The Base Selection Task in Analogical Planning

Diane J. Cook
University of South Florida
Tarmpa, FL

cook@sol.usf.edu

Abstract

Analogical planning provides a means of solving problems where other machine learning methods fail, because it does not require numerous previous examples or a rich domain theory. Given a problem in an unfamiliar domain (the *target* case), an analogical planning system locates a successful plan in a similar domain (the *bast* case), and uses the similarities to generate the target plan. Unfortunately, the analogical planning process is expensive and inflexible Many of the limiting factors reside in the base selection step, which drives the analogy formation process. This paper describes two ways of increasing the effectiveness and efficiency of analogical planning. First, a parallel graph-match base selection algorithm is presented. A parallel implementation on the Connection Machine is described and shown to substantially decrease the complexity of base selection. Second, a base-case merge algorithm is shown to increase the flexibility of analogical planning by combining the benefits of several base cases when no single plan contributes enough information to the analogy. The effectiveness of this approach is demonstrated with examples from the domain of automatic programming.

1 Introduction

Analogy is a powerful planning tool. Engineers and scientists rarely attack a problem in an unfamiliar domain from scratch. Instead, then rely on their experience with solving problems in similar domains. They adapt known techniques, map constraints from a solved problem to the new problem, and modify existing solutions to fit the current problem specification. Given a novel problem (the *target* case), an analogical planner selects a similar, solved problem (the *base* case), computes a mapping between the base and target problem descriptions, and uses the mapping the adapt the base solution to the current domain. When examples are lacking and domain theory is scarce, the intelligent agent draws on past experience in similar situations to attack a new problem.

Although analogy is considered to be a powerful tool for machine planning, it is also viewed as an expensive task which is rarely applied to large-scale problems. As Buchanan states, analogical reasoning is a "pipe dream when matched against the harsh standards of robustness of commercial applications" [Buchanan, 1990]. A second limitation of analogical planning systems is flexibility. In inductive learning systems, when the learner is not performing well enough more examples can be input to improve the hypothesis. In analogical learning systems, the strength of the output plan depends solely on the amount of similarity between base and target and on the strength of the relationship between known and inferred information. If no base cases sufficiently match the target, the system cannot, perform as needed.

The purpose of this research is to strengthen the effectiveness and efficiency of analogical planning, making it a more useful tool for machine learning. This paper describes a method of achieving the staled goal by focusing on the aspect of analogical planning that is the least researched and perhaps the most limiting: base selection. The base selection process is very expensive, yet the selected base case greatly affects the outcome of the planning process. In this paper, two improvemerits to base selection are discussed. First, a parallel graph-match base selection algorithm is described that reduces the complexity of the search process. Second, a method of merging congruent base cases when no single base case sufficiently matches the target is demonstrated to improve the flexibility and applicability of analogical planning. This approach is illustrated on automatic programming examples, and is shown to transform analogy into an effective, efficient planner for large-scale problems.

2 Analogical Planning

Analogy can be defined as an inference that if two or more things agree in some respects they will probably agree in others. The strength of the analogically-generated inferences depends on the type and strength of the relationship between the known shared properties and the inferred shared properties.

The three main steps of analogy plan formation are *base selection, map formation,* and *inference generation.* Given the target problem that needs a solution, the ana-

logical planner selects a base problem that has a successful plan and that shares crucial properties of the problem with the target. When the appropriate base case has been found, the system constructs the analogical mapping. Once the mapping is formed, the analogy system uses the information describing the base and the *base* —► *target* mapping to infer the target plan.

For example, a programmer rarely develops his code from scratch. Instead, he pulls ideas and pieces of code from similar programs he has written in the past, and modifies them to fit the peculiarities of the current goal. If he wants to implement a program that computes real-number division to a specific accuracy, he may benefit most from analogically deriving the program. First, he finds a program in his database that computes the cube-root of a real number to a specific precision (base selection). He senses the underlying similarities between the type of information used and the goal of the programs and pinpoints the correspondences (map formation). Using these correspondences, he maps the existing code to fit the current situation, and enters the new analogically-implemented program into his database (inference generation).

