Massively Parallel Support for a Case-based Planning System

Brian P. Kettler James A. Hendler William A. Andersen

Department of Computer Science
University of Maryland
College Park, MD. 20742

Abstract

Case-based planning (CBP), a kind of case-based reasoning, is a technique in which previously generated plans (cases) are stored in memory and can be reused to solve similar planning problems in the future. CBP can save considerable time over generative planning, in which a new plan is produced from scratch. CBP thus offers a potential (heuristic) mechanism for handling intractable problems. One drawback of CBP systems has been the need for a highly structured memory to reduce retrieval times. This approach requires significant domain engineering and complex memory indexing schemes to make these planners efficient.

In contrast, our CBP system, CaPER, uses a massively parallel frame-based AI language (PARKA) and can do extremely fast retrieval of complex cases from a large, unindexed memory. The ability to do fast, frequent retrievals has many advantages: indexing is unnecessary; very large casebases can be used; memory can be probed in numerous alternate ways; and queries can be made at several levels, allowing more specific retrieval of stored plans that better fit the target problem with less adaptation. In this paper we describe CaPER's case retrieval techniques and some experimental results showing its good performance, even on large casebases.

1. INTRODUCTION

A case-based planning system solves planning problems by making use of stored plans that were used to solve analogous problems. Case-based planning (CBP) is a type of case-based reasoning (CBR), which involves the use of stored experiences ("cases"). There is strong evidence that people frequently employ this kind of analogical reasoning (e.g., (Vosniadou and Ortony 1989; Gentner 1989; Ross 1989)).

CBP systems consist of a planner and a casebase holding cases, which typically include a description of a planning problem (an initial state and goals to be achieved) and the plan(s) used to solve it (a sequence of primitive, executable actions). When a CBP system solves a new planning problem, the new plan is added to its casebase for potential re-use in the future. Thus the system learns from experience. Since feedback from the new plan's execution is also typically stored as part of the new case, the system can avoid any repeating failures it encountered.

CBP planners can be distinguished from generative planners, such as NONLIN (Tate 1976), which generate a plan from scratch by searching a space of partial plans. These systems expand a partial plan by adding actions to it and then checking the plan for helpful and harmful interactions between actions in the expanded plan — a costly process. CBP systems, in contrast, do not begin from scratch and attempt to find fully-instantiated plans with any harmful interactions already removed.

Most CBP systems use a version of the following process when given a new planning problem to solve: (1) retrieve case(s) from memory that are analogous to the current ("target") problem, (2) select one or more of these candidate cases that are most similar to the target problem, (3) adapt the selected cases (plans) to a new plan for the target problem, (4) get feedback on the new plan from its execution, (5) diagnose and repair any faults in the plan found during its execution, (6) store the repaired plan and the failure/repair information in the casebase.

The CaPER system (Cased-based Planning, Explanation, and Repair) is a case-based planner that is being developed to take advantage of the efficiencies of plan re-use while addressing some of the problems and limitations of case-based planners that use serial retrieval procedures on an indexed memory. To provide fast retrieval of cases (and other kinds of episodic knowledge) from a large, unindexed memory, CaPER makes use of the massive parallelism of the Connection Machine (CM) (Hillis 1985). The ability to quickly access memory permits frequent memory accesses. The implications of being able to go to memory often are great, and CaPER is designed to take advantage of this wherever possible.

This paper describes some of CaPER's components and the implications of fast, frequent memory access on their design. Section 1.1 describes some of the problems with the approach to case retrieval taken by most "traditional" CBP systems. Section 2 describes CaPER's memory and the PARKA system used to implement it. Section 3 describes CaPER's plan retriever. Section 4 will describe some promising empirical results of the

retriever for problems in the simplified automobile assembly domain being used to test the CaPER prototype. Section 5 briefly describes the CaPER components yet to be implemented and other future work. Section 6 describes some other systems that make use of massive parallelism for memory retrieval and compares those approaches with CaPER's.

1.1 Case Retrieval in "Traditional" Case-based Planning Systems

Most case-based systems use serial procedures to retrieve a set of cases from memory that are analogous to the target problem — e.g. the CHEF system (Hammond 1990a, 1990b). Features of the target problem are used as a probe to match against stored cases. Features are objects, object attributes, and relations between objects in the description of the target problem's initial situation and goals.

