Stratified Case-Based Reasoning in Non-Refinable Abstraction Hierarchies

L. Karl Branting

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

Abstract. Stratified case-based reasoning (SCBR) is a technique in which case abstractions are used to assist case retrieval, matching, and adaptation. Previous work showed that SCBR can significantly decrease the computational expense required for retrieval, matching, and adaptation in a route-finding domain characterized by abstraction hierarchies with the downward refinement property. This work explores the effectiveness of SCBR in hierarchies without the downward refinement property. In an experimental evaluation using such hierarchies (1) SCBR significantly decreased search cost in hierarchies without the downward refinement property, although the speedup over ground-level A* was not as great as in refinable hierarchies, (2) little difference was observed in SCBR search costs between case libraries created top-down in the process of REFINEMENT and those created bottom-up from a valid ground solution, and (3) the most important factor in determining speedup appeared to be *a priori* likelihood that a previous solution can be usefully applied to a new problem.

1 Stratified Case-Based Reasoning

Stratified Case-Based Reasoning is a technique under which case abstractions are used to assist case indexing, matching, and adaptation. This approach has been applied to case-based planning [BW95, KH92], design of control software [SK94, SC92], and route planning [BA95] (See [BW96] for a comparative analysis of previous approaches). Use of case abstractions has the following potential benefits:

- **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 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.

A systematic analysis set forth in [BA95] compared the performance of heuristic search (A^*), Refinement (i.e., hierarchical problem solving), ground-level CBR, and Scbr as a function of (1) number of levels of abstraction, (2) the size of the case library, and (3) resemblance among cases. The comparison was in the context of a route-finding task restricted to fields for which a simple aggregation abstraction method produced hierarchies satisfying the downward refinement property [Kno94, BY94], i.e., every abstract solution can be refined to a concrete solution, if a concrete solution exists). Under these conditions, the Scbr algorithms outperformed ground-level CBR and ground-level A^* under all conditions, and outperformed Refinement given 3 or more levels of abstraction.

However, these results were restricted to domains for which there are abstraction methods that can create hierarchies with the downward refinement property. Unfortunately, many abstraction hierarchies lack this property [BY94]. Determining the range of applicability of Scbr requires establishing whether it can lead to improvement in hierarchies without the downward refinement property (henceforth, "nonrefinable hierarchies").

This paper describes an experimental evaluation of the relative performance of Scbr in refinable and nonrefinable hierarchies. The evaluation showed that in the route-finding domain Scbr leads to increases in search efficiency nearly as great in nonrefinable hierarchies as in refinable hierarchies.

2 The Route-Finding Task

Route-finding was originally chosen to evaluate 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. This task involves finding an optimal or near-optimal path between a given pair of start and goal positions through a field containing obstacles. Fields consist of $N \times N$ arrays of positions, where N is a power of 2. Fields of this form are amenable to a simple abstraction hierarchy in which an abstract position at Level 1 position $\langle R, C\rangle$ (zero indexing) abstracts over the following four ground-level positions: $\langle 2 \times R, 2 \times C\rangle, \langle 2 \times R, 2 \times C + 1\rangle, \langle 2 \times R + 1, 2 \times C\rangle, \langle 2 \times R + 1, 2 \times C + 1\rangle$.

The goal of the route-finding task is to locate a route connecting the start and goal positions using 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. Each position is associated with 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 1. Each blocked position is associated with the empty set of traversal operators since traversal through them is impossible. 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.

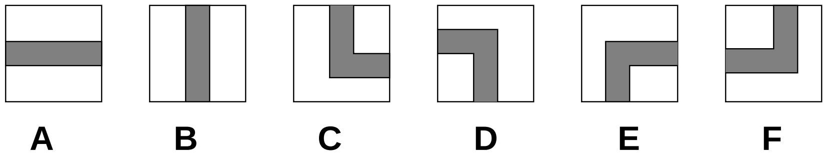

Fig. 1. The Set of Possible Traversal Operators Per Position

3 Search Using Abstraction Hierarchies

3.1 Refinement

Refinement [HMZM96] is a form of hierarchical problem solving in which a solution at one level of abstraction is used to guide search at a lower level of abstraction. One approach to Refinement is to use the length of the solution at a higher level of abstraction to estimate the path length at the lower level. In the abstraction method used in the route-finding task, this can be accomplished for positions that are in the abstraction path by multiplying the distance to the abstract goal along the abstract solution by 2 and adding the distance from the current position to the closest position that is a member of the next closer abstract position. The distance estimate can be used as the h^* estimate in A^* search. This process can be repeated at multiple levels of abstraction.

