# Abstracting the Tower of Hanoi

Craig A. Knoblock*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
cak@cs.cmu.edu

**Abstract**

This paper describes an automated approach to generating abstractions for the Tower of Hanoi and analyzes the use of these abstractions for problem solving. The analysis shows that the learned abstractions produce an exponential reduction in the size of the search space. Since few problem solvers actually explore the entire search space, the paper also presents an empirical analysis of the speedup provided by abstraction when a heuristic search is employed. The empirical analysis shows that the benefit of abstraction is largely determined by the portion of the base-level search space explored. Thus, using breadth-first search, which searches the entire space, abstraction provides an exponential reduction in search. However, using a depth-first search, the search reduction is smaller and depends on the amount of backtracking required to solve the problem.

## 1 Introduction

The Tower of Hanoi puzzle has be studied extensively in the problem-solving literature [Ernst, 1969, Eavarone, 1969, Korf, 1980, Ernst and Goldstein, 1982, Knoblock, 1990b, Benjamin *et al.*, 1990, Christensen, 1990]. Previous work has primarily focused on various approaches for generating abstractions, but has largely ignored the issues in using the abstractions for problem solving. This papers reviews an approach that generates abstractions for the Tower of Hanoi, shows that the abstractions provide an exponential reduction in the size of the search space, and then analyzes the use of the abstractions in the PRODIGY problem solver [Minton *et al.*, 1989, Carbonell *et al.*, 1991].

*The author is supported by an Air Force Laboratory Graduate Fellowship through the Human Resources Laboratory at Brooks AFB. This research was sponsored in part by the Office of Naval Research under contract number N00014-84-K-0415, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, Amendment 20, under contract number F33615-87-C-1499. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the Defense Advanced Research Projects Agency or the US Government.

The paper is organized as follows. The next section describes the Tower of Hanoi problem and presents a representation of the problem. Section 3 reviews an algorithm for automatically generating abstractions and describes the abstractions the algorithm generates for the Tower of Hanoi. Section 4 describes how the abstractions are used for hierarchical problem solving and presents both analytical and empirical results on the search reduction provided by the abstractions. Section 5 compares the abstraction generation method in this paper to other approaches that have been applied to the Tower of Hanoi. The last section concludes with some general remarks about when abstractions will be useful and how they might be used more effectively.

# 2 Representing the Tower of Hanoi

The Tower of Hanoi puzzle involves moving a pile of different size disks from one peg to another using an intermediate peg. Only one disk at a time can be moved, a disk can only be moved if it is the top disk on a pile, and a larger disk can never be placed on a smaller one. Figure 1 shows the initial and goal states of a three-disk problem.

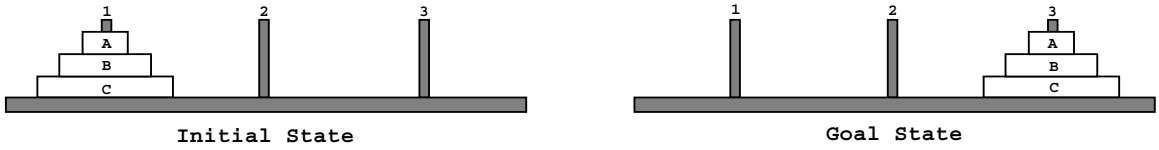


Figure 1: Intial and Goal States in the Tower of Hanoi

The most straight-forward axiomatization of this problem consists of an operator for moving each disk between each pair of pegs. For the three-disk problem, this axiomatization requires 18 operators. Table 1 shows the operator for moving disk C, the largest disk, from peg 1 to 3. The preconditions require that disk C is initially on peg 1 and that neither disk A nor B are on peg 1 or 3. This representation is far from the most concise one, but it is used in this paper to simplify the exposition. The basic ideas described in this paper apply to and have been tested on more compact representations.

```
(Move_DiskC_From_Peg1_to_Peg3
  (preconds ((and (on diskC peg1)
                  (~ (on diskB peg1))
                  (~ (on diskA peg1))
                  (~ (on diskB peg3))
                  (~ (on diskA peg3)))))
  (effects ((del (on diskC peg1))
            (add (on diskC peg3)))))
```

Table 1: Example Operator in the Tower of Hanoi Domain

The size of the problems in the Tower of Hanoi puzzle vary based on the number of disks. The number of possible states for a given puzzle with $n$ disks is $3^n$ since each disk can be on

one of the three pegs. The state space for the three-disk puzzle is shown in Figure 2 [Nilsson, 1971]. Each node represents a state and is labeled with the a picture of the state, and each arrow represents an operator that can be applied to reach the adjacent state. A solution to the three-disk problem given above consists of any path through the state space that starts at the initial state and terminates at the goal state. The shortest solution follows the path along the diagonal between the initial and goal states.

