Evaluating a Case-Based Planning System *

Brian Kettler and James Hendler

Department of Computer Science
University of Maryland
College Park, MD 20742
{kettler,hendler}@cs.umd.edu
Phone: 301-405-7027

Abstract

CaPER is a case-based planning system that makes use of the massive parallelism of the Connection Machine to access a large memory of hundreds or more cases that are not pre-indexed. Preliminary empirical results of these retrieval methods indicate their viability and scalability. These results have motivated the retrieval-intensive design of the complete system, which includes proposed plan adaptation methods that use plan validation structure information and a proposed retrieval planning procedure. The design tradeoffs in the plan retrieval and adaptation processes are discussed, and criteria for empirically evaluating these processes are presented. Many of the proposed areas for evaluation apply to other case-based planning and case-based reasoning systems.

Introduction

There are two aspects of the evaluation of any large computer system, particularly in AI. There is formal evaluation, in which experimental comparisons are made between one technique and others, or on varied aspects within the system. There is also the development of a set of axes on which to evaluate such systems and the placing of the individual system along these axes. In this short paper, we will try to develop such a set of dimensions on which to compare case-based systems, and we'll show how the case-based planning system we are developing, CaPER, fits into such a categorization.

We also describe some early evaluation of aspects of the CaPER system, particularly its memory retrieval performance. While this does not provide a direct comparison between CaPER and other case-based planning (CBP) systems, it does let us evaluate the performance of a critical aspect of our system, and provides a basis for later comparisons between CaPER and other CBP systems. Thus we also describe some of the experimentation which we hope to perform in the future to compare CaPER's techniques with other systems and/or approaches.

*This research was supported in part by grants from NSF(IRI-8907890), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039 and by ARI (MDA-903-92-R-0035, subcontract through Microelectronics and Design, Inc.).

The CaPER System

CaPER is a case-based planning system that makes use of the massive parallelism of the Connection Machine to retrieve cases from a large case memory. The focus of this research is to determine how to exploit the fast and flexible retrieval provided by the massively-parallel methods used. The primary potential benefits of case-based planning over generative planning include efficiencies through plan reuse (by remembering past successes and failures) and the ability to function in domains for which complete causal knowledge (i.e, fully specified plan operators) is lacking. On the other hand, CBP systems may waste time searching irrelevant parts of case memory or in trying to apply an irrelevant previous plan retrieved. In the absence of strong causal domain knowledge, heuristic plan adaptation methods may produce incorrect plans.

There have also been few empirical studies of CBP systems. The benefits of plan reuse have been shown in PRIAR (Kambhampati and Hendler 1992) and PRODIGY/Analogy (Veloso 1992), for example. Neither of these systems are general-purpose CBP systems (both, for example, work in conjunction with a generative planner), although many of their techniques are certainly widely applicable. Few CBR systems have been evaluated using real-world-sized casebases of hundreds or thousands of cases, and thus it remains unclear whether their methods will scale up.

Empirical evaluation of CBR/CBP systems can be done in at least three areas: performance, domain-engineering effort, and applicability (i.e., class of problems they solve). This evaluation should be done on varying casebases, problem specifications, etc. This evaluation can take place for individual components, as well as the aggregate system.

We have done some preliminary evaluation of the

157

CaPER's retrieval component on different domains. Once the complete system is implemented, a full evaluation will be performed. This will include an evaluation of the various components of CaPER, as well as its overall performance. Comparison with other CBP systems and with generative planners would also be useful. This paper discusses some areas for future evaluation of CaPER's components (based on their current design) in terms of performance, domain engineering effort, and applicability. Many of the planned evaluations could be done for other CBP/CBR systems, while others are specific to the particular methods used. Many of these methods were chosen with particular behavioral goals in mind (i.e, performance, etc.). How some of these goals have influenced the design thus far will also be discussed.

Overview of CaPER

The overall architecture of CaPER is similar to that of other CBP systems (e.g., CHEF (Hammond 1990)) and is shown in figure 1. The retrieval process (Plan Retriever component) is described briefly in this section (see (Kettler *et al.* 1994) for more details). The proposed adaptation process involving the Plan Merger, Plan Checker, and Plan Modifier components is described briefly in section "Adaptation". (Here "adaptation" includes both the initial creation of the target plan from source plans and its subsequent repair to fix any failures found). The Plan Tester component gets detailed diagnostic feedback on the proposed target plan from a human.