If the abstract solution is refinable into an optimal solution, then this estimate will be very accurate. If the abstract solution is not refinable, the estimate will still be admissible provided that the optimal solution is at least twice the length of the abstract solution plus 2. However, the treatment of positions off the abstract solution path is problematical. Ideally, one would like a heuristic for these points that is both admissible and also larger than the distance estimate for points on the abstract path (to focus search on position on the abstract path). However, for refinable abstract solutions the estimated distance for positions on the abstract solution path is very close to the actual distance, so it is not generally possible to find an admissible heuristic for points outside that is greater than the distance estimate for points on the path. In the experiments described below the distance estimate consisted of Manhattan distance times a large constant. This metric insures that the h^* value of every point off of the abstract path is higher than the h^* value of any point on the path but is still sensitive to the actual distance to the goal.

The experimental evaluation in [BA95] showed that Refinement (called "hierarchical A^*" in [BA95]) was significantly more efficient than ground-level A^* given even a single abstraction level, and its performance improved with more levels of abstraction.

3.2 Stratified CBR Algorithms

Stratified CBR algorithms can reuse case solutions stored at any abstraction level. Each algorithm 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.

The CLOSEST algorithm supports partial matching between new problems and previous solutions. Starting with the most specific matching cases (or the most abstract cases, if no cases match) CLOSEST finds the refinements of each case, 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. CLOSEST recursively calls these three steps until the ground level is reached, at which time it randomly selects and returns an adapted case.

The THRESHOLD algorithm, attempts to recognize situations in which adapting an existing case will be more expensive than problem solving *ab initio*. THRESHOLD behaves identically to CLOSEST if there are matching cases. However, if there are no matching cases, then 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 THRESHOLD treats these cases in the same manner as CLOSEST. If there are no such cases, then THRESHOLD uses REFINEMENT rather than CBR.

The process of adaptation is illustrated in Figures 2 and 3. Figure 2 shows a very simple new problem and a previous solution. Figure 3 shows an abstraction of the previous solution, an adaptation of this abstraction, and an adaptation of the previous solution at the ground level. The cases used by the SCBR algorithms can be generated by REFINEMENT. Given an abstraction hierarchy for a particular field and start and goal positions, REFINEMENT 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.

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. Generation of abstract cases by REFINEMENT is *top-down* abstract case creation. An alternative is *bottom-up* abstraction: starting with a ground-level case created by REFINEMENT or A^*, successively abstract the solution as many times as there are levels in the case library. In refinable abstraction hierarchies, top-down and bottom-up approaches generate identical abstract cases.

4 Case Retrieval and Adaptation in Nonrefinable Hierarchies

The abstraction method described in the previous section is not guaranteed to lead to refinable abstract solutions. Figure 4 illustrates a solution path through

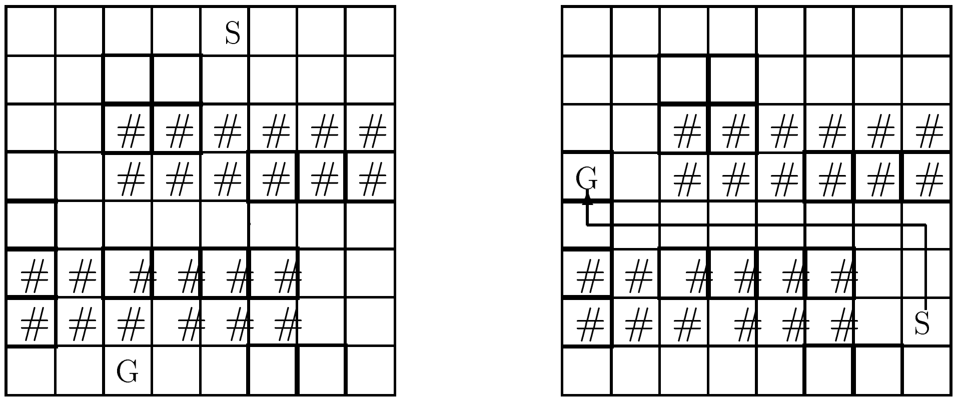

Fig. 2. A new problem and the solution to previous problem.

a 32×32 field.[1] Figure 5 shows the abstract solutions found by Refinement starting three levels of abstraction above the ground level. The two most abstract solutions show, incorrectly, that there is a path almost directly down from the start position to the bottom quarter of the field. Moreover, the abstract solutions fail to show the deviation up the right side of the field necessary to reach the bottom row of the field.

