Stratified Case-Based Reasoning: Reusing Hierarchical Problem Solving Episodes

L. Karl Branting
Department of Computer Science
University of Wyoming
P.O. Box 3682
Laramie, WY 82071
karl@eolus.uwyo.edu
(307) 766-4258 / FAX: -4036

David W. Aha
Navy Center for Applied Research in AI
Code 5510
Naval Research Laboratory
Washington, DC 20375
aha@aic.nrl.navy.mil
(202) 767-9006 / FAX: -3172

Abstract

Stratified case-based reasoning is a technique in which abstract solutions produced during hierarchical problem solving are used to assist case-based retrieval, matching, and adaptation. We describe the motivation for the integration of case-based reasoning with hierarchical problem solving, exemplify its benefits, detail a set of algorithms that implement our approach, and present their comparative empirical evaluation on a path planning task. Our results show that stratified case-based reasoning significantly decreases the computational expense required to retrieve, match, and adapt cases, leading to performance superior both to simple case-based reasoning and to hierarchical problem solving *ab initio.*

1 Problem and Proposed Solution

Case-based reasoning (CBR) is a problem-solving paradigm that focuses on the retrieval, revision, and reuse of stored solutions to solve newly presented problems (Kolodner, 1993; Aamodt & Plaza, 1994). The representation of cases can be simple (e.g., feature vectors) or complex (e.g., semantic networks). An advantage of simple representations is that retrieval and revision algorithms for such representations have comparatively low computational cost. However, this low computational cost comes at the expense of limited representational power. For example, the relational knowledge required for such tasks as planning, design, and analogical reasoning is difficult to express in a feature-vector case representation. However, more expressive case representations require more expensive retrieval and revision functions. For example, when cases are represented as semantic networks, case matching requires subgraph isomorphism determination, an NP-complete task (Garey *k* Johnson, 1979).

In this paper, we propose using abstraction to combat the complexity associated with expressive case representations. We explain how abstraction can be used to reduce this complexity in Section 2 and describe an example task in Section 3. Section 4 then introduces a set of algorithms that implement our approach. We evaluate their capabilities in comparison with simpler case-based and abstraction approaches in Section 5 and discuss related work in Section 6.

2 Abstraction: Reducing CBR Complexity

Two principal approaches exist for reducing the complexity of indexing and matching complex cases on sequential machines.[1] One approach is to precompute some portion of the match between cases (e.g., by organizing cases into a hierarchy partially ordered by subgraph relations (Levinson, 1991)). An alternative approach involves using case abstractions for indexing and matching. Previous use of abstraction in indexing has generally focused on thematic abstractions, such as goals (Hammond, 1989) or expectation failures (Schank & Leake, 1989). Use of abstraction in case matching has generally been limited to feature generalization (e.g., finding common ancestors of new-case and old-case features in a hierarchy that encodes subsumption relations on features values (Porter et al., 1990)).

This paper advocates a new approach for using abstraction in CBR. Underlying this approach is the observation that hierarchical problem solving[2] gives rise to solutions at multiple levels of abstraction. If the solutions produced by hierarchical problem solving at every level of abstraction are retained, then the more abstract solutions can assist in indexing, matching, and adapting less abstract solutions. We refer to a case whose representation has multiple abstractions as a *stratified case,* and we define *stratified CBR* as the use of stratified cases in case-based problem solving. The potential benefits of this approach are:

- Indexing and retrieval. A more abstract solution to a problem can provide an accurate index to less abstract solutions to the problem because it consists

[1]See (Kolodner, 1988) and (Kettler *k* Hendler, 1994) for related work on machines with parallel architectures.

[2] In hierarchical problem solving, search is performed in an abstraction of the original search space. The solution in the abstract space is then used to guide problem solving in the original space. Where applicable, this technique can lead to significant reductions in search time (Sacerdoti, 1974; Holte et al., 1995; Knoblock, 1994; Bacchus *k* Yang, 1994).