In order to provide more efficient case retrieval, the technique of indexing is used to restrict the kinds of features that may be used as part of a memory probe. It would be prohibitively expensive to sequentially compare a probe against each case in an entire casebase of realistic size (i.e., hundreds or thousands of cases). Indexing makes this approach more tractable by restricting the search for old cases to only parts of a casebase. When indexing is used, a stored plan can only be retrieved through a subset of its features or some abstraction of these features. For example, CHEF stores a plan for cooking beef and broccoli stir-fry under the indices "beef" and "broccoli" and abstractions "meat" and "vegetable". The abstractions used need to be limited to a few for efficiency's sake.

Choosing an indexing scheme that can provide efficient retrieval of relevant cases from memory is a difficult task. Indexing schemes are often domain-specific and task-specific and thus limit the general-purpose utility of the memory. As an alternative to the explicit indexing of cases, a domain theory might be used to limit the search of case memory. The use of a highly indexed memory or a domain theory limits the flexibility of memory retrieval.

Using a highly indexed memory or a strong domain theory sharply constrains case retrieval. A case can only be retrieved through its indices. Cases that share unindexed features with the target problem will not be retrieved. For example, one might have limited time to devote to cooking and thus desire a cooking plan for cooking any dish that takes less than 15 minutes to prepare. CHEF could not cope with such a query since it does not index plans by the time they take. It therefore would have to resort to checking potentially all cases in its casebase. One could add an index for time to each case in CHEF's casebase. But to do this for many features would lead to a proliferation of indices which could defeat the purpose of indexing, namely efficiency. As another example, one might be doing the cooking in a friend's kitchen that lacks a wok. Thus the query might be to find a cooking plan that does not require a wok. It is unlikely that cases would be indexed under each item that they do not use.

Indexing schemes typically impose an organization on memory. For example, a discrimination network is often used to organize cases in the casebase. Cases with similar indices are stored in the same subtree of a discrimination tree. A new case's indices typically must be computed when the case is stored in the casebase since a case needs to be placed under the right node of the discrimination network. Thus the designer of the indexing scheme needs to anticipate all the ways a case might usefully be retrieved in the future.

The computation of indexes can be expensive. One has to determine the indices of a new case at case storage time and the indices of the target problem that comprise the memory probe at case retrieval time. Finally, the psychological plausibility of indexing is questionable given that human memory access is parallel and associative with many remindings being generated: e.g., (Waltz 1989; Thagard and Holyoak 1989; Gentner 1989).

1.2 Overview of CaPER

CaPER uses a planning cycle similar to that of most CBP systems: (1) the Plan Retriever component retrieves, via the Episode Retriever component, plans from memory used to solve analogous planning problems; (2) the Plan Adapter/Modifier adapts these plans to produce a new plan for the target problem; (3) the Plan Tester presents the new plan to a user who supplies feedback from its execution; (4) the Plan Failure Diagnoser attempts to find the cause of any faults found in the plan and calls the Plan Adapter/Modifier to fix the plan and the Plan Tester to test the repaired plan; (5) the Episode Storer stores a successful plan is stored (along with execution feedback) as a new case in the casebase.

The Memory, Plan Retriever, and Episode Retriever components have been implemented. The Plan Adapter and Episode Storer components are currently under development. The remaining components are in the design phase.

2. CaPER's MEMORY

CaPER's memory includes both episodic and conceptual knowledge. Episodic knowledge describes particular, dated experiences that the system has had (or has been told of). Every episode is associated with a particular date, time, place, and other aspects of the situation in which it occurred. CaPER can apply planning episodes (cases), failure episodes, and repair episodes from particular situations to analogous situations it encounters in the future. A case is a kind of episode that describes a planning experience. This description includes the planning problem (initial situation and target goals), the plan(s) generated, the plan(s) actually instantiated during plan execution, and feedback on the success of the executed plan(s).