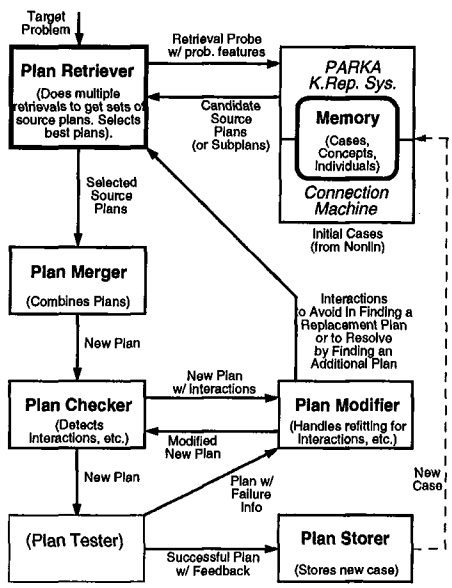

Figure 1: CaPER's Architecture (Top-Level Processes and Components)

CaPER's memory is implemented using Parka (Evett *et al.* 1994), a massively-parallel knowledge representation system. Episodic knowledge (particular plans, actions, and objects) is stored together with conceptual knowledge (taxonomies, partonomies, etc.) in one large semantic network stored on the Connection Machine. A sample piece of memory is shown in figure 2.

Parka supports very fast inferencing procedures: for example, recognition queries of the form "find all x such that $p_1(x, c_1) \wedge p_2(x, c_2) \wedge \cdots \wedge p_m(x, c_m)$" – where p_i is a slot relating frame x with value c_i) – run in $O(m + d)$, where d is the depth of the network.[1] The PARADYME system (Kolodner and Thau 1988) is similar to Parka in that it was used to implement the memory of a CBR system (Kolodner 1989), although the parallel representations and methods used differed (see (Evett *et al.* 1994) for a comparison). The implications of Parka's fast inferencing methods are discussed in the next section.

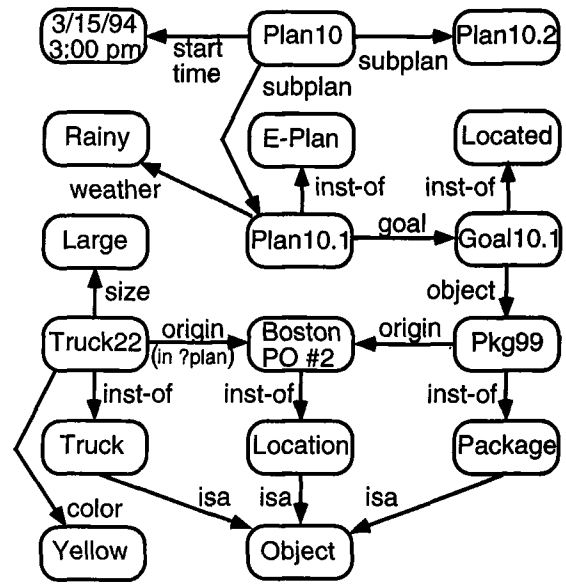

Figure 2: Sample Piece of CaPER's Memory for Transportation Logistics Domain. ("inst-of" links are "Instance Of" ($\in$). "isa" links are $\subset$ relations.)

The initial CaPER prototype retriever was tested on a car construction "toy" domain. CaPER is currently being tested on a simplified transport logistics (package delivery) domain based on one used in (Veloso 1992). Some of CaPER's retrieval mechanisms have been used for planning by derivational analogy in a protein sequencing experiment domain (Kettler and Darden 1993). It will probably be necessary to develop another richer and more realistic domain for testing

[1] Since d remains fairly constant as the net grows, the time for such queries is essentially independent of the size of the net (d is typically $<<$ 20). See (Evett *et al.* 1994) for details.