Much of analogy research has focuses primarily on the creation of the *base* —► *target* mapping. Centner's [Clement and Centner, 1989; Centner, 1988; Centner and Toupin, 1986] theory of *systematictty* shows that humans use analogies between concepts whose underlying *structures are* the same. Other popular methods of map formation include Carbonell's *transformational analogy* approach [Carbonell, 1983] which uses means-ends analysis to reduce the difference between base and target, and the *explanation-based* approach of Kedar-Cabelli [Kedar-Cabelli, 1988], which constructs an *explanation* of the difference between base and target concepts.

.Analogical reasoning, though generally considered intuitive and compelling, is often looked upon as a computationally infeasible form of learning. This feeling is fueled by the fact that much attention has been given to the task of finding coherent mappings between base and target. On the other hand, little attention has been given to computationally-complex tasks such as base selection. Work focusing on the base selection task includes case-based reasoning research [Hammond, 1986a; Hammond, 1986b; Kolodner *et* al., 1985] which uses plan keywords to organize the databa.se and select a base case.

3 Parallel Analogical Planning

A goal of this research is to design an analogical planning system efficient and effective enough to apply to large-scale problem domains. The implemented system is based on a graph match algorithm, which compares two plans represented as graphs to determine the similarity of the plans. The system is called ANAGRAM (ANAlogical GRAph Match). Given a target problem specification represented in graph form, ANAGRAM'S colored graph match technique generates a plan which will achieve the target goal.

Each plan in ANAGRAM'S database is represented as a directed acyclic graph. The nodes in the graph represent object names and attributes, and the links represent relations between objects. A base case is selected if the structure of the base graph matches the structure of the target graph (embodying Centner's theory of *systematicity).*

The system accepts as input two subgraphs, representing the target problem's initial-state description and goal-state specification. ANAGRAM searches through the database, finding the best match for both subgraphs. Using the output of the individual graph matches, ANAGRAM then maps the base plan over to the target domain to generate a solution for the target problem. If the resulting plan is unsuccessful, or if no sufficiently similar base cases are found, the system attempts to merge several base cases that are similar to each other and to the target problem. The result is a *virtual base graph* that eliminates anomalies and generalizes various options in the plan to the extent that it covers the target domain.

3.1 Parallel Graph Match

ANAGRAM'S graph match algorithm is implemented on a Connection Machine 2, a Single-Instruction-Multiple-Data (SIMD) machine with 32,768 processors. The algorithms are implemented in *Lisp, a parallel extension of Common Lisp. ANAGRAM makes use of parallel computation in two aspects of base case selection. The graph match algorithm which compares a base graph with the target is implemented in parallel, and each base case in the database is examined simultaneously.

In the parallel graph match, the data structure representing each graph node is stored in a separate CM processor. Each node in the first graph looks for a match at the same time with a node at the same level in the second graph. An integer is assigned to each node in the base graph. When a match is found for a node from the base graph, the corresponding target node is assigned the same integer When testing if node n_1 matches n_2, the link labels *and* the integers assigned to the parents and children of n_1` are matched with the corresponding labels and integers for n_2.

The data structure describing each node contains the level of the node in the graph,[1], the node label, and a tuple which consists of the incoming link labels and corresponding parent nodes as well as the outgoing link labels and corresponding child nodes. Initially, nothing is known about parent or child nodes, so the parent/child integer slots are set to "?". Each node simultaneously looks for a match by comparing levels and tuples. If two complete tuples match (the tuples are complete if they have no "?"s), the match is added to the global map (gmap) and a unique integer is assigned to the two nodes. If a tuple is incomplete, it generates a list of partial matches. When one or more matches are found, each node in both graphs simultaneously updates its tuples. Assigned integers are propagated across the links. Once the tuples are updated, matches between incomplete tuples are checked once again — if they no longer match, the algorithm returns failure.