A plan that has been instantiated in a particular situation is termed an "e-plan" in CaPER and is also an episode. Plans consists of a hierarchy of component plans ("subplans"), each solving a goal/subgoal. For example, plan Build-Car-1 (used in Case-Build-Car-1) consists of subplans Build-Body-1, Install-Engine-1, Install-Sunroof-1, etc. These subplans can have component plans of their own. At the bottom of a plan hierarchy are primitive actions. These actions that have been instantiated as part of an e-plan are episodes themselves and are termed "e-actions". For example, e-action P-Install-Sunroof-1 is a primitive action of subplan Install-Sunroof-1 (and its parent plan Build-Car-1). A top-level plan, a plan that is not a component of another plan, is termed a "root" plan (e.g., Build-Car-1).

In addition to episodes, CaPER's memory includes conceptual knowledge: representations of concepts. Concepts are organized in generalization taxonomies, part/whole taxonomies, and in other relationships. Concepts include general concepts (e.g., taxonomies of physical things and abstract things), planning concepts (e.g., taxonomies of action conditions and effects), and domain concepts (e.g., a taxonomy relating car types and car parts and a taxonomy of types of car assembly actions).

Individuals are instances of concepts that participate in situations (real or hypothetical) and episodes. For example, car-1 is an instance a class whose members are denoted by the concept Car. Car-1 is a particular car with particular properties that here was the product of a planning episode (Case-Build-Car-1), the result of executing some plan (e-plan Build-Car-1). (Note that a numerical suffix is used to distinguish names of episodes and individuals from names of concepts).

CaPER's memory holds episodic and conceptual knowledge in a semantic network. Each node corresponds to an episode, concept, or individual. Nodes are linked by arcs that represent is-a, part/whole, and other kinds of relations. A piece of CaPER's memory is shown in figure 1. This includes a case (Case-Build-Car-1) of planning to build a car with a sedan frame, 4 cylinder engine, hard top, A/C, and sunroof. A plan (e-plan Build-Car-1) was used to build a car of this type in the Detroit factory, on May 6, etc. With inheritance mechanisms, plans can inherit goals and other properties from their parent plan or case. For example, plan Build-Car-1 (and all of its subplans) inherits the context feature "place Detroit-Factory" from its parent case, Case-Build-Car-1.