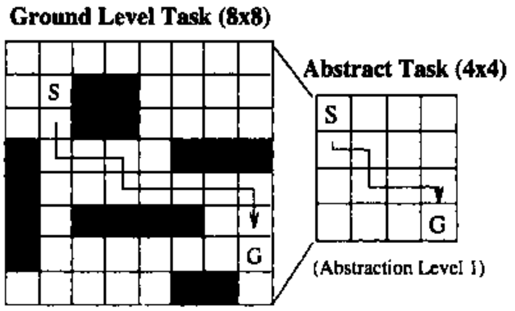

Figure 1: Two Abstraction Levels and Example Ground-Level & Abstract Solution Paths for a Route-Finding Task

of the most important aspects of the less abstract solutions.

- Matching. Retaining case abstractions permits cases to be compared in an abstract space in which matching may be much less expensive than at the ground level of abstraction.
- Adaptation. An abstraction of a stored case may be much easier to reuse than the ground-level case itself. Stratified cases can be reused at the most specific level of abstraction at which they can be applied to the given problem without requiring adaptation of less abstract, non-matching facts.

Since stratified CBR involves reusing hierarchical problem solving episodes, its applicability is limited to domains in which there is an abstraction hierarchy suitable for hierarchical problem solving. However, several techniques that automatically derive abstraction hierarchies have recently been developed, and the development of new techniques is the focus of an active research community (Knoblock, 1994; Christensen, 1990; Bacchus & Yang, 1994).

3 The Route-Finding Task

We chose a route-finding task to demonstrate the utility of stratified case-based reasoning because this task is an important area of activity in robotics and is amenable to hierarchical problem solving. Our task-involves finding an optimal or near-optimal path between a given pair of start and goal positions through a field containing obstacles. We chose to work with fields described by *N* x *N* arrays of positions, where N is a power of 2. This allowed us to define a simple abstraction hierarchy in which an abstract position at Level 1 position *(R, C)* (zero indexing) abstracts over the following four ground-level positions: $\langle 2\times R, 2\times C\rangle$, $\langle 2\times R, 2\times C+1\rangle$, $\langle 2\times R+1, 2\times C\rangle$, $\langle 2\times R+1, 2\times C+1\rangle$.

Figure 1 shows a situation in which *S* and *G* denote the start and goal positions. The figure on the left is a ground level field containing 64 positions, where the grey positions denote obstacles, while the field on the right is its Level 1 abstraction.

The goal of this task is to locate a route connecting the start and goal positions. We modeled this as the task of constructing a sequence of straight and curved track segments, or *traversal operators,* such that the start and goal positions lie at the ends of the connected track. We associate with each position the set of operators that can be used to traverse it. Thus, each unblocked position at the ground level is associated with all possible operators, which are shown in Figure 2. Each blocked position is associated with the empty set of traversal operators since traversal through them is impossible.

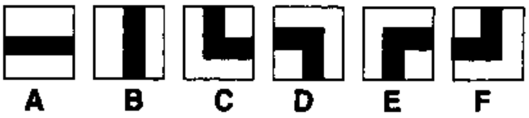

Figure 2: The Set of Possible Traversal Operators Per Position

Determining the available operator set for each abstract position involves determining (1) what operators are available for each of the four positions it abstracts and (2) which of the six operators, if any, are still possible after joining these four lower-level positions. For example, while all six traversal operators are available for the top-left abstract position, the only operators available for the position to its right are *{A,C, F}* (i.e., the others are unavailable because no traversal through this abstract position can reach the position below it).

We restricted our consideration to fields whose obstacle configurations guarantee that hierarchies created using our abstraction operators satisfy the *downward refinement property* (Bacchus & Yang, 1994) (i.e., every abstract solution can be refined to a concrete solution, if a concrete solution exists).[3] Thus, we considered only fields for which hierarchical problem solving reduces search.

The stratified CBR approach to this problem is to retain and reuse the solution paths found at *every level of abstraction* in hierarchical search. Retaining abstract solutions decreases the cost of finding the most similar stored solution path because the distance from the new start and goal to these paths can be accurately measured at the highest levels of abstraction, where search is much less expensive (i.e., since the fields are smaller in size).