[1]The *level* of a node in a DAG is defined here as the shortest path length from the node to any leaf in the DAG.

If no unique matches are found for any of the nodes, the algorithm takes one node from the list of nodes having more than one candidate match, and randomly selects a match for the node. If there are nodes from the first graph that cannot be matched with any node from the second graph, the algorithm returns failure. The process is successfully completed when a match is found for each node in the first graph.

3.2 Parallel Base Selection

Probably the greatest speedup occurs when base selection is parallelized. Normally, the base selection process is very time consuming because each potential base solution must be compared with the target problem specification. Fortunately, each of these comparisons is independent, of t he others, so the bases can be examined in parallel.

The time saved by performing base selection in parallel is enormous, as would be expected. However, as the database grows large, it is not possible to examine each of the base cases in parallel because of the limit on the number of processors. On the Connection Machine it is possible to create virtual processors (extra processors which overlay the existing physical processors). Using the virtual processors eventually degrades the performance of the algorithm, however, and even with the existence of virtual processors it is not possible to examine a huge database completely in parallel because of memory limitations.

fur this reason, as the database becomes very large, indices are attached to each entry in the database. The indices chosen are ones which do not add new information to the graphs; rather, they compress the information contained in the graph. These features include graph invariants such as the size of the graph and the degrees of each node. Some indices also encapsulate information contained in the graph such as the list of operators used in the plan. Entries in the database with the same indices are then grouped together. When base selection is performed, the algorithm uses the attributes of the target specification to select a group of base cases from the database, and examines these base cases in parallel. In fact, the number of cases allowed in each database subgroup can be calculated from the size of the individual plans and the number of processors available on the machine, so that it is always possible to examine the plans within a single group in. parallel.

3.3 Example

This section illustrates the application of ANAGRAM'S parallel algorithms to automatic programming. Analogy proves to be a valuable tool for automatic programming. Instead of constructing programs from scratch or working from abstract theories, many scientists start with existing program segments that achieve similar goals, and modify them to meet their current needs.

The following example is based on a program segment described by Dershowitz [Dershowitz, 1986]. The base program, a program that computes the cube-root of *a* within an error tolerance *e,* is used to generate a program that computes *c/d* within an error tolerance *e.* Figure 1 illustrates the goal-state subgraphs of the base and target plans.

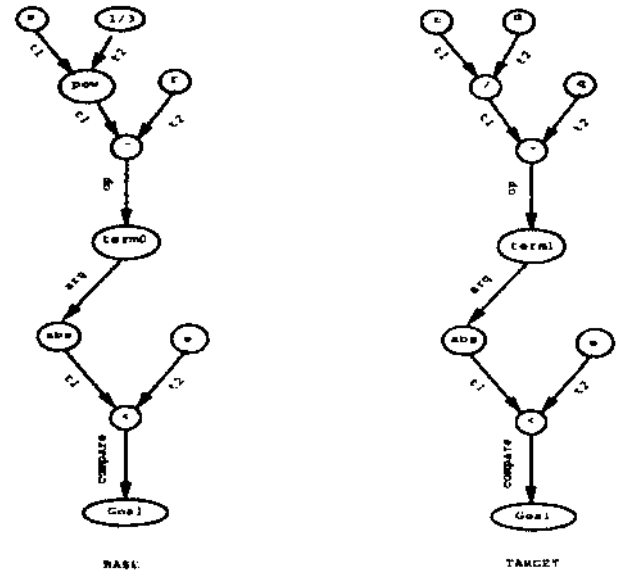

Figure 1: Base and Target Goal-State Subgraphs

ANAGRAM successfully matches the initial-state subgraphs and the goal-state subgraphs. Tables 1 and 2 show the tuples representing the nodes in the two goal-state subgraphs at the end of the graph match. The first column represents the node label, the second column lists the integer assigned to the node, the third column shows the corresponding tuple, and the fourth column lists the candidate matches / final match for the node. Note that the tuples are represents as ({(child-integer outlink-label) } {(parent-integer inlink-label) }).