Errors like these can arise whenever a series of adjacent obstacles are arranged in a row that (1) crosses a pair of a region which is abstracted into a single position and that (2) leaves unobstructed positions on either side. For example, the barrier in the upper-left corner of the field shown in Figure 4 crosses abstract positions with width 4 leaving a path on either side. Abstract positions crossed by such a barrier contain operators A, C, D, E, and F, but not B. Refinement may construct a path across such a barrier because information about which side of the barrier the current position is on has been abstracted away. For example, in the most abstract solution shown in Figure 5 the path goes straight down from S because at this level it can't be determined which side of the barrier the starting position is on. The abstract solution at the next lower level of abstraction shows a horizontal path for one step followed by a downward path. This path was constructed from operators A and D because when operator D was applied no information was available about which side of the barrier the current position was on.

Nonrefinability can potential create problems for Scbr algorithms in two different ways. First, an abstract solution that appears to be reusable at an

[1] In this representation of solutions, the actual operators are not shown because there are insufficient typographic characters to represent all six. Instead, the direction of the path from the given position is represented by $\vee$, $\wedge$, $<$, and $>$. At the ground level, obstacles are represented by Xs.

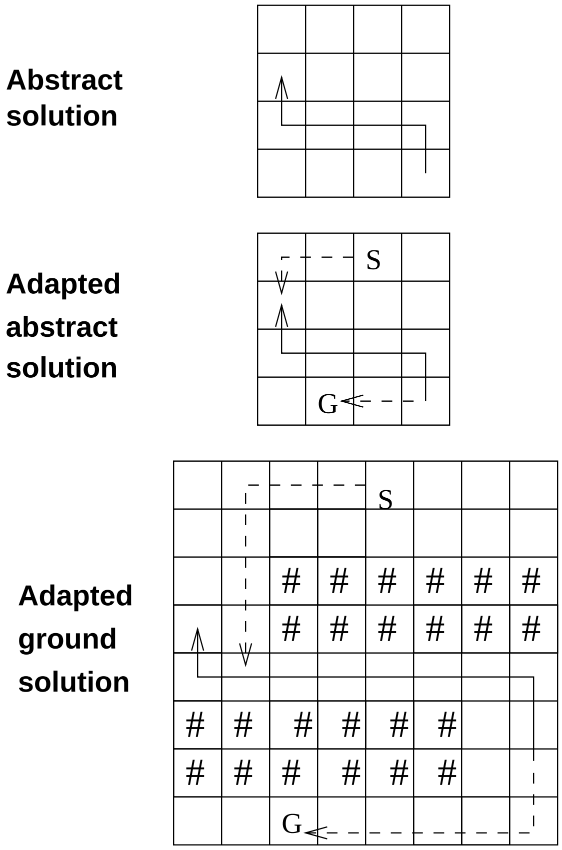

Fig. 3. Adaptation of an abstract solution leading to adaptation of the ground-level solution. Adaptation paths are shown as dotted lines.

abstract level may turn out not to have useful children. For example, if the start and goal positions in a new case were both along the left edge of the field shown in Figure 4, the most abstract case in Figure 5 would appear to be a good precedent, since its path appears to go the length of the left edge. However, the solution two abstraction levels lower goes down along the middle and right side of the field, but not the left side. Thus, reuse of this solution might be more expensive than simply generating a new path *ab initio*. Second, CLOSEST and THRESHOLD use REFINEMENT to find adaptation paths (e.g., the paths shown as dotted lines in Figure 3). The computational expense of this application of REFINEMENT will typically be greater in nonrefinable hierarchies. The cost of adaptation will therefore be higher as well.

