feature is of the currently active candidate pool, each processor corresponding to a currently active candidate case is instructed to send a message to each processor corresponding to a feature that is represented in that case. Features can thus be ordered on the basis of how well each represents the common qualities of all the cases in the candidate pool.

Features that are highly representative or not at all representative of a given pool are not likely to be good discriminators: they are not worth extracting from the new input case because they cannot be used to reduce the number of candidates in the pool. Features that are representative of about half the candidates in a pool, on the other hand, are very good discriminators. These are the features that the system suggests trying to infer next.

Of course simply counting the number of cases that each feature would index is only the crudest possible use of this calculation strategy. Just as features can be weighted in retrieval mode, cases can be weighted in feature suggestion mode. Representativeness does not have to be determined on the basis of numerical case counts; it can be determined on the basis of weighted case counts. The algorithm can be used to select features that divide the pool of cases not in half, but in half on a weight-adjusted basis. The source of these weights can be, for example, based upon features correlated with the retrieval goals discussed previously.

Features of this algorithm

Since this approach uses the contents of cases for organizing memory and requires no separate indexing knowledge, it makes it easy for the system to accept new cases. Since the cases themselves suggest the indexing features that would discriminate between themselves and other cases in memory, the new cases are included in the next memory retrieval cycle without the need to perform any explicit reorganization or re-indexing.

Adding new indexing knowledge to existing cases, on the other hand, is slightly more complicated. When the retrieval process encounters two cases that cannot be discriminated from each other, that indicates that one or both of the cases are not represented in enough detail. Currently the system is only able to indicate cases that need their degree of detail enhanced; it is not able to add detail to a case representation. The intention is to add detail whenever the system is unable to discriminate between two cases in memory; the mechanism for doing so is to build a causal explanation of the difference between the cases and use the features that participate in that explanation as new discriminating features.

CONCLUSIONS

No matter what kind of architecture one uses to accomplish the actual details of memory search, one must make a strong commitment to the idea of abstract features. The cases of which one wants to be reminded are often those that share abstract, rather than concrete or surface-level similarity to the current problem situation. Often these abstract features cannot be calculated by boolean combination or weightings of the concrete perceptual features and must therefore be derived from complicated and difficult-to-calculate relationships between perceptual features.

But, just because one is committed to the idea of abstract features does not mean that implementations must have a feature extraction or parsing process separate from the memory search process. There is no reason why all possible abstract features must be extracted from the input before the search of memory can begin. In fact, the approach of extracting a vector of abstract features and then using that whole vector as a search key is too costly in terms of processing resources. Instead, a more incremental approach that lets the contents of memory determine what features need to be extracted from input makes much better utilization of a system's inferencing power. This approach controls the complexity of inference, expresses retrieval goals as a function of the cases already in memory, and has desirable properties with regard to the reorganization of memory to take into account new experiences and new indexing features.

Obviously this approach does not solve the problem of how to recognize any given feature in input; that is still an open question. What it does accomplish, however, is to show how a system can have a great many abstract and difficult-to-infer features as part of its indexing vocabulary without having to identify the presence or absence of each one of them every time it processes a new piece of input.

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under contract N00014-85-K-0108 and by the Air Force Office of

Scientific Research under contracts AFOSR-85-0343, AFOSR-89-0100 and F49620-88-C-0058. Larry Birnbaum and Alex Kass provided helpful comments on this material.

References

[Ashley and Rissland, 87] K. Ashley and E. Rissland. Compare and contrast, a test of expertise. In *Proceedings of the Sixth Annual National Conference on Artificial Intelligence*, pages 273–284, Palo Alto, 1987. AAAI, Morgan Kaufmann, Inc.

[Carbonell and Veloso, 88] J. Carbonell and M. Veloso. Integrating derivational analogy into a general problem solving architecture. In J. Kolodner, editor, *Proceedings of a Workshop on Case-Based Reasoning*, pages 104–124, Palo Alto, 1988. Defense Advanced Research Projects Agency, Morgan Kaufmann, Inc.

[Dyer, 82] M. Dyer. In-depth understanding: A computer model of integrated processing for narrative comprehension. Technical Report 219, Yale University Department of Computer Science, May 1982.

[Hammond, 86] K. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, 1986. Technical Report 488.

[Kass and Owens, 88] A. Kass and C. Owens. Learning new explanations by incremental adaptation. In *Proceedings of the 1988 AAAI Spring Symposium on Explanation-Based Learning*. AAAI, 1988.

[Kolodner, 87] J. Kolodner. Extending problem solver capabilities through case-based inference. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 167–178, Los Altos, CA, June 1987. University of California, Irvine, Morgan Kaufman Publishers, Inc.

[Leake, 88] D. B. Leake. Using explainer needs to judge operationality. In *Proceedings of the 1988 AAAI Spring Symposium on Explanation-based Learning*. AAAI, 1988.

[Schank, 86] R. Schank. *Explanation Patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Seifert, 88] C. Seifert. A retrieval model for case-based memory. In E. Rissland and J. King, editors, *Proceedings of a Case-Based Reasoning Workshop*, pages 120–125. AAAI, 1988.

[Simpson, 85] R. Simpson. *A Computer Model of Case-based Reasoning in Problem-solving: An Investigation in the Domain of Dispute Mediation*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1985.

[Stanfill and Kahle, 86] C. Stanfill and B. Kahle. Parallel free-text search on the connection machine system. *Communications of the ACM*, 29(12):1213–1228, December 1986.

[Stanfill and Waltz, 88] C. Stanfill and D. Waltz. The memory-based reasoning paradigm. In J. Kolodner, editor, *Proceedings of a Workshop on Case-Based Reasoning*, pages 414–424, Palo Alto, 1988. Defense Advanced Research Projects Agency, Morgan Kaufmann, Inc.

[Stepp and Michalski, 86] R. E. Stepp, III and R. S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, Volume II*, chapter 17, pages 471–498. Morgan Kauffmann, Los Altos, CA, 1986.

[Sycara, 87] E. P. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-based and Analytic Methods*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[Winston, 80] P. Winston. Learning and reasoning by analogy. *Communications of the ACM*, 23(12):689–703, 1980.