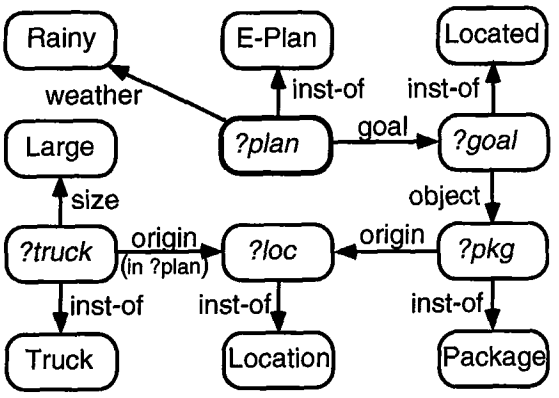

Figure 3: Sample Retrieval Probe Transportation Logistics Query "find all (sub)plans with a goal of delivering a package whose origin is the same as that of an available large truck. Prefer plans done on days with weather similar to the present weather (rainy)." This probe graph would match parts of the sample semantic net in figure 2.

some of the proposed features of CaPER.

In CaPER, to retrieve a case[2] a retrieval description is formed that contains various features (surface or derived) from the target situation. These features may include plan goals, structural (spatial or temporal) relations among objects, etc. The retrieval description is converted to one or more probe graphs such as the one shown in figure 3. These are matched against the semantic network by the Parka Structure Matcher (Andersen *et al.* 1994), which returns sets of variable bindings.

Exploiting Retrieval Flexibility

The availability of Parka's fast parallel methods has driven the design of CaPER. Because such methods are fast, memory does not have to be pre-indexed. Pre-indexing is the designation of indices (particular case features) prior to *case retrieval time* through which cases are subsequently retrieved.

CaPER does not use indices, where indices are construed in the traditional way as pointers for retrieval. Recently indices have been construed more broadly (e.g., (Kolodner 1993)) to include labels or annotations that, particularly in some parallel implementations, are used for discriminating among retrieved cases. The term "index" is used in this paper in its traditional, narrower sense.

For efficiency reasons, serial CBR systems often use pre-indexing in conjunction with a specific memory organization such as a discrimination net to restrict the case retrieval search to a small subset of the casebase.

[2]In this paper we generally use "case" synonymously with "plan" because a case contains at least one plan that solves its planning problem.

Hence pre-indexing can constrain case retrieval: the system, or more often a human, has the difficult task of anticipating which features of a case will prove useful for retrieving it in the future. Pre-indexing schemes are often domain/task-specific.

In CaPER, a case can potentially be retrieved via any of its target features: i.e., any feature of the retrieval description (including complex derived features) can be included in the Structure Matcher probe graph.

Retrieving Relevant Cases

The relevance problem — selecting retrieval criteria such that a reasonable number (i.e., not too many nor too few) of relevant cases is recalled — must still be addressed in CaPER. Like other CBR systems, CaPER needs to know or learn features of the target situation which should be used to retrieve (i.e., match) relevant cases. Unlike other CBR systems, the set of features from which these are chosen is *not* restricted a priori to a set of features used to pre-index the casebase. Thus CaPER can use potentially any features (surface, derived, etc.) from the target situation. This greater flexibility comes potentially at the expense of more work at retrieval time to determine which features should be included in the retrieval probe. This possible tradeoff needs to be assessed empirically.

To determine which features should be used to match cases and also which features should be used to rank the cases retrieved, knowledge about what kinds of features are indicative of case relevance for what kinds of target problems could be acquired and stored in "metamemory". This dynamic knowledge can come from a human knowledge engineer or preferably can be learned by the system. A proposed enhancement to CaPER includes treating retrieval description formulation as a planning task in and of itself. This will enable CaPER to reason explicitly about what features to compute and include in the retrieval probe. This planning subtask could make use of generative or case-based methods.

Frequent Retrieval

Because in CaPER retrieval is fairly cheap and flexible (due to the lack of memory pre-indexing), the design philosophy has been to make the most of memory. CaPER's Plan Retriever (PR) component retrieves plans from memory that match features in the retrieval description. It ranks the plans retrieved and returns them for further processing. The PR can be called multiple times. For example, CaPER may have three target goals. On its initial call, the PR may return a single source plan (S_1) – or a subplan of a source plan – that addresses two of these goals. The PR can then be called to find an source plan that addresses the remaining goal that was not matched. This source plan S_2 can be combined with S_1 by the Plan Merger to get a single plan which can then be adapted for the target problem by the Plan Modifier.