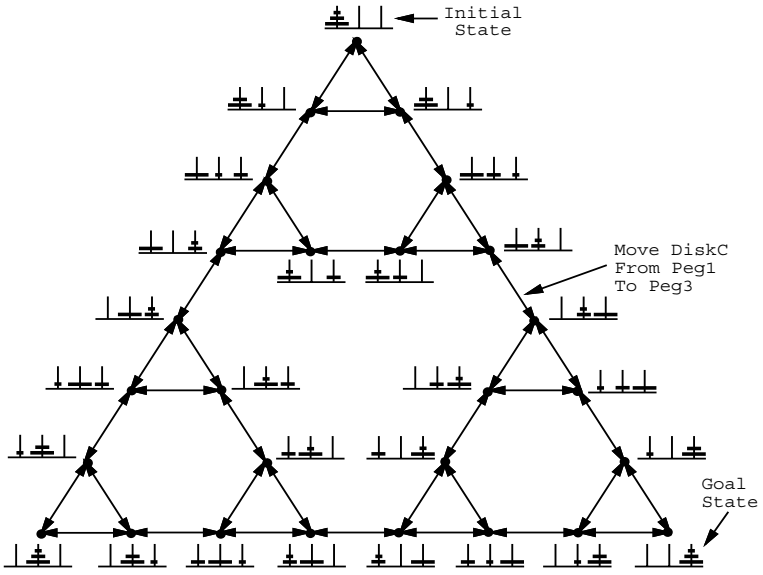


Figure 2: State Space for the Three-Disk Tower of Hanoi Puzzle

# 3 Automatically Generating Abstractions

This section reviews an algorithm for generating abstraction hierarchies [Knoblock, 1990b] and describes the hierarchy produced for the three-disk Tower of Hanoi problem. The algorithm is given a set of operators, which describe a domain, and it produces an abstraction hierarchy by partitioning and ordering the literals in the domain. *Literals* are possible negated atomic formulae (e.g., `(on diskC peg1)`), which are used to define the preconditions and effects of the operators. A hierarchy of abstraction spaces is formed by removing successive classes of literals, such that each abstraction space is an approximation of the original problem space. The hierarchy is ordered such that the highest level is the most abstract. The final hierarchy has the *ordered monotonicity property* [Knoblock, 1990a], which requires that the literals are partitioned in such a way that the achievement of a literal introduced at one level cannot change the truth value of a literal in a more abstract level.

The algorithm for producing a hierarchy of abstraction spaces is shown in Table 2. The algorithm forms a directed graph, where the vertices of the graph represent one or more literals (called a literal class) and the edges represent constraints between literals. A directed edge from one literal class to another indicates that the first literal class must be higher or at the same level in the abstraction hierarchy as the second literal class. A literal and its

**Input**: The set of operators for a domain.
**Output**: A hierarchy of abstraction spaces.

```
Create_Abstraction_Hierarchy(OPERATORS)
1. ForEach OP in OPERATORS
       ForEach LIT1 in Effects(OP)
       i. ForEach LIT2 in Effects(OP)
              Add_Directed_Edge(LIT1,LIT2,GRAPH)
       ii. ForEach LIT2 in Preconditions(OP)
              Add_Directed_Edge(LIT1,LIT2,GRAPH)
2. Combine_Strongly_Connected_Components(GRAPH)
3. Topological_Sort(GRAPH)
```

Table 2: Algorithm for Producing Abstraction Hierarchies

negation are always placed in the same class since they directly interact with one another. The algorithm first adds constraints that guarantee the achievement of a particular literal could never require adding or deleting a literal higher in the abstraction hierarchy. Next, the algorithm removes the cycles in the graph by combining the literals within a strongly connected component into a single class. Third, the partial order of literal classes is transformed into a total order, which forms an abstraction hierarchy.