4 Algorithms

This section introduces two non-CBR algorithms and five CBR algorithms (Table 1) that can be applied to the route-finding task. Each of the CBR algorithms re-

[3] The downward refinement property is satisfied for our abstraction operators if (1) all adjacent obstacles are arranged in rectilinear regions and (2) whenever the length of a sequence of adjacent obstacles is at least the size of an abstract region at some level of abstraction, then its width is at least the region size minus 1.

Table 1: Summary of the Five CBR Algorithms

Algorithm	Hier-archical	Type of Matching		
		Perfect	Partial	Thresh-olded
GROUND COVER		√		
GROUND CLOSEST			√	
COVER	√	√		
CLOSEST	√		√	
CLOSEST THRESHOLD	√		√	√

trieves and reuses previously stored solutions while problem solving. They can be distinguished by (1) whether they access cases stored at only the ground level or at all abstraction levels, and (2) whether they support perfect or partial matching.

4.1 Non-CBR Algorithms

We considered two non-CBR algorithms for the route-finding task. The first is best-first search (A*) using manhattan distance as the heuristic estimation of distance from the current position to the goal position. The second is *hierarchical* A*, which uses A* to find a path from the start position to the goal position in the highest level abstraction of the given field. At each lower level of abstraction, search by A* is restricted to positions falling within the solution path at the next higher level of abstraction.

The cases used by the CBR algorithms described below are generated by hierarchical A*. Given an abstraction hierarchy for a particular field and start and goal positions, hierarchical A* generates a path connecting the start and goal positions at every level of abstraction. Each solution at a given level of abstraction is treated as a separate case. A refinement of a case (i.e., a solution at the next lower level of abstraction) is termed a *child* of the case, and a case is termed the *parent* of its refinements. Since distinct positions at the ground level may be identical at more abstract levels, distinct cases at a lower level may have identical parents. Cases can therefore be organized into a forest of taxonomic trees. A *case library* consists of a taxonomic forest of cases sharing a common abstraction hierarchy but distinct pairs of start and goal positions.

4.2 Ground level CBR algorithms

The simplest of the CBR algorithms is GROUND COVER (Table 2), which searches the case library for ground solution paths that include the new start and goal positions. If any are found, then it returns a randomly-selected shortest solution path. Otherwise, it invokes A* to find a solution.

The second ground level CBR algorithm is GROUND CLOSEST (Table 3), which also searches the case library for ground solution paths that include the new start and

Table 2: GROUND COVER: Perfect Matching at the Ground Level

```
Given: $S$: start position
       $G$: goal position
       Lib: case library

GROUND-COVER($S$,$G$,Lib) =
  Cases := ground_level_matches($S$,$G$,Lib)
  IF (Cases $\neq \emptyset$)
    THEN random(shortest_solution(Cases))
    ELSE best_first_search($S$,$G$)
```

Table 3: GROUND CLOSEST: Partial Matching at the Ground Level

```
GROUND-CLOSEST($S$,$G$,Lib) =
  Cases := ground_level_matches($S$,$G$,Lib)
  IF (Cases $\neq \emptyset$)
    THEN random(shortest_solution(Cases))
    ELSE
    random(shortest_adapted_solution($S$,$G$,Lib))
```

goal positions. If none are found, then GROUND CLOSEST adapts each ground case (i.e., uses A* to locate the shortest paths joining the new start and goal positions with the stored case's solution) and returns the shortest adapted solution. Thus, this algorithm supports partial matching. To prevent retrieval from increasing in proportion to the size of the case library, GROUND CLOSEST uses a modification of A* that prunes each node *n* for which *f(n)* is greater than the cheapest adaptation found so far. Thus, GROUND CLOSEST performs best-first search for the shortest adaptation paths within each case, but branch-and-bound search through the case library for the case with the shortest adaptation paths.