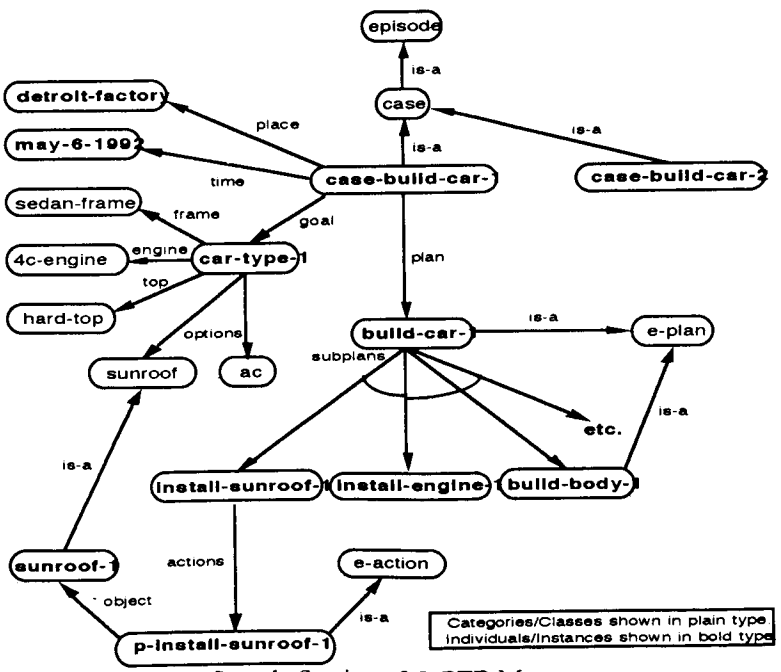


Figure 1. Sample Section of CaPER Memory

2.1 PARKA

CaPER's case memory is built on top of the PARKA knowledge representation system (Spector et al. 1990; Evett et al. 1992). PARKA is a frame-based system implemented on the CM-2 SIMD parallel supercomputer. It provides a rich representation language, consisting of concept descriptions and relations on those descriptions. In addition to representing concepts as a collection of relations with multiple inheritance, PARKA has mechanisms for easily describing set-theoretic and partonomic relations among concepts. PARKA is optimized for the performance of fast "recognition" queries of the form "all X's with features Y, Z, W..." in time proportional to the depth of the knowledge base (KB) — specifically O(D+m), where D is the depth of the KB and m is the number of conjuncts in the query. Typical queries of this type are answered in a fraction of a second (order 10 msec.), even for KBs in the tens of thousands of frames. Since the retrieval times are sensitive only to the depth of the KB, the retrieval algorithms are largely insensitive to KB size (Evett et al. 1992).

Concepts and individuals in CaPER's memory are represented using PARKA category frames and individual frames, respectively (see (Spector et al. 1992)). To extend the use of PARKA for work with CaPER, a "structure matcher" has been implemented, based on the conjunctive retrieval algorithm. This facility allows cases to be retrieved on arbitrary features, not restricted to any set of predetermined indexes. Retrieval probes are specified as a constraint graph which is matched against the entire KB simultaneously. A retrieval probe consists of a graph (V,R), where V is a set of variables, quantified over concepts in the KB, and R is a set of relations which must hold simultaneously between elements of V. The algorithm returns all such satisfying assignments. Preliminary experiments with the initial implementation of the structure matcher have demonstrated retrieval times under a second for a complex probe against a case base with hundreds of cases. Future implementations are expected to reduce retrieval times to under 100 msec. for comparable probes.

3. THE CaPER PLAN RETRIEVER

The design of the Plan Retriever exploits CaPER's ability to do fast, and hence frequent, memory accesses. Here the ramifications of this ability are twofold. First, CaPER can retrieve cases from memory that share any features in common with target problem. Second, CaPER can efficiently do multiple plan retrievals to get different plans (or pieces of plans), each of which can be used to solve a different part of the target problem.

With indexed memories, a case can only be retrieved through its indices, which are typically fixed at case storage time. Since it uses parallel procedures for retrieving episodes, CaPER does not need to use indexing, which serial retrieval procedures require for efficiency. CaPER's memory is therefore unindexed, and episodes can be retrieved on potentially any combination of their features. In CaPER, episodes are interconnected semantic networks of their constituent concepts. Thus it possible to access an episode through any of these constituent concepts. Furthermore, it is possible to issue highly specific queries of CaPER's memory since any feature can be a part of the memory probe.

The use of highly specific queries is facilitated by the ability to do many such queries cheaply. When looking for an episode analogous to a target, CaPER can issue highly specific queries and, if they fail, more general ones. If a more specific query is successful, fewer episodes will typically be retrieved, and there will be less work (in serial) to chose the best from among them (i.e., the ones most similar to the target). A more general query will typically result in more candidate episodes being retrieved and more work being required to evaluate them.

Another ramification of doing frequent memory accesses is being able to retrieve multiple different plans from memory, each of which solves a part of the target problem. CBP systems that use serial procedures typically go to memory infrequently because of the high cost of doing so. Thus they retrieve only one old case/plan (or in some systems, a few old cases) to adapt to solve all of the goals of the target problem. In contrast, CaPER can quickly retrieve several plans (or subplans) to achieve different goals of the target problem and merge them with the result being a better-fitting, composite plan that solves all (or most) of the target goals.

Massive parallelism also makes PARKA's retrieval algorithms run in time virtually independent of the size of the memory (see section 2.1). This means that the techniques used by CaPER/PARKA can scale up to very large memories with thousands of cases or more. In contrast, CBP systems that use serial procedures can typically provide reasonable performance on a casebase with at most tens of cases, unless highly restrictive indexing schemes (or domain theories) are used.

3.1 The Plan Retrieval Process

The CaPER Plan Retriever attempts to retrieve one or more e-plans from memory that solve one or more of the target goals (or goals similar to the target goals). The Plan Retriever also looks for old plans that had initial situations similar to the initial situation of the target problem. The old plans retrieved from memory may be whole plans or plans that are components of other plans. The hierarchical organization of plans in CaPER's memory enables component plans to be retrieved out of larger plans.