node	*int*	*tuple*	*match*
Goal	11	({} {(12 compare)})	Goal
<	12	({(1 compare)} {(13 t1) (14 t2)})	<
abs	13	({(2 t1)} {15 arg)})	abs
e	14	({(2 t2)} {})	e
term0	15	({(3 arg)} {(16 op)})	term1
-	16	({(5 op)} {(17 t1) (18 t2)})	-
pow	17	({(6 t1)} {(19 t1) (20 t2)})	/
r	18	({(6 t2)} {})	q
a	19	({(7 t1)} {})	c
1/3	20	({(7 t2)} {})	d

node	*int*	*tuple*	*match*
Goal	11	({} {(12 compare)})	Goal
<	12	({(1 compare)} {(13 t1) (14 t2)})	<
abs	13	({(2 t1)} {15 arg)})	abs
e	14	({(2 t2)} {})	e
term1	15	({(3 arg)} {(16 op)})	term0
-	16	({(5 op)} {(17 t1) (18 t2)})	-
/	17	({(6 t1)} {(19 t1) (20 t2)})	pow
q	18	({(6 t2)} {})	r
c	19	({(7 t1)} {})	a
d	20	({(7 t2)} {})	1/3

The result of the match is the set of node-to-node matches that comprise the global map:

$$\{\text{Goal} \rightarrow \text{Goal}, < \rightarrow <, \text{abs} \rightarrow \text{abs}, \text{e} \rightarrow \text{e}, - \rightarrow -, \text{term0} \rightarrow \text{term1}, \text{pow} \rightarrow /, \text{r} \rightarrow \text{q}, \text{a} \rightarrow \text{c}, 1/3 \rightarrow \text{d}\}.$$