For the three-disk Tower of Hanoi problem, this algorithm generates the directed graph shown in Figure 3. The ovals in the graph represent the strongly connected components. The literals within each component must be placed in the same abstraction level. The arrows between the components specify the constraints on the order in which the literal classes can be removed to form the abstraction spaces.

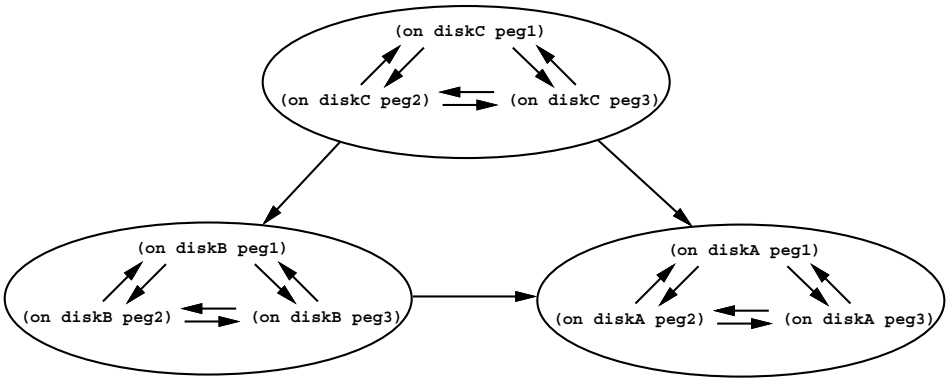


Figure 3: Constraints on the Literals in Tower of Hanoi Domain

The resulting abstraction hierarchy, shown in Figure 4, partitions the literals of the domain based on the size of the disks. For the three-disk puzzle, the highest abstraction level includes literals involving only the largest disk, the next level includes both the largest and middle size disk, and the third level includes all three disks. The disks can be divided into separate abstraction levels since the steps necessary to move a given disk can always

be inserted into an abstract plan without interfering with any larger disks. For a n-disk problem, the algorithm produces a n-level abstraction hierarchy.

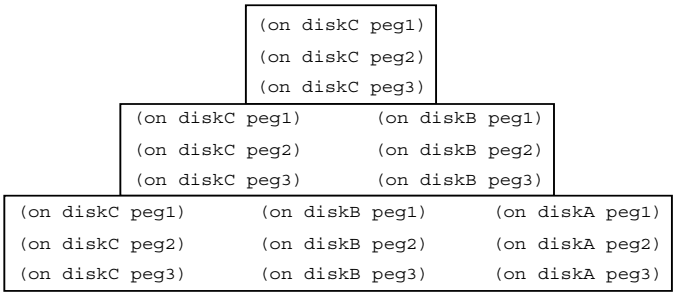


Figure 4: Abstraction Hierarchy for the Tower of Hanoi

The reduced state spaces of the three-disk puzzle are shown in Figure 5. The state space on the left shows the result of removing the smallest disk. The nodes that differ only by the location of the smallest disk are collapsed into a single node, reducing the state space from 27 states to 9 states. The operators that are not relevant to a given state space are replaced with dotted lines. Removing the two smaller disks produces a state space with only 3 states, which is shown on the right of Figure 5.