The Plan Retrieval Process is controlled by the Plan Retriever and contains these (high-level) steps:

1. Attempt to retrieve an old ("source") plan belonging to a case in memory that is most similar to the entire target problem (the target goals plus the initial situation). This plan is termed the "root source plan". If this step fails, CaPER cannot proceed since it must find at least one old plan to be the basis for the new plan. A generative planner could then be invoked in this event.
2. For each target goal/subgoal that the root source plan does *not* achieve, attempt to retrieve a source plan/subplan that is most similar to the target subproblem (the target goal plus the initial situation). If this step fails for a target goal, CaPER could use a generative planner to create a plan to achieve the goal.
3. For the root and other source plans retrieved, invoke the Plan Adapter to adapt each plan to the target problem and merge the adapted plans.

An example of this process in operation is given in Section 3.1.1.

In Steps 1 and 2, an attempt is made to find plans to build a similar car in a similar situation. Plans are retrieved by probing memory via the Episode Retriever which in turns calls PARKA's Structure Retriever. Memory is probed using the most important ("key") features of the target problem with respect to the domain (since memory is unindexed, any feature can potentially be a key feature). Key features typically include the target goals and relevant features from the initial situation of the target problem. For the simplified automotive assembly domain, the types of key features in order of importance are: type of frame (e.g., "sedan frame"), type of top (e.g., "hard top"), type of engine (e.g., "8 cylinder turbo engine"), type of options (e.g., "sunroof"), and certain features of the initial situation (e.g., "Detroit factory"). A feature is considered more important if it has a greater impact in generating a plan.

In the event that no plans are retrieved with the probe, the features it contains are generalized in order of increasing importance using taxonomic information from memory, and memory is probed again. If a probe succeeds, the cases (or parts of cases) that were retrieved with it are ranked using a similarity metric. A similarity metric is a measure of the goodness of the analogy between an old case and the target problem/case.

CaPER's similarity metric is based on how many key features the cases share (at some level of generality). For each feature of a retrieved case that matches, at some level of generality, a key feature of the target problem, the old case receives 1 point × 1/match-generality. An exact match of features (e.g., "8 cyl. turbo engine" with "8 cyl. turbo engine") is a match generality of 1. For each level difference between the features in the generalization hierarchy, the match generality increases by 1. For example, "8 cyl. turbo engine" with "8 cyl. engine" is a match generality of 2 (since the latter is a parent — one level above — the former) and a score of $1 \times 1/2 = 0.5$.

The effect of using match generality is to give more weight to more specific matches. The advantage of using a simple similarity metric, which CaPER currently computes serially for each case returned, is its ease of computation. Whether this simple similarity metric results in the best ranking of cases is still an open question to be resolved by experimentation.

3.1.1 An Example of the Plan Retrieval Process

To test the CaPER prototype, we have developed a simplified automobile assembly domain. CaPER's initial casebase consists of plans for building cars that were generated using our implementation of Tate's NONLIN planner.

In this example, the target problem has a goal to build a car with a sedan frame, hard top, 8 cylinder turbo engine, A/C, and a sunroof. (It is assumed that the car is being built by hand/robot rather than using an assembly line so that the order of steps may vary from car to car). The target problem also has an initial situation: the car is to be built at the Detroit factory, on a particular date, etc.

In Step 1, memory is first queried using the probe: <sedan frame, hard top, 8 cyl. turbo engine>. This probe fails, and the feature "8 cyl. turbo engine" is be generalized to "8 cyl. engine" (using the knowledge from the concept taxonomy that an 8 cyl. turbo engine "is-a" 8 cyl. engine). The probe <sedan frame, hard top, 8 cyl.

engine> fails, and "8 cyl. engine" is generalized (maximally) to "engine". If the probe <sedan frame, hard top, engine> fails then the next least important key feature, "hard top", would be generalized (to "top"). Fortunately, this probe does not fail and the following old cases are retrieved:

1. **Case: Case-Build-Car-1; E-Plan: Build-Car-1;**
   Car: Car-1 (sedan frame, hard top, 4 cyl. engine, A/C, sunroof, luggage rack); Location: Detroit-factory; Plan Status: Successfully executed; Similarity Score: 5 1/3.
2. **Case: Case-Build-Car-4; E-Plan: Build-Car-4;**
   Car: Car-1 (sedan frame, hard top, 4 cyl. engine, A/C, sunroof, luggage rack); Location: Seattle-factory; Plan Status: Successfully executed; Similarity Score: 4 1/3.
3. **Case: Case-Build-Car-8; E-Plan: Build-Car-8;**
   Car: Car-1 (sedan frame, hard top, 4 cyl. engine, A/C, luggage rack); Location: Seattle-factory; Plan Status: Successfully executed; Similarity Score: 3 1/3.