4.3 Stratified CBR algorithms

The three stratified CBR algorithms can reuse case solutions stored at any abstraction level. Each of these algorithms starts by retrieving from the case library the set of *most specific matching cases* (i.e., lowest-level cases whose solutions include abstract positions that abstract the given start and goal positions). This search begins at the root of the case library, recurses with its children (i.e., top-level abstractions of solved cases), and continues recursing until it reaches the ground level (in which case the segment of a solution connecting the new start and goal positions is returned) or cases that no longer cover both the start and goal positions.

In the latter case, the first algorithm, COVER (Table 4), randomly selects one of the most specific matching cases and performs hierarchical A* starting at the next lower level of abstraction, restricting search to positions falling within the solution path in the most specific matching case.

In contrast, the second algorithm, CLOSEST (Table 5), supports partial matching between new problems and previous solutions. Starting with the most specific

Table 4: COVER: Perfect Hierarchical Matching

```
COVER(S,G,Lib) =
  Cases := most_specific_matches(S,G,Lib)
  Case := random(Cases)
  hierarchical_solve(S,G,Case)
```

Table 5: CLOSEST: Partial Hierarchical Matching

```
CLOSEST(S,G,Lib) =
  Cases := most_specific_matches(S,G,Lib)
  CLOSEST_RECURSE(S,G,Cases)

CLOSEST_RECURSE(S,G,Cases) =
  Refined_cases = refine(Cases)
  Adapted_cases = adapt(S,G,Refined_cases)
  Best_cases = shortest(Adapted_cases)
  IF (ground_level(Best_cases))
   THEN random(Best_cases)
   ELSE CLOSEST_RECURSE(S,G,Best_cases)
```

matching cases (or the most abstract cases, if no cases match) CLOSEST finds the refinements of each case (i.e., the case's children), adapts each refinement (i.e., uses A* to find the shortest adaptation paths from the start and goal positions to the solution path at that level of abstraction, restricting search to positions in the parent case's adaptation paths), and selects the refinements having the shortest adapted solution paths.[4] CLOSEST recursively calls these three steps until the ground level is reached, at which time it randomly selects and returns an adapted case.

The final algorithm, CLOSEST THRESHOLD (Table 6) attempts to recognize situations in which adapting an existing case will be more expensive than problem solving *ab initio.* CLOSEST THRESHOLD behaves identically to CLOSEST if there are matching cases. However, if there are no matching cases, then CLOSEST THRESHOLD uses A* to find the shortest path from the start to the goal position at the highest level of abstraction. If there are top-level cases whose adapted solution paths are no longer than the path length found by A*, then CLOSEST THRESHOLD treats these cases in the same manner as CLOSEST. If there are no such cases, then CLOSEST THRESHOLD uses hierarchical A* rather than CBR.

[4]To prevent the performance of CLOSEST from degrading when there are large numbers of redundant cases (i.e., cases with equally short adapted solutions), CLOSEST performs two additional types of pruning. First, among cases with the shortest adapted solution paths, CLOSEST retains only those for which the sum of the lengths of the paths from the start and goal positions to the previous solution path are least. Second, below the highest level of abstraction, if there are multiple cases for which the paths from the start and goal positions to the previous solution path intersect the previous solution path at identical positions (i.e., multiple cases for which the reusable segments of the old solution paths are indistinguishable), then only one such case is retained.

Table 6: CLOSEST THRESHOLD: Partial Hierarchical Matching with Threshold

```
CLOSEST-THRESHOLD(S,G,Lib) =
  Cases := most_specific_matches(S,G,Lib)
  Best := shortest_solution(S,G)
  IF (length_of_shortest_solution(Cases)
      > length(Best))
   THEN hierarchical_solver(S,G,Best)
   ELSE CLOSEST_RECURSE(S,G,Cases)
```

5 Empirical Evaluation

Previous work has demonstrated the advantages of hierarchical problem solving over non-hierarchical methods(e.g., Holte et al., 1995; Knoblock, 1994; Bacchus & Yang, 1994). We focus here on demonstrating the utility of reusing the abstract solutions generated by a hierarchical problem solver. Therefore, this section describes empirical comparisons of case-based vs. non-case-based approaches for hierarchical problem solving. We also include comparisons with non-hierarchical methods, both case-based and otherwise, to verify that the more storage-intensive hierarchical approaches deliver better performance.