Adaptation

Retrieving multiple plans/subplans that address the target problem can result in a combined plan that achieves many of the target goals and hence requires less adaptation work. On the other hand, merging plans from multiple source cases may result in harmful interactions in the resulting plan. These interactions may be expensive to detect by plan analysis and/or by plan testing and costly to correct.

In CaPER, the Plan Checker attempts to detect interactions. Interactions found by the Plan Checker or during plan testing are handled by the Plan Modifier. The Plan Checker can make use of plan validation structure information (Kambhampati and Hendler 1992), which captures the dependencies among actions in the plan. This information is used to detect differences between the old source plans and the new target plan such as different goals, different features in the initial situations, interactions, etc. These differences determine applicable modifications including plan pruning, refitting, and remapping.

Given a predicted interaction between actions in two different plans, it is also possible to reinvoke the PR to retrieve a patch plan (i.e., a "white knight") that corrects the interaction or to retrieve a replacement plan(s) that avoids the interaction. Information about the problem to avoid in the target plan could be added to the retrieval description when the PR is reinvoked. Other CBP systems such as CHEF add information about problems to avoid to the retrieval probe (although CHEF does so prior to any retrieval and thus can only avoid failures detected during previous planning episodes). To find a plan that avoids an interaction or a patch that fixes some interaction while avoiding others, CaPER could match features from the target problem/plan against the plan validation structure information stored in memory.

In may even prove beneficial to interleave retrieval with adaptation. In the above example, after S_1 is retrieved it would be modified for the target situation. Any problems encountered during modification or special plan dependencies noted could then potentially be used in the retrieval description used to obtain a "better-fitting" S_2.

Comparisons

CaPER's plan retrieval and plan adaptation components (and those of other CBP systems) can be evaluated in the areas of performance (planning cost and plan quality), domain engineering, and general applicability of methods used.

Retrieval versus Adaptation

There is often a tradeoff between the effort spent finding relevant source cases and that spent adapting the these cases (plans)found to the target problem. In a CBP system, the most "relevant" case is one whose plan requires the least adaptation effort (time). Adaptation often requires strong causal knowledge about the objects and actions in the domain. Adaptation itself may fail in the absence of such knowledge as the system may not be able to detect or remedy interactions and other plan failures. Thus in cases where adaptation is expensive or impossible, it is desirable to find the best source cases possible. The system should not, however, take forever to find cases, particularly when planning must be fast and/or the world (and hence the target situation) is changing.

Some performance issues include:

1. What is the cost (time and space) of retrieval?
2. Are the cases retrieved relevant? What is the cost (e.g., extra time spent on adaptation, etc.) of retrieving an irrelevant or misleading plan?
3. Does the cost (time) of using multiple source plans (i.e., merging them and detecting and handling any resulting interactions among them) outweigh the cost of using a single plan? Are these methods correct relative to the completeness of the domain knowledge?
4. What is the cost (time and space) of adaptation?
5. Are the adaptation methods complete and correct?
6. Is the cost (time and space) warranted for using plan validation structure information (when available) for detecting areas for adaptation and suggesting potential repairs or could some cheaper, alternative method be used?

Evaluating Retrieval in CaPER CaPER's Plan Retriever converts a retrieval description to one or more probe graphs to be processed by the Parka Structure Matcher. We have evaluated the time to process these probes on both the car construction and transport logistics domains being used to test CaPER. The purpose of these experiments was to assess the absolute parallel retrieval times and the scalability of the parallel methods used, for a variety of representative retrieval probes (generated by hand) and casebase sizes. The probes were matched to an *unindexed* memory using Parka on the massively parallel Connection Machine[3] and again using a serial version of Parka[4].

These results, detailed in (Kettler *et al.* 1993; Kettler *et al.* 1994), show parallel retrieval times of about a second, even for a 100 case memory which contained 1213 (sub)plans (8616 frames). Not only are the absolute parallel retrieval times low but they scale well to large casebases: the retrieval times appear to grow better than logarithmically in the number of cases (frames).[5] The actual complexity of the underlying

[3] a CM-2 with 32K virtual (16K real) processors

[4] a Macintosh. A direct comparison of serial to parallel times would not make sense, given the disparity in total processing power

[5] An optimized *serial* implementation on an indexed

subgraph matching (constraint satisfaction) algorithm on actual casebases is much less than the worst case complexity (exponential in the number of binary constraints) due to several factors discussed in (Andersen *et al.* 1994).