The similarity score for each case/plan retrieved is computed. While plans Build-Car-1 and Build-Car-4 both achieve the same goals, they differ as to location where the car was built. Since the target car is to be built in Detroit, plan Build-Car-1 will be preferred. The location might be important in choosing between the plans since the available machines, tools, and expertise differ between locations. Plan Build-Car-1 is selected as the root source plan. It contains the following primitive actions:

1. p-build-hard-top
2. p-install-sunroof
3. p-install-ac-compressor-4c
4. p-install-4c-pistons
5. p-build-4c-block
6. p-assemble-4c-engine
7. p-build-sedan-frame
8. p-build-body
9. p-paint-exterior
10. p-install-luggage-rack
11. p-install-engine
12. p-connect-ac
13. p-install-wiring

While this plan builds a car with a sedan frame, hard top, A/C, and sunroof, the car has a 4 cyl. engine instead of the target 8 cyl. turbo engine. In Step 2, the Plan Retriever probes memory with the goal 4 cyl. engine (plus features of the initial situation — e.g., location Detroit, etc.). As in Step 1, the probe would be generalized on failure. Here the probe succeeds, and the plans retrieved are ranked using the similarity metric. Here the highest ranking plan retrieved is plan Build-Engine-2 (a subplan of plan Build-Car-2 in Case-Build-Car-2), a plan for building an 8 cylinder fuel-injected engine with primitive actions:

1. p-build-8c-block
2. p-install-8c-pistons
3. p-install-turbo
4. p-assemble-8c-engine

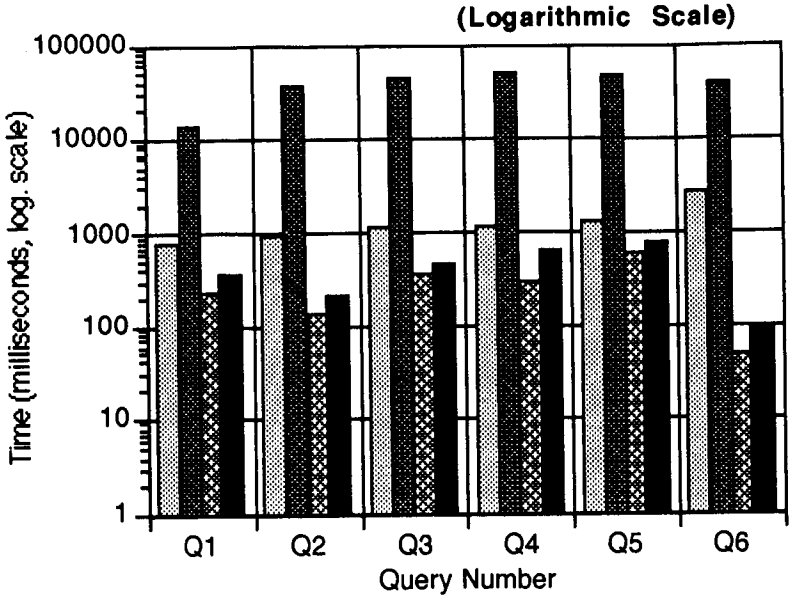
Using the concept hierarchy, CaPER judges the 8 cyl. fuel injected engine built by this plan to be closer to than the target goal of an 8 cyl. turbo engine than to a 4 cyl. engine, built by plan Build-Engine-1, a component plan of root source plan Build-Car-1. Build-Engine-1 is thus replaced by Build-Engine-2, which is predicted to require less adaptation to achieve the target goal.

