Using Pattern Databases to Find Macro Operators

István T. Hernádvölgyi
University of Ottawa
School of Information Technology & Engineering
Ottawa, Ontario, K1N 6N5, Canada
Email: istvan@site.uottawa.ca

In this work we employ heuristic search to obtain *macro operators* for spaces defined in our production system. A macro operator is a sequence of original operators which reaches a subgoal from a state without search. A *macro table* has operators for each subgoal. Korf (Korf 1985) used macro operators to find suboptimal solutions for the Rubik's Cube and the 15-Puzzle. While the paths found by the macro method are not guaranteed to be optimal, once the macro table is calculated the search effort is negligible. Traditionally macro operators were found by uninformed search methods, because there were no obvious heuristics. We have devised a simple notation, PSVN (Hernádvölgyi & Holte 1999), to represent state spaces. In PSVN, states are vectors of labels and the operators are simple rewriting rules. For this notation we invented a technique to automatically generate admissible and monotonic heuristics to guide the A* family of algorithms. We apply a simple transformation – *domain abstraction* – on the description of the original space to obtain the abstract space where the distance between two states in the original space is never shorter than the distance between their images in the abstract space. The heuristic values are the lengths of shortest paths in the abstract space. We calculate the distance between the image of the goal state and the rest of the abstract states and store them in a look-up table indexed by the abstract states. This look-up table is also called a pattern database and the method was first used by Culberson and Schaeffer to solve the 15-Puzzle (Culberson & Schaeffer 1994). Korf used pattern databases to solve random instances of the Rubik's Cube for the first time. So far pattern databases have only been used to obtain shortest paths. In this work we use them to find macro operators.

A macro operator reaches a subgoal state without search. To solve for a goal state, each macro brings one (*or a few*) of the labels in the vector representing the state to the index where they occur in the goal state. Subsequent application of the macros in the order of the subgoals fixes all labels and the goal state is reached. The subgoals are patterns where labels at specific indices are identical to the labels of the goal state at those indices. The first subgoal is to fix the label at index i_1, the next subgoal is to fix the label at index i_2 such that the label at index i_1 remains intact and the last subgoal is to fix the last label leaving already fixed labels undisturbed. It is very simple to describe subgoals in PSVN. We introduce a special label which represents "don't care". If our current subgoal is to move the label from index i_x to index i_y while leaving the labels at indices i_1 to i_k intact, then the labels at the other indices can be relabeled to the "don't care" label and the macro operator is a (shortest) path from the relabeled states where i_x and i_y are out of order to the one where they are in order. The labels i_1 to i_k mentioned above are the ones to be left intact. The abstract space is derived by a domain map which renders some of these labels identical. By applying this method we were able to build optimal macro-tables for the Rubik's Cube, where the longest optimal macros are composed of 13 moves. The average branching factor of our IDA* search tree was 14.35.

Establishing the order of subgoals for permutation groups can be done automatically, but for a general PSVN search space it may require the user's intervention. It may also take some experimentation to find a pattern database which is not too large but provides good heuristic values. The size of the pattern database is determined by the domain abstraction, which is simple to encode in our notation. Korf's partial-match with bi-directional search (Korf 1985) is expected to expand $b^{d/2}$ states, where b is the branching factor and d is the optimal macro's length, but it must store the entire frontier of the search trees. Our pattern databases with modest size (1M entries) provide average heuristic values which are greater or equal to $d/2$ for the macros of the Rubik's Cube resulting in less than $b^{d/2}$ states expanded. We use IDA*, hence no memory additional to the database is needed.

I would like to thank Dr. Jonathan Schaeffer and Dr. Robert Holte for their encouragement and guidance to complete this work.

References

Culberson, J. C., and Schaeffer, J. 1994. Efficiently searching the 15-puzzle. Technical report, Department of Computer Science, University of Alberta.

Hernádvölgyi, I. T., and Holte, R. C. 1999. PSVN: A vector representation for production systems. Technical Report TR-99-04, School of Information Technology and Engineering, University of Ottawa.

Korf, R. E. 1985. Macro operators: A weak method for learning. *Artificial Intelligence* 26:35–77.