The cases retrieved were judged "relevant" by a human. Of course once the plan adaptation component is complete, we will need to assess relevance by the amount of plan adaptation effort required. We are now building a 1000 case (approx. 100K frames) for the transport logistics domain and believe even large casebases can be supported with comparable retrieval performance.[6]

In addition to their time efficiency, CaPER's retrieval procedures require no space to store special-purpose indexing structures built on top of the casebase. CaPER can retrieve subplans with little storage overhead. Using Parka's mechanisms, contextual features can be inherited by a subplan from its parent plan and thus does not need to be explicitly stored with the subplan (as is done in CELIA's snippets (Redmond 1990), for example).

The quality of retrieval in CaPER remains to be thoroughly evaluated. The simple similarity metrics used thus far[7] appear to be useful in getting good cases in the car domain (as judged by CaPER's developers). In the more complicated domains, it is less clear as to the relevance of the retrieved cases. This is primarily due to the potential complexity of features (i.e., complex derived relationships among locations, vehicles, and packages in initial situations appear relevant) and the mapping/matching complexities (i.e., each target package may potentially mapped to one of several source packages).

While human similarity judgements have been used to initially develop and tune retrieval, match, and similarity criteria in CaPER, the true relevance of a plan retrieved is inversely proportional to the amount of adaptation effort it takes, which will need to be measured (see below) once the required procedures have been completely implemented. The standard information retrieval metrics of recall (percentage of relevant cases that were retrieved) and precision (percentage of retrieved cases that were relevant) could then be applied.

The cost of using an inappropriate source plan needs to be assessed in terms of total retrieval plus adaptation effort required versus having solved the problem from scratch. An inappropriate plan can result from faulty retrieval, match, or similarity criteria. In CaPER we hope to minimize faulty retrieval by being able to issue very precise queries (some generated by the "planning to retrieve" process), by merging multiple plans retrieved, and by being able to retrieve additional plans, possibly using information about plan dependencies and failures. The cost of these measures needs to be assessed.

An inappropriate target plan can be created due to errors in mapping target situation objects/relations to those in the source plan(s). Matching heuristics may be incorrect. The Parka Structure Matcher could be used to map objects by virtue of the relations (spatial, causal, etc.) in which they participate (such as is done in SME (Falkenhainer *et al.* 1990)). Alternative mappings returned by the Structure Matcher could also be stored or generated on demand (trading time for storage). An inappropriate plan may also be chosen due to bad similarity criteria. It is therefore desirable to dynamically modify similarity metrics (via weight adjustment, etc.) based on the relevancy of the cases they select for.

Finally the cost of using multiple source plans needs to be compared to the cost of using a single source plan. This can be done empirically be switching CaPER's mode of operation. The cost of using multiple source plans includes the cost of issuing multiple retrieval probes to get the plans (this is fairly cheap as noted above), merging them, and detecting interactions during adaptation (see below). The merging of plans is guided by the relationships among the goals addressed by the individual source plans. A least-commitment strategy with respect to subplan or action ordering could be used. The adaptation effort for multiple plans is in part a function of the modularity of domain plans.

Evaluating Adaptation in CaPER The effort (time cost) of adaptation includes the time to detect problems with the target plan, to find applicable fixes, to select one or more fixes, and to apply the fix (i.e., modify the plan). In CaPER, the first two of these activities are done by the Plan Checker, the last two by the Plan Modifier.

Since CaPER's proposed adaptation methods include using strong domain knowledge (i.e., plan validation structures, etc.), it should be possible to detect interactions and suggest possible fixes for them. Some of these fixes could be done by heuristic methods or by retrieving patch or replacement plans. Another possibility is to use a generative planner for plan refitting (like PRIAR does). The methods that use plan validation structure have proven efficient in PRIAR and should be in CaPER as well. The extra frames required to store this information in memory is small and has negligible adverse impact on the retrieval times. It is also desirable to have CaPER function in domains where plan validation structure information is incomplete.

memory would have $O(\log n)$ performance if a balanced discrimination tree were used. Of course such a system would have the disadvantages of pre-indexing previously mentioned.

[6] As in all data-parallel applications, this assumes that the amount of parallel hardware can scale with the problem size, if necessary.