For this sample target problem, two plans have been retrieved for adaptation. Plan Build-Car-1 built a car with a sedan frame, hard top, A/C, and sunroof. Plan Build-Engine-2 built an 8 cyl. fuel injected engine. The Plan Adapter will adapt these plans to the target situation and combine the new plans into a single plan for achieving all of the target goals.

4. EMPIRICAL RESULTS

To test the efficiency of CaPER's retrieval methods, several queries were issued via the PARKA Structure Retriever (described in section 2.1), which CaPER uses to probe its memory to retrieve cases and other episodes. The queries were done using both the serial and parallel versions for two casebases of 10 cases (1K PARKA frames) and 100 cases (8K frames). Each case was a plan for building a particular type of car that was generated by NONLIN for the simplified automotive assembly domain. The serial version of PARKA was run on a Macintosh IIcx. The parallel version was run on a CM-2 with 8K processors (16K virtual processors). The results are shown in figure 2.The queries, selected to be representative of those CaPER's Plan Retriever issues, were:

1. Find all plans for building a car in Detroit
2. Find all plans for building a car with a 4 cyl. engine
3. Find all plans for building a car with a coupe frame, convertible top, and a 4 cyl. engine
4. Find all plans for building a car with A/C
5. same as in (3) but also with a sunroof and A/C
6. find all plans for building any component in Detroit.



	Serial 10	Serial 100	Paral. 10	Paral. 100
1	772	13444	224	362
2	961	35389	133	217
3	1145	45155	367	469
4	1189	48223	297	637
5	1294	45278	603	758
6	2689	37522	47	92

Figure 2. Queries of CaPER Casebases Using PARKA Structure Retriever

These results show that serial retrieval methods clearly do not scale well to larger casebases. As can be seen, the runtime performance of the parallel algorithms is no worse than logarithmic in the size of the memory. It is expected that an optimized implementation of the structure retriever's parallel algorithms will produce a significant linear decrease in the retrieval times. The parallel system's runtime performance follows from the fact that retrieval times for PARKA queries specified without the structure retriever are dependent only on the depth of the memory network, not its size (Evett et al. 1992).

5. FUTURE WORK

The Plan Adapter component will adapt the source plan(s) found by the Plan Retriever to the target problem. We believe that less adaptation will be required than with traditional CBP systems since more specific plans can be retrieved. The adapted plans will then be merged into a new (composite) plan. Interactions between these subplans of the new plan need to be detected — either at adaptation time (possibly using mechanisms from generative planners) or during plan testing.

The Plan Failure Diagnoser and the plan repair mechanisms, making use of case-based techniques wherever possible, will repair faults (such as harmful interactions between subplans) in a new plan found by the Plan Tester. CaPER's memory will contain past failure episodes to be used to explain similar failures found in the current situation and past repair episodes to be used to fix similar problems.

Another goal is to test CaPER in more realistic domains. One characteristic of the simplistic automobile assembly domain is the modularity of plans to solve problems in it. Our car plans are modular since they assume the modular construction of cars: i.e., the subplan for building the engine is somewhat independent of the subplan for building the frame. This modularity allows subplans from different car plans to be more easily combined. The Plan Adapter has to check for fewer interactions between them, etc. It is therefore desirable to see how CaPER performs on "less modular" kinds of problems

One domain under consideration for implementation is a transport logistics domain involving deliveries of things by plane, truck, etc. This domain has problems that are less modular than those of the car assembly domain. Case retrieval in this domain would probably be highly sensitive to the initial situation of target problems (e.g., what

trucks are where, which cities have airports, etc.). The space of target problems is also very large. In contrast, problems in the car assembly domain have highly similar initial situations and a narrower range of goals.

6. RELATED WORK

Closely related to work done in CBR is work done in memory-based reasoning (MBR). MBRTalk is a MBR system for speech pronunciation that uses massive parallelism, as CaPER does, to access a large, unindexed memory of cases residing in the CM (Stanfill and Waltz 1986). MBRTalk retrieves cases that are the shortest distance from a probe. Each case, in parallel, computes its distance from the probe using a simple similarity metric. CaPER, in contrast, computes similarity scores for candidate cases in serial (due to cases being distributed across CM processors). MBRTalk uses a flat representation of cases. CaPER uses a highly structured, semantic network representation of cases. CaPER also uses a stronger domain model than MBRTalk.