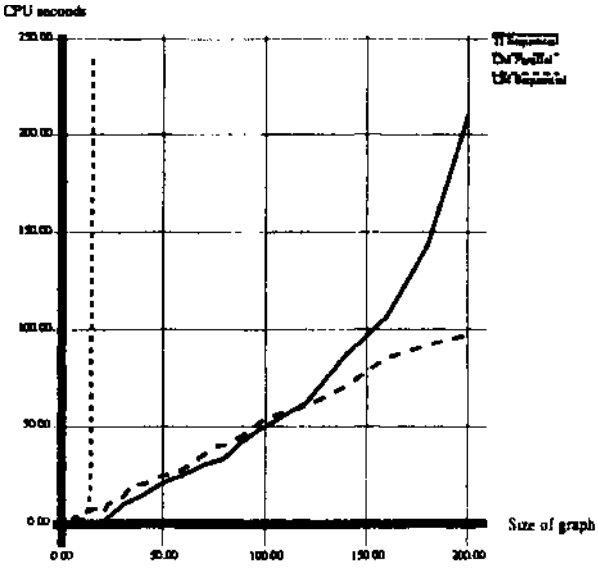

Figure 2: Graph Match Results

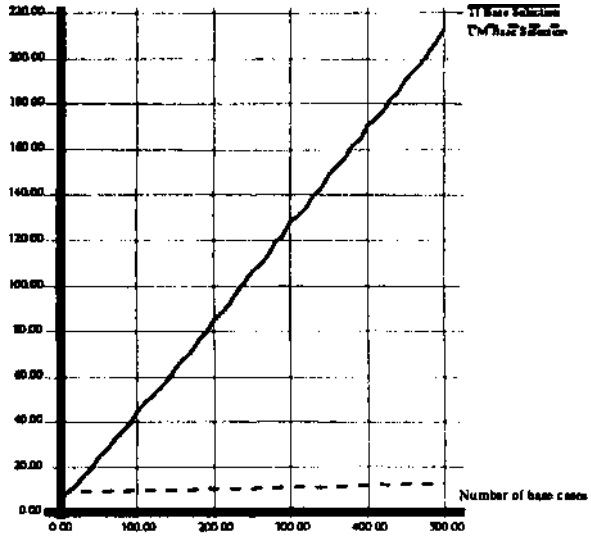

Figure 3: Base Selection Results

The mapping is used to transform the base program into a program that achieves the target goal. The code segments are shown below

```
/* Compute cube-root of a */          /* Compute c/d */
  begin cube-root                       begin target
        /* Initial State */                 /* Initial State */
     assert a ≥ 0, e > 0                  assert d > c ≥ 0, e > 0
        /* Goal State */                    /* Goal State */
     goal | a^1/3 − r |< e                goal | c/d − q |< e
     (r, s) := (0, a + 1)                 (q, s) := (0, c + 1)
     loop L3 until s ≤ e        ⟹        loop L3 until s ≤ e
        s := s/2                             s := s/2
        if (r + s)^3 ≤ a                     if (q + s) × d ≤ c
           then r := r + s                      then q := q + s
        endif                                endif
        repeat                               repeat
  end                                   end
```

After the target plan generation is complete, the plan is tested. If the plan is not successful, ANAGRAM may attempt a new analogy, merging several base cases to form a more flexible analogy. This process is described is section *4.* If the plan is successful, the new plan is indexed by its features and added the database.

3.4 Analysis

Figures 2 and 3 show the results of running the parallel and sequential graph match and base selection algorithms. For these experiments, a database of graphs representing program specifications was created. In the first experiment, the graph sizes range from 4 to 200 nodes. Noise was minimized by generating 50 graphs of each desired size. Three implementations of the graph match algorithm were tested. The first curve represents the sequential version of the algorithm implemented on a Texas Instruments Explorer II. The second curve shows the results of the parallel implementation running on a Connection Machine 2. The third curve represents the algorithm running on the CM 2 without taking advantage of the parallelism.

The results shown in the graph match graph come very close to expectation. The actual complexity of the graph match algorithm is $O(\lfloor \frac{n-h}{p} \rfloor + ((n-h) \bmod p))$, where n is the number of nodes in the graph, h is the height of the graph, and p represents the number of physical processors residing on the machine. This is linear speedup from the sequential algorithm, which has complexity $O(n^2)$.

The second experiment compares sequential and parallel implementations of the base selection algorithm. From a database of 500 graphs, one base case was chosen. When the entire database must be searched, the complexity of the sequential algorithm is proportional to the size of the database, while the parallel algorithm is almost constant. The sequential algorithm actually has complexity $O(bn^2)$, while the parallel algorithm is $O(\lfloor \frac{b(n-h)}{p} \rfloor + (b(n-h) \bmod p))$.

The results of testing these algorithms reveals that an analogy system implemented on a massively parallel machine is a powerful and efficient planning tool. Taking advantage of available hardware allows analogical planning to be applied to complex problems without unreasonable time expense.

4 Merging Congruent Cases

Implementing an analogy system on a parallel machine transforms analogical planning into an efficient process. However, the tool still suffers from a lack of flexibility. In particular, a base case must be found that is sufficiently similar to the target to analogically generate each step of the target plan. Because in actuality there exists a great diversity among graph representations of plans,

the chance of finding such a similar base case is slim. Although analogical learning is sometimes preferred to inductive learning because it does not require multiple examples, there are many instances in which multiple base cases would strengthen an analogy.

One example of using multiple base cases is incremental analogy [Burstein, 1988]. One base case may provide some of the information needed for the target, but not all. Another base case may provide the remaining needed information, but nothing else. An analogy formed between the target and either one of these bases would be insufficient, but the merging of the two separate analogies results in a complete, useful analogy.