[7] a weighted combination of features taking into account the level of abstractness of matches

Other plan adaptations include substitution (reinstantiation) of the source plans in the target situation. This may involve additional matching (heuristic or via the Parka Structure Matcher).

Storing alternative potential fixes also trades storage for time and may not be warranted. The assumption is that it is desirable to try the "best" fix first (i.e., the highest expected chance of success per cost, with repairs that transform the plan typically costing more than those that make substitutions in it). Certain adaptations like pruning extra goals via PRIAR-like methods from the plan can be done with certainty of success, assuming the plan validation structure information is accurate.

CaPER will initially apply repairs cumulatively. Hence a newly applied repair may generate additional problems to be fixed. This trades potentially longer plans for planning time, rather than diagnose potential interactions among repairs and potentially rolling back one of them to produce a new version of the target plan.

Domain-Engineering Issues

Some domain-engineering issues include:

1. How do initial cases get into the casebase? What is the quality of these initial cases?
2. What kinds of domain-specific knowledge need to be supplied for retrieval, match, and similarity criteria? How good is this knowledge? How can it be modified?
3. What kinds of domain-specific adaptation methods are required? Do they require strong causal domain knowledge? How can these methods be modified?

In CaPER, plans (along with problem decomposition and plan validation structure information) for the initial cases are generated using UMCP Nonlin (Ghosh *et al.* 1992), a Common Lisp version of Tate's Nonlin planner, on randomly-generated planning problems. This allows us to easily build large casebases. Because problems are randomly generated, there will be a diverse set of previously solved problems for the planner to draw upon. Planning problems and their solutions are automatically converted to cases (i.e., Parka frame definitions).

Because Nonlin produces correct (but not necessarily optimal) plans, so we can rely on the quality of the initial cases. Of course this assumes some causal domain knowledge is available to be encoded as Nonlin hierarchical task networks and operators. In the absence of such knowledge, cases would have to be acquired via other means and care would have to be taken to ensure that undetected bad plans do not get put into the casebase. Plan validation structure information could be manually annotated to plans.

Because cases are not pre-indexed, no manual indexing of cases is required. Effort is still required to determine the relevant features of a target situation at case retrieval time. It is hoped that the planning to retrieve methods will allow the system reason about the relevancy of features for particular domains/tasks. In addition, match criteria and similarity metrics must be determined for particular domains/tasks. An implementation goal is to represent such knowledge declaratively (so that it is explicit and can be reasoned about) and in a modular way (e.g., via CLOS), so that it can be easily modified by a human knowledge engineer or by the system itself.

CaPER's substitution methods will require taxonomic knowledge about domain objects and their relationships. This non-episodic knowledge for each domain must initially be hand-encoded as Parka concept frames encoded and added to Parka's base ontology (which contains generic abstract concepts such as "plan", "action", etc.). PRIAR's methods for detecting plan inconsistencies and find applicable fixes require strong domain knowledge in the form of default action preconditions, filter conditions, and effects. This knowledge is also hand-encoded as Parka frames.

Finally, there are some end-user ease of use issues. An end-user (as opposed to a knowledge engineer) can interact with CaPER during problem specification, target plan testing, and possibly to provide input when the system gets "stuck" (e.g., can't find an applicable adaptation method, etc.). Because the user can provide the target problem specification, it may be useful to check for and/or handle an incorrectly or incompletely specified problem. Because certain failures may only be detectable during plan testing (by a human), the user needs to be presented with the target plan (possibly with the source cases by way of explanation), so that he/she can provide detailed feedback from plan execution.[8] An interface for knowledge engineers and end-users will have to be developed and evaluated as to its usability.

Applicability Issues

CaPER's CBP techniques are intended to be general-purpose with domain knowledge being integrated in a modular fashion. The memory organization of an unindexed semantic network (frames) and the low-level Parka query procedures are general-purpose too. It remains to be seen how robust CaPER's methods are in the various kinds of domains on which it is being tested, how these methods work in the presence/absence of strong domain knowledge, how dependent they are on the heuristics used, etc.