5.1 Hypotheses and experimental variables

The empirical hypotheses explored in this section are:

1. Reusing stored case solutions decreases search costs.
2. Reusing abstract case solutions decreases costs more than reusing only ground-level solutions.
3. Partial matching yields lower search costs than perfect matching.
4. Preventing expensive partial matching (via thresholding) yields the best performance on this task.

We will compare the five algorithms described in Section 4 against A* and hierarchical *A**. The primary dependent variable of interest is work effort as measured by the number of nodes expanded by *A** in the course of executing each algorithm.[5] We will also not discuss the lengths of the algorithms' solutions since they were all very similar.[6] The independent variables are

1. the size of the case base,
2. the number of abstraction levels, and
3. the distribution of selected problems (i.e., start and goal positions).

[5] We ignore the additional computational costs incurred by the hierarchical and case-based algorithms. The one-time cost of building a hierarchy is $0(N \times O)$, where *N* is the number of ground level positions and *O* is the number of traversal operators. This cost is amortized over each set of training and testing problems drawn from a single ground level field and obstacle configuration. The cost of case retrieval is reduced to a constant by hashing cases on their abstraction level, start position, and goal position.

[6]In Experiment 2, their averages varied from 154.8 (A*) to 157.3 (Ground Closest).

Table 7: Ranges and Defaults of Independent Variables

Variable	Range
# Cases	{0, 2, 5, 10, **25**, 50}
# Abstraction levels	{1, 2, **3**, 4}
Problem distribution	{random, opposite sides, **opposite ends**}

We expected that the benefits of case-based reasoning, as measured by decreased search costs, would increase as the library size increases. Similarly, we expected the cost of searching to decrease as the number of abstraction levels increases. Therefore, we expected the utility of stratified CBR to increase with the number of abstraction levels. Finally, we expected problem distribution to affect problem-solving behavior.7 Therefore, we experimented with distributions that vary the average solution path length.

5.2 Experiments

We varied one of the independent variables listed above in each of three experiments. The settings for the other two variables were held constant at default values (see Table 7). All the reported results refer to averages over sets of 100 testing problems. Each set is decomposed into ten subsets often problems. Each subset refers to a different layout of obstacles, where the layouts are constrained as described in Section 3. Each algorithm was applied to the same set of training and test sets.

Varying the number of cases

We varied the number of stored cases while fixing the field size to be 32 x 32 and the number of abstraction levels to three. The problems were selected so that the start and goal positions were on "opposite ends" of the field, meaning that the value of one randomly selected dimension (e.g., row) was chosen randomly and the other (e.g., column) was fixed to maximally distance the two positions.

Figure 3 summarizes the results. Since A* and hierarchical A* do not use stored cases, their results remain constant. The five CBR algorithms all outperformed A*. GROUND COVER'S work effort decreased linearly with library size, indicating that it found almost no stored matching problems for small-sized case bases but occasionally found such matches for larger case bases. GROUND CLOSEST outperformed GROUND COVER for larger case bases, which indicates the benefits of adapting stored solutions when they are sufficient in number and similarity to new problems.

The performance improvement delivered by the three stratified CBR algorithms was dramatic. COVER'S average work effort decreased linearly with library size, indicating the utility of reusing stored solutions at different abstraction levels. This approach significantly outperformed GROUND COVER ($p < 0.025$ for 50 training cases), indicating that the stratified approach is better than the non-stratified CBR approach for this task. Similarly, CLOSEST significantly outperformed GROUND CLOSEST ($p < 0.1$). Finally, CLOSEST significantly outperformed COVER ($p < 0.05$), indicating that partial matching is preferable to perfect matching on this task. However, no significant difference was found (i.e., for 50 training cases) suggesting that the thresholding algorithm outperforms CLOSEST. Thus, these results support our first three hypotheses, but not our fourth.