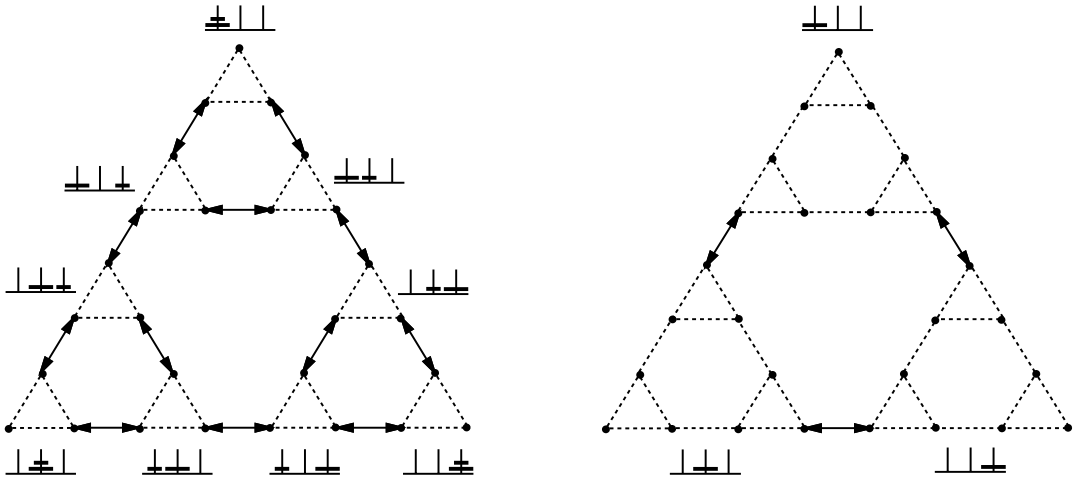

Figure 5: Reduced State Spaces for the Three-Disk Puzzle

# 4 Using Abstractions for Problem Solving

This section describes how abstractions are used for problem solving and the effect of the abstractions in reducing search. The use of the abstraction hierarchy described in the previous section provides an exponential reduction in the size of the search space [Knoblock, 1990b]. This section describes how such dramatic reductions are achieved in this domain, verifies the result empirically, and then explores the effect of the search space reduction on a depth-first search.

## 4.1 Hierarchical Problem Solving

The abstractions described in the previous section can be used to solve problems hierarchically. Hierarchical problem solving maps the initial and goals states into the most abstract space, solves the abstract problem, and then uses the intermediate states to guide the search for a solution at each successive level. Figure 6 shows a hierarchical solution to the three-disk Tower of Hanoi problem. The picture shows the plans produced at each level of abstraction and the mapping between the levels. The upward arrows specify the mapping of the initial and goals states into the abstraction spaces and the downward arrows indicates how an abstract plan guides the search at the next level. At the highest level there is simply a one step plan that moves the largest disk from the first peg to the third peg. This creates two new subproblems at the next level, where the first subproblem is to get to the state where the abstract operator can be applied, and the second subproblem is to reach the goal state. The resulting three-step plan at the second level then creates four subproblems in the base-level, which are solved to produce the final plan.

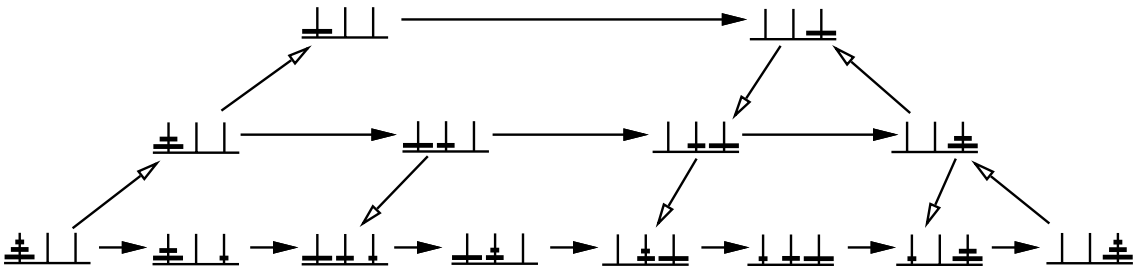

Figure 6: Hierarchical Planning in the Tower of Hanoi

This example illustrates the importance of the ordered monotonicity property, where the work done at each abstraction level is never undone in the process of refining the plan. The solution in the most abstract space produces a plan that moves the largest disk to the goal peg. Since the abstraction hierarchy has the ordered monotonicity property, at the next level only steps for moving the medium disk can be inserted. Thus, the abstraction hierarchy partitions the state space into three smaller spaces and any subproblem must be solved within one of these smaller state spaces. At the final level the hierarchy partitions the state space into nine separate state spaces all of equal size.

## 4.2 Analytical Results

Hierarchical problem solving using the abstractions for the Tower of Hanoi reduces the size of the search space from exponential to linear in the length of the solution. In the original problem the size of the search space is $O(b^l)$, where $b$ is the branching factor, and $l$ is the length of the solution. The abstraction hierarchy divides up the problem into $l$ subproblems of equal size that can be solved serially. The abstraction hierarchy partitions a problem such that each disk is placed in a separate abstraction level and the levels are ordered such that any disk smaller than the disk at the given level will be ignored. Each step in the final solution will correspond to a subproblem in hierarchical problem solving, so the number

of subproblems is $l$. Using an admissible search procedure (i.e., one that is guaranteed to produced the shortest solution), each subproblem will be solved in one step. Since each disk can be moved from one of two places, the branching factor is two. Thus, the size of each subproblem is $2^1 = 2$, so the entire search is bounded by $2l$, which is $O(l)$. Consequently, hierarchical problem solving reduces the search space in this domain from $O(b^l)$ to $O(l)$.