Comparative Evaluation

It is desirable to compare CaPER to other domain-independent A.I. planning systems. Metrics for evaluation include overall performance (total planning time), performance over time as new cases are added versus

[8] Automated diagnosis of plan execution failures is not an initial focus of this research.

the additional storage cost, scalability of performance as the casebase and planning problems grow in size, and plan quality (plan correctness, optimality in terms of length and resources consumed, etc.). Domain-engineering effort and applicability should also be evaluated.

CaPER could be compared to other CBP systems in which cases for some domain could be stored and in which the domain-specific knowledge (similarity metrics, etc.) could be expressed. One such interesting comparison would be to compare CaPER against a serial system that uses pre-indexing. In particular the hypothesis that the flexible parallel retrieval in CaPER will result in less adaptation effort needs to be tested. The scalability of smart serial methods needs to be compared against the CaPER's parallel methods, which require expensive hardware. CaPER could also be compared to a generative planning system, such as Nonlin. This is possible for domains in which Nonlin is used to seed the casebase. The cost versus benefits of plan reuse could be assessed.

Conclusion

The preliminary empirical evaluation of CaPER's plan retrieval methods that make use of massive parallelism have been encouraging and have motivated the design of the rest of the system with its emphasis on fast, frequent, and flexible plan retrieval. Proposed plan adaptation methods using simple substitution and PRIAR-like methods that use plan validation structure information need to be evaluated once they are completely implemented. Of particular interested is the cost and benefit of merging multiple source (sub)plans and the cost and benefit of treating retrieval description formation as a planning task. The dimensions for evaluation include performance, domain-engineering effort, and general applicability of methods used. Many of these areas for evaluation can be applied to other CBP and CBR systems so that their techniques can be better understood and applied.

References

Andersen, William A.; Hendler, James A.; Evett, Matthew P.; and Kettler, Brian P. 1994. Massively parallel matching of knowledge structures. In Kitano, Hiroaki and Hendler, James, editors 1994, *Massively Parallel Artificial Intelligence*. AAAI Press/The MIT Press, Menlo Park, California. in press.

Erol, Kutluhan; Nau, Dana S.; and Subrahmanian, V.S. 1992. On the complexity of domain-independent planning. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Menlo Park, California. AAAI Press. 381–386.

Evett, Matthew P.; Hendler, James A.; and Spector, Lee 1994. Parallel knowledge representation on the Connection Machine. *Journal of Parallel and Distributed Computing* in press.

Falkenhainer, Brian; Forbus, Kenneth D.; and Gentner, Dedre 1990. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence* 41:1–63.

Ghosh, Subrata; Hendler, James; Kambhampati, Subbarao; and Kettler, Brian 1992. *Common Lisp Nonlin Version 1.2 User Manual.* University of Maryland at College Park, Department of Computer Science, 1.2 edition.

Hammond, Kristian J. 1990. Case-based planning: A framework for planning from experience. *Cognitive Science* 14:384–443.

Kambhampati, Subbarao and Hendler, James A. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55:193–258.

Kettler, Brian P. and Darden, Lindley 1993. Protein sequencing experiment planning using analogy. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, Menlo Park, California. AAAI Press. 215–224.

Kettler, Brian P.; Hendler, James A.; Andersen, William A.; and Evett, Matthew P. 1993. Massively parallel support for case-based planning. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence Applications*, Washington. IEEE Computer Society Press.

Kettler, Brian P.; Hendler, James A.; Andersen, William A.; and Evett, Matthew P. 1994. Massively parallel support for case-based planning. *IEEE Expert* 8–14.

Kolodner, Janet L. and Thau, Robert 1988. Design and implementation of a case memory. Technical Report RL88-1, Thinking Machines Corporation.

Kolodner, Janet L. 1989. Judging which is the best case for a case-based reasoner. In *Proceedings of the Case-Based Reasoning Workshop*, San Mateo, California. Morgan Kaufmann Publishers.

Kolodner, Janet L. 1993. *Case-Based Reasoning.* Morgan Kaufmann Publishers, San Mateo, California.

Redmond, Michael 1990. Distributed cases for case-based reasoning: Facilitating uses of multiple cases. In *Proceedings of the Eigth National Conference on Artificial Intelligence.* 304–309.

Veloso, Manuela M. 1992. *Learning By Analogical Reasoning in General Problem Solving.* Ph.D. Dissertation, Carnegie Mellon University, School of Computer Science.