The problems associated with nonrefinable hierarchies were not addressed in the experiments described in [BA95] because those experiments involved only fields satisfying the following conditions (referred to hereinafter as "field refin-

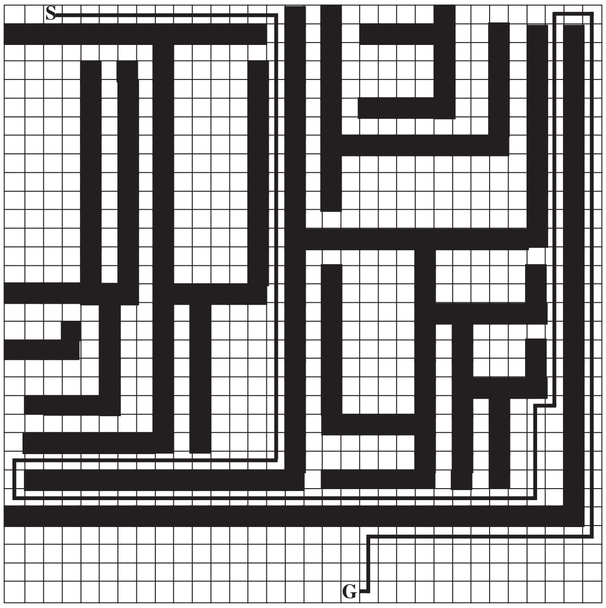

Fig. 4. A ground-level solution in a 32 × 32 field.

ability conditions"): (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. These conditions preclude the errors described above.

5 Empirical Evaluation

Intuitively, it seems that SCBR algorithms would perform well even in nonrefinable hierarchies because these algorithms would have access to previous successful solutions. This intuition suggests the following hypotheses:

1. SCBR can reduce search even in hierarchies without guaranteed refinability.
2. The performance of CLOSEST relative to that of REFINEMENT is better for nonrefinable than for refinable hierarchies.
3. Abstraction hierarchies created bottom-up lead to better performance by SCBR and THRESHOLD than those created top-down in nonrefinable hierarchies.

These hypotheses were tested in experiment that compared the performance of ground-level A*, REFINEMENT, and the SCBR algorithms as a function of the nature of the abstraction hierarchy. The dependent variable was computational expense as measured by the number of nodes expanded by A* in the execution of each algorithm. The independent variables were:

1. Whether the abstraction hierarchy guarantees refinability.

```
S...
v...
v...
>>G.

Sv......
.v......
.v......
.v......
.v......
v<....>v
>>>>>>^v
....G<<<

.S>>>>>v......>v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
.......v......^v
v<<<<<<<......^v
>>>>>>>>>>>>>>^v
.........v<<<<<<
.........G......
```

Fig. 5. Three abstract solutions created by REFINEMENT.

2. For nonrefinable hierarchies, whether abstract cases were created top-down or bottom-up.
3. The number of cases in the case library (1, 5, 10, or 20).
4. Whether the start and goal positions are selected randomly or are constrained to be on opposite sides of the field. This affects the likelihood that a case will be useful for reuse (start/goal positions on opposite sides increase the likelihood that some segment of a previous solution can be reused in any new problem).

Field size was fixed at 32×32 and the number of levels of abstraction was fixed at 3 (not counting the ground level). Refinable hierarchies were created by using fields that satisfy the "field refinability conditions" described above. Nonrefinable hierarchies were created by using fields that violate these conditions, such as the field shown in Figure 4. Six trials with 10 test cases each were run for each combination of refinable vs. nonrefinable hierarchy, case library size, top-down vs. bottom-up abstract case creation, and random vs. opposite sides selection of start and goal positions.

Figure 6 shows the mean speedup of CLOSEST over ground-level A^* on case libraries generated bottom-up as measured by the mean number of nodes expanded by ground-level A^* divided by the mean number of nodes expanded by the CLOSEST. These results provide initial confirmation for the first hypothesis: SCBR can reduce search even in hierarchies without guaranteed refinability. For both the random and opposite ends conditions the speedup was greater for refinable hierarchies than for nonrefinable hierarchies. However, for both refinable and nonrefinable hierarchies the speedup was greater for the opposite sides condition than for random start and goal positions.

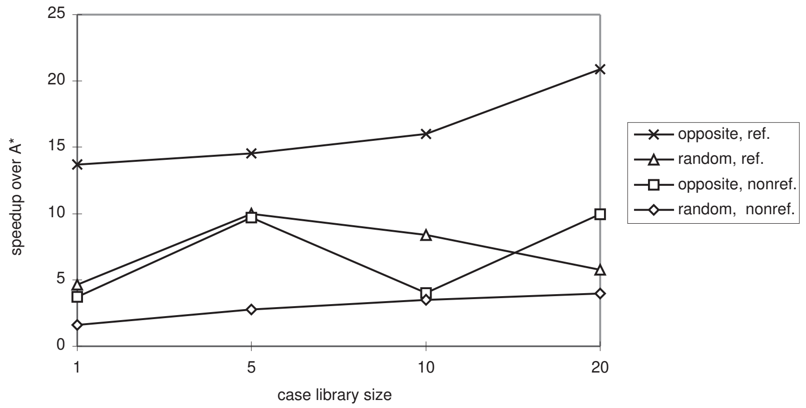

Fig. 6. Mean speedup of SCBR algorithms over ground-level A* (i.e., ratio of nodes expanded by ground-level A* to those expanded by COVER) for refinable vs. nonrefinable abstraction hierarchies and random vs. opposite sides selection of start and goal positions. All abstraction hierarchies were created bottom-up.