## 4.3 Empirical Results

This section compares problem solving in the Tower of Hanoi both with and without using the abstractions described in this paper. The abstractions are generated by the ALPINE abstraction learner [Knoblock, 1990a, Knoblock, 1991] and then used in a hierarchical version of PRODIGY [Minton *et al.*, 1989, Carbonell *et al.*, 1991], a means-ends analysis problem solver. To evaluate the abstractions empirically, PRODIGY was run both with and without the abstractions using breadth-first search, depth-first search, and depth-first search with an additional restriction on how the problem is solved. The experiments compare the number of nodes searched and the length of the solutions on problems that range from one to as many as seven disks. The graphs below measure the problem size in terms of the optimal solution length, not the number of disks, since the solution to the 4-disk problem is twice as long as the solution to the 3-disk problem.

Figure 7 compares hierarchical and nonhierarchical problem solving using breadth-first search. As the analytical results predict, the use of abstraction with breadth-first search produces an exponential reduction in the amount of search. The results are plotted with the problem size, which is the optimal solution length, along the x-axis and the number of nodes searched along the y-axis. With abstraction the search is linear in the problem size and without abstraction the system cannot solve problems with more than 3 disks. (The 4-disk problem could not be solved in 100,000 nodes.) The dashed line shows the optimal search, which is the number of nodes that would be searched if the system made the correct choice at every point. Both with and without abstraction, the breadth-first search produces the optimal (shortest) solutions.

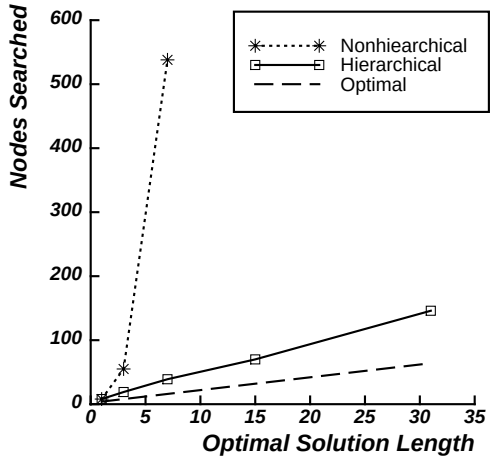


Figure 7: Comparison using Breadth-First Search

In most interesting domains it is impractical to produce optimal solutions to problems. Thus, an interesting question arises as to the effect abstraction has on search when a non-admissible search procedure is used. Figure 8 compares the nodes searched and the solution lengths for problem solving with and without abstraction using depth-first search. As shown in the graphs, problem solving with abstraction produced fewer nodes and shorter solutions than problem solving without abstraction, but the differences are small. This can be attributed to the fact that simply using a depth-first search neither configuration is performing much search. When they make a mistake they simply plow forward adding steps to undo their mistakes. Problem solving with abstraction performed better because the abstraction provides some guidance on which goals to work on first and thus produces shorter solutions.