A second way of using multiple base cases is to merge similar base cases, resulting in a "virtual" base case. This virtual base case is more beneficial to the analogy than a single case, because it removes anomalies and generalizes alternative operations. Furthermore, merging base cases focuses the analogy on the relevant, aspects of the base cases, because those aspects of previous plans that are beneficial to the target are retained in the virtual base. Base cases that are sufficiently similar in structure to be merged together are termed *congruent* base cases.[2]

4.1 The Merge Algorithm

This section describes how ANAGRAM merges congruent base cases to enhance its analogy-formation and problem-solving capabilities. The issue of deciding when to merge plans is first addressed, and then the merging method is described.

ANAGRAM uses the graph-merge algorithm in the following cases:

- A match is found between the target initial/goal states and the base initial/goal states, but the intermediate steps in the base case are not mappable or cannot be applied in the target domain.
- No base case matches perfectly, but several match rather closely. Moreover, the unmappable parts either 1) are generalizable in a way that map to the target plan, or 2) they do not overlap (the base cases fail to match the target at distinct points in the target graph).

When selecting bases for merging, the algorithm chooses cases based on ease of generalization. The types of graph merge are (in order of preference):

1 *Merging graphs with distinct base/target differences.* The simplest and most beneficial method of merging occurs when the candidate base graphs match each other and their differences with the target do not overlap, as shown in figure *4.* The mappable portions of each base are retained in the virtual base graph.

2. *Relaxing order constraints.* The graph match algorithm looks only for matches between nodes at corresponding levels in the graphs. Often an operator re-ordering will make a base case fit the target exactly. Comparing multiple plans whose only difference is the order of operations, it becomes apparent that maneuvering the operators may also solve the target problem. When this situation arises, *Anagram* looks for correlations between the order of operations and ordering constraints in the initial or goal state.

3. *Disjunction/'generalization of subgraphs.* When comparing subgraphs whose "defective" parts *do* overlap, ANAGRAM generalizes the overlapping subgraphs. The methods of generalization correspond to those found in most induction systems, such as adding disjunctions and climbing generalization trees. The purpose of the generalization is to abstract the non-mappable subgraphs to th extent that the generalized subgraph will cover the target case. As the number of base cases included in the generalization increases, the method of merging base graphs begins to behave similar to pure indue tion.

[2]This term is borrowed from geometry, where two triangles are congruent if they have the same angles and proportions of side lengths.

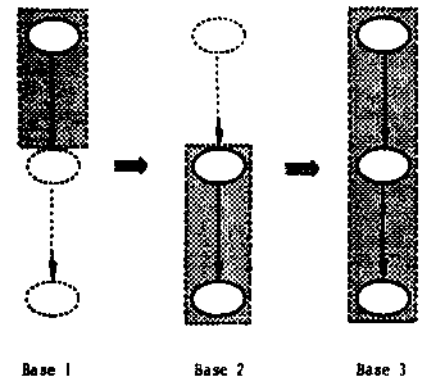

Figure 4: Distinct Base Case Merge

4.2 Example

This section describes an application of ANAGRAM'S graph merge algorithm to a problem in the domain of automatic programming. In this example, the target goal is to construct a program that uses inorder traversal to traverse a given binary tree. The initial and goal state descriptions are given:

assert (tree ∈ binary-trees) **and** null(vlist)
goal ∀(*y* ∈ *vlist*)
[left-son(y) **before** y **before** right-son(y)]

Among the base cases residing in the database are the algorithms for preorder and postorder tree traversal. The matches are equally good with either base case, so the selection process arbitrarily chooses the preorder plan. However, the resulting program is

```
begin inorder
    x := root(tree)
    unless null(tree)
        vlist := append(vlist, x)
        inorder(left-son(x), vlist)
        inorder(right-son(x), vlist)
end
```