7 Past cases with short solution paths at the ground level are less likely to contain segments useful for subsequent problems. Similarly, the benefit of case reuse is less likely to outweigh the overhead of retrieval and adaptation in new problems involving start and goal positions that are near to each other, since such problems can be inexpensively solved *ab initio.*

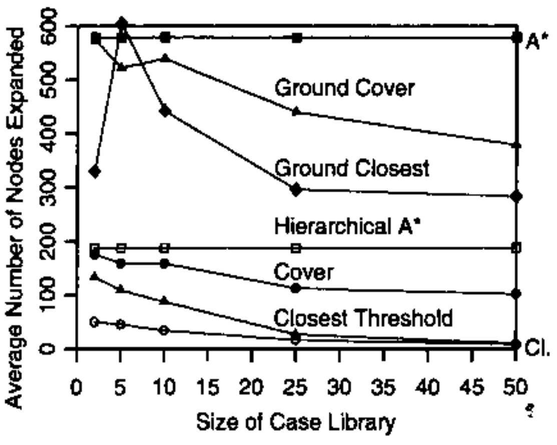

Figure 3: Learning Curves for the Seven Algorithms

Varying the number of abstraction levels

In this experiment, we fixed the size of the ground level field to be 64 x 64 and varied the number of abstraction levels. The results for *A** and the non-stratified CBR algorithms remain constant since they are not effected by stored cases at varying abstraction levels.

In contrast, Figure 4 shows that the search costs of the hierarchical problem solver and the stratified CBR algorithms decrease with the number of abstraction levels. COVER, as expected, performs slightly better than hierarchical A* because it reduces search whenever it finds a perfect (abstract) match and otherwise does not incur additional search costs. The partial matching stratified CBR algorithms perform poorly when only a few abstraction levels are used because adaptation is costly when the positions along a solution path lie far from the start and goal positions. However, they perform significantly better than the other algorithms (i.e., at least at the $p < 0.1$ level) when four abstraction levels are used since adaptation is much less costly in the higher abstraction levels. Although CLOSEST outperformed CLOSEST THRESHOLD, their differences were not significant. These results again support the first three hypotheses described in Section 5.1 and contradict the fourth.

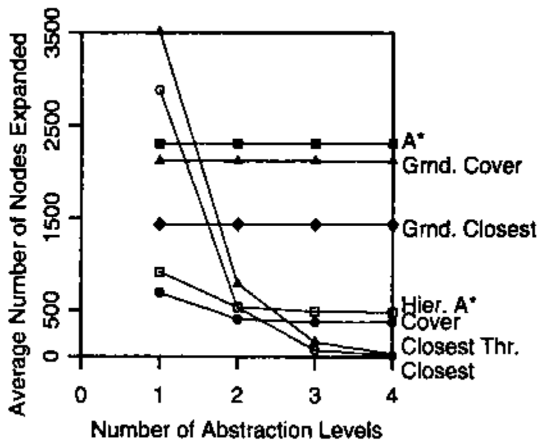

Figure 4: Number of Abstraction Levels vs. Search Time

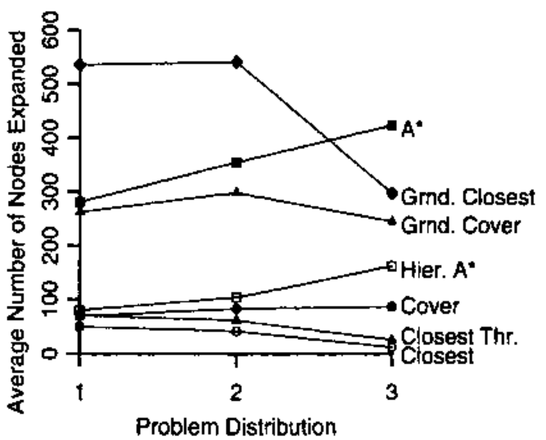

Figure 5: Problem Distribution vs. Search Time

Varying the problem distribution

In Experiment 3, we varied only the way in which problems (i.e., pairs of start and goals positions) were selected. We used the *opposite ends* strategy in the other experiments, which chooses positions that are maximally distant in the space along one axis and randomly selected on the other axis. This is a favorable bias for stratified CBR since it guarantees long solution paths, which increases the probability of reuse. We add two strategies here that decrease this probability: *random.,* which selects positions randomly, and *opposite sides,* which selects positions randomly but guarantees them to be on opposite sides of one of the two dimensions. These three strategies differ in the degree to which they prevent the selection of nearby start and goal positions. *Random* allows selection of such pairs. *Opposite sides* prevents this to some degree, and *opposite ends* prevents this entirely.