PARADYME (Kolodner and Thau 1988) is an implementation of a memory for a CBR system that has much in common with parts of CaPER/PARKA. PARADYME uses the massive parallelism of the CM to access an unindexed memory. Memory holds concepts as well as cases. Cases are represented using a semantic net/frame-based representation. Cases are annotated with critical features (similar to CaPER's key features) to be used for selecting the best matching cases. PARADYME used different retrieval procedures than CaPER and was not yet integrated with a CBR system. Additionally, PARKA's retrieval algorithms are significantly more complete and faster than PARADYME's (Evett et al. 1992).

7. CONCLUSION

CBP systems can take advantage of plan re-use where possible. The success of this approach depends on the ability to retrieve old cases that are similar to the target problem and to adapt these cases appropriately. Using its unindexed memory and massively parallel retrieval mechanisms, CaPER can do fast, frequent retrievals using any features of the target problem. The result of this is that CaPER can retrieve cases that better match the target problem and thus require less adaptation. Empirical results indicate that the often inflexible, special-purpose indexing schemes required for the serial retrieval methods of traditional CBP systems using indexed memories can be abandoned in favor of parallel methods. These parallel methods can scale up to casebases of hundreds or thousands of cases. Larger casebases can improve the plans a CBP system produces by providing more cases to drawn upon.

REFERENCES

Gentner, D., "The mechanisms of analogical learning," In *Similarity and Analogical Reasoning*, Eds. S. Vosniadou and A. Ortony, Cambridge: Cambridge University Press, 1989, pp. 199-241.

Evett, M.P., Hendler, J.A., and Spector, L., "Parallel Knowledge Representation on the Connection Machine," *Journal of Parallel and Distributed Computing*, 1992 (forthcoming).

Hammond, K.J. (1990a), "Case-based planning: A framework for planning from experience," *Cognitive Science*, Vol. 14 (1990), pp. 384-443.

Hammond, K.J., (1990b), "Explaining and Repairing Plans That Fail," In *Artificial Intelligence*, Vol. 45 (1990), 173-228.

Hillis, W.D., *The Connection Machine*, Cambridge, Massachusetts: The MIT Press, 1985.

Kolodner, J.L., and Thau, R., *Design and Implementation of a Case Memory*, Technical Report RL88-1, Thinking Machines Corporation, Cambridge, Massachusetts, 1988.

Ross, B.H., "Some Psychological Results on Case-based Reasoning," In *Proceedings: Case-Based Reasoning Workshop (DARPA)*, San Mateo, California: Morgan Kaufmann, 1989, pp. 144-147.

Spector, L., Hendler, J.A., and Evett, M.P., *Knowledge Representation in PARKA*, Technical Report 2410, Department of Computer Science, University of Maryland at College Park, 1990.

Spector, L., Anderson, B., Hendler, J., Kettler, B., Swartzman, E., Woods, C., and Evett, M., *Knowledge Representation in PARKA - Part 2: Experiments, Analysis, and Enhancements*, Technical Report 2837, Department of Computer Science, University of Maryland at College Park, 1992.

Stanfill, C. and Waltz, D., "Toward Memory-Based Reasoning," *Communications of the ACM*, Vol. 29, No. 12, December 1986, pp. 1213-1228.

Tate, A., *Project Planning Using a Hierarchic Non-linear Planner*, Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh.

Thagard, P.R. and Holyoak, K.J., "Why Indexing is the Wrong Way to Think About Analog Retrieval", In *Proceedings: Case-Based Reasoning Workshop (DARPA)*, San Mateo, California: Morgan Kaufmann, 1989, pp. 36-40.

Vosniadou, S. and Ortony, A., "Similarity and analogical reasoning: a synthesis," In *Similarity and Analogical Reasoning*, Cambridge: Cambridge University Press, 1989, pp. 1-17.

Waltz, D., "Is Indexing Used for Retrieval?" In *Proceedings: Case-Based Reasoning Workshop (DARPA)*, San Mateo, California: Morgan Kaufmann, 1989, pp. 41-45.