This does not solve the problem (remember that the graph matcher does not consider re-ordering the operators). The system then compares the preorder and postorder algorithms, and notices that the operators are the same in the two algorithms, but the order of application is different. The goal description in both base cases places ordering constraints on the output:

goal-preorder: $\forall(y \in vlist)$
[y before left-son(y) before right-son(y)]
goal-postorder: $\forall(y \in vlist)$
[left-son(y) before right-son(y) before y]

By comparing the order of operators with the order imposed by the goal description, ANAGRAM observes that the placement of *y* corresponds with the *append* operation, *lcft-son(y)* with the recursive call to the left son, and *right-son(y)* with the recursive call to the right son. ANAGRAM is able to generate a virtual base graph that contains the correspondences between the three operators and the desired order of elements in *vlist.* The resulting target plan shown below is successful.

```
begin inorder
    x := root(tree)
    unless null(tree)
        inorder(left-son(x), vlist)
        vlist := append(vlist, x)
        inorder(right-son(x), vlist)
end
```

5 Conclusions and Future Directions

This paper presents two steps that make analogical planning a more effective and efficient machine learning tool. There are many extensions of this research which could push the advancement of the area even further. Searching for and combining partial matches would be a beneficial alternative to merging base cases. Developing a fuzzy graph match would allow a broader range of analogies to be created. Other aspects of the analogical planning task could be parallelized, such as inference generation. In addition, other parallel architectures should be analyzed for their potential to speed up the analogy process. For example, a hypercube M1MD architecture could be used to develop multiple independent analogies for the same target, or expand multiple partial matches simultaneously.

In this paper, the parallel graph match and base selection algorithms were described as implemented on the Connection Machine. The results were substantially faster than results from the comparable sequential method. In addition, a method of combining the benefits of several successful plans by merging multiple base cases was presented. The extension of analogy to utilize multiple base cases was shown to increase the effectiveness of analogical planning in large problem domains. By designing efficient parallel analogy algorithms, and by extending the applicability of analogical planning, this research offers a valuable step toward the automation of analogical planning.

References

[Buchanan, 1990] B ruce G Buchanan. Can machine learning offer anything to expert systems? *Machine Learning,* pages 5-8, 1990.

[Burstein, 1988] M H Burstein. I ncremental learning from multiple analogies. In A Prieditis, editor, *Analogica,* chapter 2, pages 37 62. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[Carbonell, 1983] J G Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R S Michalski, J G Carbonell, and T M Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. I*, chapter 5, pages 137-162. Tioga, Palo Alto, CA, 1983.

[Clement and Centner, 1989] Cath erine A Clement and Dedre Gentner. Systematicaty as a selection constraint in analogical mapping. Technical Report UIUCDCS-R-89-1558, University of Illinois, Urbana-Champaign, IL, September 1989.

[Dershowitz, 1986] N Dershowitz. Programming by analogy. In R S Michalski, J G Carbonell, and T M Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II,* chapter 15, pages 393-421. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[Gentner and Toupin, 1986] Dedre Gentner and C Toupin. Systematicity and surface similarity in the development of analogy. *Cognitive Science,* 10:277 300, 1986.

[Gentner, 1988] Dedre Gentner. Analogical inference and analogical access. In A Prieditis, editor, *Analogtea,* chapter 3, pages 63-88. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[Hammond, 1986a] K Hammond. Chef: A model of case-based planning. In *Proceedings of the 1986 National Conference on Artificial Intelligence,* pages 267-271, 1986.

[Hammond, 1986b] K J Hammond. *Case-based Wanning: An Integrated Theory of Planning, Learning and Memory.* PhD thesis, Yale University, Boston, MA, 1986.

[Kedar-Cabelli, 1988] S Kedar-Cabelli. Toward a computational model of purpose-directed analogy. In A Prieditis, editor, *Analogica,* chapter 4, pages 89-108. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[Kolodner *et al,* 1985] J L Kolodner, R L Simpson, and K Sycara. A process model of case-based reasoning in problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 284-290, 1985.