Figure 5 summarizes the results for this experiment. As expected, search costs rise for A* and Hierarchical A* as the average minimum solution length increases (i.e., it was 34.2, 42.9, and 61.4 for *random, opposite sides,* and *opposite ends* respectively). This increase also yields abstract solutions that require less search to adapt (i.e., since they cover more of the space), which explains why CBR performance improves as solution length increases. The stratified CBR algorithms again outperformed the ground CBR algorithms, and the stratified partial matching algorithms outperform COVER.

5.3 Summary

The results of our experiments provide initial support for the first three hypotheses stated in Section 5.1. The CBR algorithms outperform the non-CBR algorithms on which they are based (i.e., especially for the *opposite ends* problem distribution), which supports the claim that solution reuse can decrease search costs. The stratified CBR algorithms outperform the ground level CBR algorithms, which supports the claim that the reuse of abstract case solutions further improves performance. The partial matching algorithms (i.e., CLOSEST and GROUND CLOSEST), outperform the perfect-matching CBR algorithms (i.e., COVER and GROUND COVER). Finally, the results of CLOSEST THRESHOLD reflects how it mediates between these two extremes, but it does *not* outperform CLOSEST, contradicting our fourth hypothesis (i.e., that thresholding yields the best results). However, we conjecture that, for case libraries for which the adaptation cost of the best case is sufficiently high, thresholding will be beneficial.

6 Discussion and Related Work

The distinguishing characteristic of stratified case-based reasoning is its reuse of stored solutions at all abstraction levels. Holte et al.'s (1995) *alternating opportunism* algorithm similarly uses hierarchical refinement to reduce search costs. It caches, in each state in the search space, information that simplifies A* search (i.e., g(n) and h(n) values). Their approach is complementary to ours in that it reduces search effort from a state-oriented rather than a solution-path perspective.

CBR adaptation methods are typically either *transformational,* which retrieve solutions and apply adaptation operators to revise them to solve a new problem, or *derivational,* which replay the derivations of problem-solving episodes (Carbonell, 1986). Stratified CBR employs a transformation approach that uses more abstract solutions to index and guide the adaptation of less abstract solutions. Since stratified CBR partitions cases by level of abstraction rather than by stages of goal reduction, it is applicable to tasks that use problem-solving methods other than goal reduction. Moreover, we hypothesize that in tasks that use goal reduction stratified CBR will often order goals by degree of constraint more effectively than derivational analogy, because abstraction is often more indicative of degree of constraint than order of appearance in a goal-reduction graph.

The PRIAR (Kambhampati, 1994) framework for plan modification embodies many stratified CBR characteristics. PRIAR retrieves solutions with minimum predicted adaptation costs and uses causal dependency structures to guide adaptation via a minimum-conflict search strategy. These structures function as partially-ordered abstraction hierarchies for decomposing actions. Retrieved abstract solutions are not guaranteed to be refinable and can be extended during adaptation (i.e., with additional planning). Previous work on PRIAR has focused on its semantics and its applicability rather than on isolating the contribution of resuing abstract solutions. Haigh et al. (1994) integrate derivational analogy with a limited (i.e., two abstraction levels) stratified CBR approach. Their work focused on a route planning task rather than on isolating the contributions of stratified CBR. Hanks and Weld (1995) describe a similar general framework that permits adaptation of abstract plans. However, unlike stratified CBR, this approach does not use more abstract solutions to index less abstract solutions.