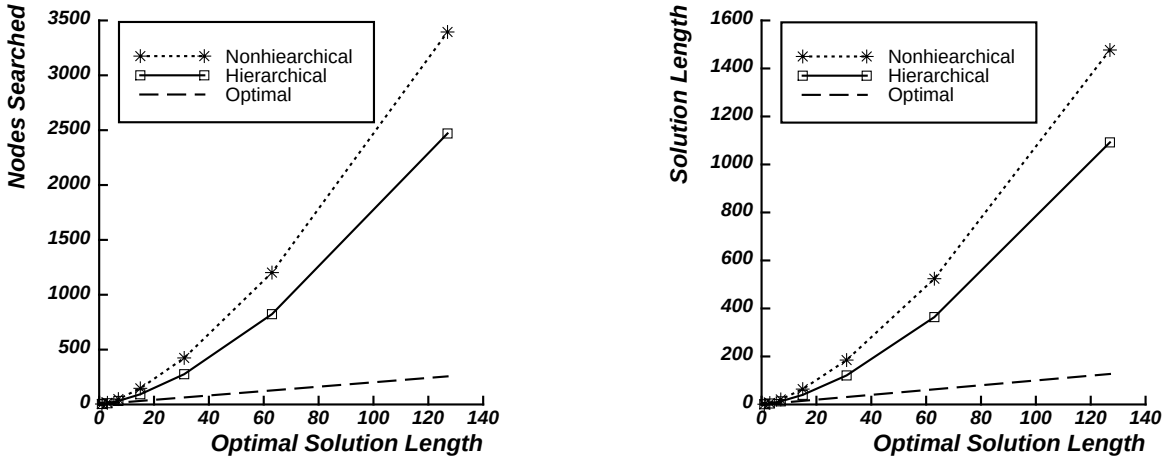


Figure 8: Comparison using Depth-First Search

The small difference between the hierarchical and nonhierarchical depth-first search is largely due to the fact that the problem can be solved with relatively little backtracking. If we impose some additional structure on the domain the difference will be much greater. Consider a variation of the Tower of Hanoi problem that has the additional restriction that no disk can be moved twice in a row [Amarel, 1984, VanLehn, 1989]. This constrains the problem considerably since the suboptimal plans in the previous graph were caused by moving disks to the wrong peg. Figure 9 compares the nodes searched and the solution lengths for the two configurations with this additional restriction on the domain. This small amount of additional structure enables the abstract problem solver to produce the optimal solution in linear time, but has a much smaller effect on the nonhierarchical version.

# 5 Related Work

The Tower of Hanoi has been studied extensively in both artificial intelligence and psychology. This section reviews only the most closely related work, which includes both reformulating the puzzle and automatically generating abstractions.

The Tower of Hanoi puzzle has been discussed in a number of papers on reformulation. Korf [1980] was the first to identify the reformulation, which the algorithm in this paper generates, where the problem is simplified by successively removing the smallest disk. Korf's

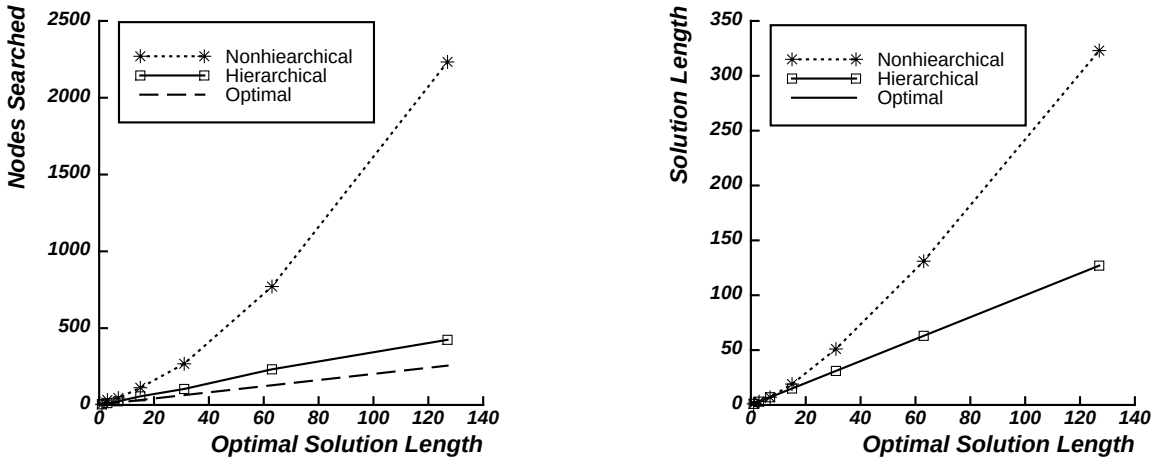


Figure 9: Comparison using Depth-First Search with Restriction

paper presents a language for representing reformulations, but does not provide a technique for finding them. Amarel [1984] describes a variety of reformulations that can be generated using production and reduction schemas. Benjamin et al. [1990] presents an approach to reformulating the problem by combining disks into macro objects to simplify the problem. Thus, the three-disk problem can be transformed into the two-disk problem by combining the two smallest disks. This abstraction is analogous to the one described in this paper, but it is generated by forming aggregate objects instead of ignoring details.

