Searching with Pattern Databases

Joseph C. Culberson and Jonathan Schaeffer

Department of Computing Science, University of Alberta,
Edmonton, Alberta, Canada, T6G 2H1.

Abstract. The efficiency of A* searching depends on the quality of the lower bound estimates of the solution cost. Pattern databases enumerate all possible subgoals required by any solution, subject to constraints on the subgoal size. Each subgoal in the database provides a tight lower bound on the cost of achieving it. For a given state in the search space, all possible subgoals are looked up, with the maximum cost over all lookups being the lower bound. For sliding tile puzzles, the database enumerates all possible patterns containing N tiles and, for each one, contains a lower bound on the distance to correctly move all N tiles into their correct final location. For the 15-Puzzle, iterative-deepening A* with pattern databases (N=8) reduces the total number of nodes searched on a standard problem set of 100 positions by over 1000-fold.

1 Introduction

The A* search algorithm for single-agent search is of fundamental importance in artificial intelligence. Improvements to the search efficiency can take the range from general algorithm enhancements (application independent) to problem-specific heuristics (application dependent):

- General A* search improvements that are applicable to a wide class of problems. For example, iterative deepening can be used to reduce storage requirements [6].
- General search space properties. Many search domains can be represented as directed graphs rather than as trees. The removal of duplicate nodes from the search can result in potentially large savings [14, 16].
- Branch selection. Given a search tree node with several descendants, search efficiency can be influenced by the order in which the descendants are considered [14]. Although the idea of considering the branch most likely to succeed first is a general principle, the techniques used to make that decision are often application-dependent.
- Symmetry reduction. Many problems have inherent symmetries which can be removed. Recognizing and eliminating the symmetries can have an enormous impact on the search space, but it is an application-dependent property.
- Solution databases. In many problems, the states near to the goal nodes can be pre-computed by a backwards search. In two-player games, such as chess and checkers, the backwards searches are saved in endgame databases

and are used to stop the search early, improving search efficiency and accuracy [15]. Recently, in single-agent search new bidirectional search methods are creating so-called perimeters around that goal node and using that to improve lower bound cost estimates [10].

- Problem-specific properties. Problem domains may have constraints that can preclude parts of the search space. For example, although the 15-Puzzle has $16! \approx 10^{13}$ positions, only one half of them can be reached from the goal (a parity test can detect this [5]).

Completely general, application-independent search enhancements are of great interest but are rare, while problem-specific tricks are plentiful and usually not of general interest. More often, an idea is presented that has applicability over a class of problems (has partial problem independence) but needs problem-specific information (has partial problem dependence). Often, the more problem-specific the search enhancement, the greater the search savings that are possible.

This paper introduces *pattern databases* as a new approach for enhancing single-agent search. Planning involves deciding on a series of subgoals that are directed towards the solution [7]. Typically, extensive application-dependent knowledge is required to make the appropriate subgoal choices. Instead, we adopt a "brute force" approach by taking a problem and enumerating all possible subgoals (partial solutions) that satisfy some constraints. At each state in the search, the program can compare its progress towards achieving all possible valid subgoals, and make its next decision based on some or all of this information. We call the set of subgoals a pattern database, because each subgoal is represented as a pattern that can be matched to a search state.

In this paper, pattern databases are applied to permutation problems (although, as shown in Section 6, it generalizes to a wider class of problems). In a *permutation problem*, we have a set of operators that convert one permutation into another. We start with an initial permutation and a specified goal(s). The object is to convert the initial permutation into the goal using the operators. In the optimization version, the object is to minimize the number of applications of the operator during the conversion.

The 15-Puzzle, Rubik's Cube, and other similar combinatorial puzzles are examples of permutation problems. A human-like approach to solving such problems non-optimally is to move some of the elements into their correct locations, thereby reducing the complexity of the remaining problem to be solved. This type of "divide-and-conquer" behavior is commonplace in human problem solving. A similar approach is also evident in human play in many multi-player games, for example chess, where the player establishes a plan consisting of a series of intermediate goals. Although the sequence of goals may be non-optimal, the end result is all that matters.

We adapt this idea to the problem of finding optimal solution paths in single-agent search by noting that knowing the minimum distance from a permutation to the nearest other permutation partially matching the goal is a lower bound on reaching the goal. To define a *pattern database* we designate N elements of the goal permutation. Given an arbitrary initial permutation, the locations of