Figure 7 shows the ratio of the number of nodes expanded by REFINEMENT to the number of nodes expanded by CLOSEST as a function of case library size for refinable and nonrefinable hierarchies (with cases created bottom-up in the latter). Contrary to hypothesis 2, the speedup over REFINEMENT appeared to be greater for refinable than for nonrefinable hierarchies and opposite sides start/goal position selection. For random start/goal pairs, refinability had little effect on the relative performance of CLOSEST and REFINEMENT.

Figure 8 shows the ratio of nodes expanded by CLOSEST and THRESHOLD in abstraction hierarchies created top-down to nodes expanded in abstraction hierarchies created bottom-up in fields without the field refinability conditions (in field with the field refinability conditions, abstraction hierarchies created top-down and bottom-up are identical). Hypothesis 3 was confirmed only in THRESHOLD under the opposite sides condition, when bottom-up abstraction led to much higher performance. For CLOSEST and random start/goal positions bottom-up abstraction produced little improvement.

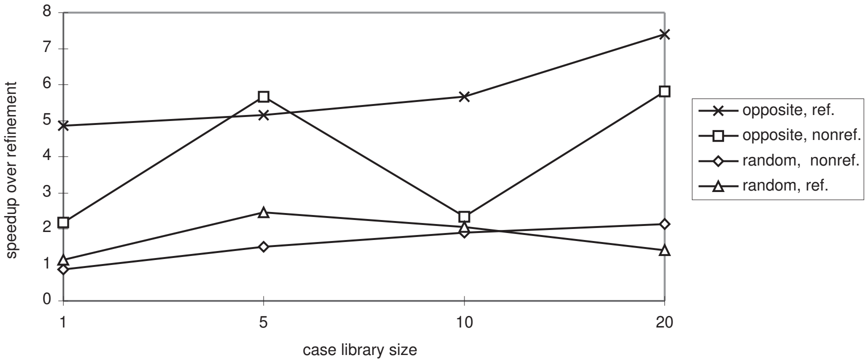

Fig. 7. Mean speedup of CLOSEST over REFINEMENT in refinable and nonrefinable abstraction hierarchies for random and opposite sides selection of start and goal positions.

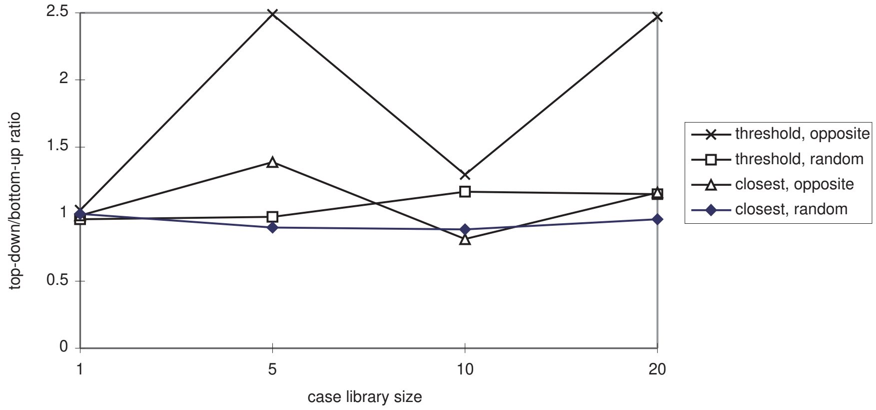

Fig. 8. Ratio of nodes expanded by CLOSEST and THRESHOLD in abstraction hierarchies created top-down to nodes expanded in abstraction hierarchies created bottom-up.