Explanation-based learning (EBL) (Mitchell et al., 1986) resembles stratified CBR in that a *macro* derived by EBL algorithms from a proof of concept membership constitutes an abstraction of the proof (i.e., a *generalized example).* However, EBL systems typically allow reuse at only a single level of abstraction for each generalized example, whereas stratified CBR allows reuse at any level of abstraction.

We believe that the stratified CBR approach could be beneficially incorporated into EBL algorithms, into hierarchical reinforcement learning approaches (Kaelbling, 1993), and into other AI approaches involving hierarchical problem solving. We view it as a general contribution to AI research on abstraction and envision its incorporation into many multi-strategy systems.

7 Conclusion

This paper has described *stratified case-based reasoning,* an application of case-based reasoning to hierarchical problem solving. An empirical evaluation in a route-finding task demonstrated that this approach can lead to significantly lower computational costs than either hierarchical problem solving *ab initio* or ground-level CBR. Moreover, the evaluation showed that reusing abstract solutions yields better performance than reusing only ground level solutions, that the performance benefit of stratified CBR increases with the size of the case base and with the number of abstraction levels, and that partial matching outperforms perfect matching.

Acknowledgements

Many thanks to Patrick Harrison for supporting this research and for discussions leading to this paper. Rob Holte's input and feedback were also valuable. This research was supported in part by a Visiting Summer Faculty Research Grant from the Naval Research Laboratory.

References

Aamodt, A., k Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICOM, 7,* 39-59.

Bacchus, F., k Yang, Q. (1994). Downward Refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence, 71,* 43-100.

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, k T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach (Volume II).* Los Altos, CA: Morgan Kaufmann.

Christensen, J. (1990). A hierarchical planner that generates its own hierarchies. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 1004-1009). Boston, MA: AAAI Press.

Garey, M. R., k Johnson, D. S. (1979). *Computers and intractability.* New York, NY: Freeman.

Haigh, K. Z., Shewchuk, J. R., k Veloso, M. (1994). Route planning and learning from execution. In *Working notes form the AAAI Fall Symposium "Planning and Learning: On to Real Applications"* (pp. 58-64). New Orleans, LA: AAAI Press.

Hammond, K. (1989). *Case-Based Planning: Viewing Planing as a Memory Task,* San Diego, California: Academic Press, Inc.

Hanks, S., k Weld, D., (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research, 2,* 319-360.

Holte, R. C, Mkadmi, T., Zimmer, R. M., k MacDonald, A. J. (1995). *Speeding up problem-solving by abstraction: A graph-oriented approach* (Technical Report TR-95-07). Ottawa, Canada: University of Ottawa, Department of Computer Science.

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 167-173). Amherst, MA: Morgan Kaufmann.

Kambhampati, S., (1994). Exploiting causal structure to control retrieval and refitting during plan reuse. *Computational Intelligence, 19,* 212 244.

Kettler, B., k Hendler, J. (1994). Evaluating a case-based planning system. In D. W. Aha (Ed.), *Case-Based Reasoning: Papers from the 1994 Workshop* (Technical Report WS-94-01). Menlo Park, CA: AAAI Press.

Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence, 64-*

Kolodner, J. L. (1988). Retrieving events from a case memory: A parallel implementation. In *Proceedings of the Workshop on Case-Based Reasoning* (pp. 233-249). Clearwater, FL: Morgan Kaufmann.

Kolodner, J. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann.

Levinson, R. (1991). *Pattern associativity and the retrieval of semantic networks* (Technical Report UCSC-CRL-91-14). Department of Computer Science, University of California, Santa Cruz.

Mitchell, T., Keller, R., k Kedar-Cabelli, S. (1986). Explanation-based learning: A unifying view. *Machine Learning, 1,* 47-80.

Porter, B. W., Bareiss, E. R., k Holte, R. C, (1990). Concept learning and heuristic classification in weak-theory domains, *Artificial Intelligence, (45),* 229-263.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5,* 115-136.

Schank, R., k Leake, D. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence, (40),* 353-385.