The work on difference ordering in GPS [Ernst, 1969, Eavarone, 1969, Ernst and Goldstein, 1982] is closely related to the algorithm for generating abstractions described in this paper. Their work focuses on producing a triangular table of connections for solving a problem in GPS. The construction of a triangular table is based on the interactions of the effects of operators, while the construction of abstraction hierarchies in this paper is based on both the interactions of the effects and preconditions of the operators. The techniques for producing good difference orders for GPS is only able to identify the disks as good differences, but cannot produce a useful ordering of the disks. For example, [Eavarone, 1969] presents a program that produces 24 triangular tables of connections for the four-disk problem without any preference for which table provides the best difference ordering.

Christensen [1990] describes a system called PABLO that automatically generates abstractions for this domain. His system partially evaluates the operators by back-propagating the goals through the operators. PABLO uses this result to determine the number of steps required to achieve each of the given goals. The problem solver solves a problem by focusing at each point on the part of the problem that requires the greatest number of steps. His approach decides on what to work on based on the number of steps required to achieve a goal, while the approach presented in this paper focuses on the parts of the problem that can be solved and left unchanged while the remaining parts of the problem are solved.

# 6 Discussion

This paper described a technique for generating abstractions of the Tower of Hanoi and then explored the utility of these abstractions in an actual problem solver. The empirical analysis shows that with breadth-first search the use of abstraction produced an exponential reduction in search, but with depth-first search the use of abstraction produced a much smaller reduction. However, by imposing an additional constraint on the problem that increased the amount of search required to solve a problem, the difference between hierarchical and nonhierarchical problem solving was much larger.

There are several general conclusions that one can draw from the experiments. First, the degree to which abstraction reduces search depends on the portion of the base-level search space that is explored. Thus, the more backtracking in a problem, the more benefit provided by the use of abstraction. Second, with a nonadmissible search procedure the use of abstraction will tend to produce shorter solutions since the abstractions focus the problem solver on the parts of the problem that should be solved first.

These experiments also point out several ideas that might be used to improve the effectiveness of hierarchical problem solving. First, performing an admissible search, such as depth-first iterative deepening, in an abstract space may save time and produce better solutions. In the Tower of Hanoi, the quality of the abstract solution has a direct impact on the time required and the quality of the final solution. Second, imposing additional structure on a domain either through restrictions on the operators or added search control knowledge may improve the effectiveness of the abstractions.

# Acknowledgements

I am grateful to Oren Etzioni and Manuela Veloso for their comments on drafts of this paper.

# References

[Amarel, 1984] Saul Amarel. Expert behaviour and problem representations. In *Artificial and Human Intelligence*, pages 1–41. North-Holland, New York, NY, 1984.

[Benjamin *et al.*, 1990] Paul Benjamin, Leo Dorst, Indur Mandhyan, and Madeleine Rosar. An introduction to the decomposition of task representations in autonomous systems. In D. Paul Benjamin, editor, *Change of Representation and Inductive Bias*, pages 125–146. Kluwer, Boston, MA, 1990.

[Carbonell *et al.*, 1991] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[Christensen, 1990] Jens Christensen. A hierarchical planner that creates its own hierarchies. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1004–1009, Boston, MA, 1990.

[Eavarone, 1969] Daniel S. Eavarone. A program that generates difference orderings for GPS. Technical Report SRC-69-6, Systems Research Center, Case Western Reserve University, 1969.

[Ernst and Goldstein, 1982] George W. Ernst and Michael M. Goldstein. Mechanical discovery of classes of problem-solving strategies. *Journal of the Association for Computing Machinery*, 29(1):1–23, 1982.

[Ernst, 1969] George W. Ernst. Sufficient conditions for the success of GPS. *Journal of the Association for Computing Machinery*, 16(4):517–533, 1969.

[Knoblock, 1990a] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

[Knoblock, 1990b] Craig A. Knoblock. A theory of abstraction for hierarchical planning. In D. Paul Benjamin, editor, *Change of Representation and Inductive Bias*, pages 81–104. Kluwer, Boston, MA, 1990.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Available as Technical Report CMU-CS-91-120.

[Korf, 1980] Richard E. Korf. Toward a model of representation changes. *Artificial Intelligence*, 14:41–78, 1980.

[Minton *et al.*, 1989] Steven Minton, Craig A. Knoblock, Daniel R. Kuokka, Yolanda Gil, Robert L. Joseph, and Jaime G. Carbonell. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University, 1989.

[Nilsson, 1971] Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, NY, 1971.

[VanLehn, 1989] Kurt VanLehn. Discovering problem solving strategies: What humans do and machines don't (yet). In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 215–217, Ithaca, NY, 1989.