6 Discussion and Future Work

The empirical evaluation demonstrated that the range of applicability of SCBR algorithms extends to at least some domains lacking the downward refinement property. CLOSEST produces impressive speed-ups over both ground-level A* and REFINEMENT even for hierarchies that lack guaranteed refinability. Although the speed-ups were greater for refinable than for nonrefinable hierarchies, the variable with the greatest affect on the efficiency of SCBR appeared to be the distribution of start/goal positions. This variable determines the *a priori* likelihood that a previous solution can be usefully applied to a new problem.

Bottom-up abstraction had surprisingly little effect on the efficiency of CLOSEST. The only pronounced benefit occurred for THRESHOLD under the opposite sides condition, evidently because bottom-up abstraction makes estimation of adaptation costs at a high level more accurate. This permits THRESHOLD to recognize more accurately abstraction cases whose adaptation cases are greater than *ab initio* problem solving. The observation that the speed-up of the SCBR algorithms over REFINEMENT is less for nonrefinable than for refinable hierarchies even when bottom-up abstraction is used suggests that nonrefinability diminishes the performance of SCBR algorithms by increasing adaptation costs rather than by diminishing retrieval accuracy.

The evaluation suggests a number of additional questions. The abstraction hierarchies generated from fields like that shown in Figure 4 lack the guarantee of refinability, but the abstract solutions are close enough to enable REFINEMENT to outperform ground-level A^*. However, there are many abstraction hierarchies where lack of refinability makes REFINEMENT more expensive than ground-level search [BY94]. It is unclear from the empirical evaluation whether, or under what conditions, SCBR algorithms would outperform ground-level search in such domains.

To clarify the range of applicability of SCBR, additional experiments are needed to test the sensitivity of SCBR to the following independent variables:

(1) The degree of refinability. This will help to determine whether there are circumstances under which SCBR, but not REFINEMENT, outperforms ground-level search.

(2) The degree of intercase relevance. The apparent sensitivity of SCBR to the distribution of start/goal positions suggests that it would be desirable to develop a general metric for intercase relevance to help determine the conditions under which SCBR is appropriate.

We are currently investigating the utility of SCBR on other types of information-processing tasks, including a configuration task—constraint satisfaction—and an analytical task, analogical legal reasoning. In addition, we are attempting to apply SCBR to abstraction hierarchies created using the star abstraction method described in [HMZM96].

7 Conclusion

This paper has described the process of stratified CBR in the context of hierarchies without the downward refinement property. A source of nonrefinability was identified in route finding in fields that fail to satisfy two "field refinability conditions." An experimental evaluation using hierarchies derived from such fields showed that SCBR significantly decreased search cost in such hierarchies, although the speedup over ground-level A^* was not as great as in refinable hierarchies. The evaluation showed that the ratio of the search costs for REFINEMENT to those of SCBR was highest for opposite sides start/goal position selection, indicating that *a priori* likelihood that a previous solution can be usefully applied to a new problem is a more important factor than refinability in determining the relative performance of REFINEMENT and SCBR. Finally, a difference in SCBR search costs between case libraries created top-down in the process of REFINEMENT and those created bottom-up from a valid ground solution was observed only for THRESHOLD with opposite sides start/goal selection.

Acknowledgments

This research was supported by NSF Faculty Early Career Development Grant IRI-9502152. This work is an extension of collaborative research with David Aha.

References

[BA95] K. Branting and D. Aha. Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence(IJCAI-95)*, Montreal, Canada, August 20–25 1995.

[BW95] R. Bergmann and W. Wilke. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, 3(53–118), 1995.

[BW96] R. Bergmann and W. Wilke. On the role of abstraction in case-based reasoning. In *Proceedings of the Third European Workshop on Case-Based Reasoning (EWCR-96)*, pages 28–43, Lausanne, Switzerland, November 1996.

[BY94] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1994.

[HMZM96] R. Holte, T. Mkadmi, R. Zimmer, and A. MacDonald. Speeding up problem-solving by abstraction: A graph-oriented approach. *Artificial Intelligence*, 85(1–2):321–362, 1996.

[KH92] S. Khambamppati and J. Hendler. A validation-structure-based theory of plan modification. *Artificial Intelligence*, 55:193–258, 1992.

[Kno94] C. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 64, 1994.

[SC92] B. Smyth and P. Cunningham. Deja vu: A hierarchical case-based reasoning system for software design. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 587–589, Vienna, Austria, 1992.

[SK94] B. Smythe and M. Keane. Retrieving adaptable cases. In S. Wess, K. Althogg, and M. Richter, editors, *Topics in Case-Based Reasoning*, pages 209–220. Springer, 1994.