A Pattern-Weight Formulation of Search Knowledge

Robert Levinson
Gil Fuchs

UCSC-CRL-94-10
supersedes UCSC-CRL-89-22 and UCSC-CRL-91-15
February 1994

Board of Studies in Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064
(408)459-2087
ARPANET:levinson@cse.ucsc.edu
UUCP:ucbvax!ucsc!cse!levinson

ABSTRACT

Pattern-weight pairs (pws) are a new form of search and planning knowledge. Pws are predicates over states coupled with a least upper bound on the distance from any state satisfying that predicate to any goal state. The relationship of pws to more traditional forms of search knowledge is explored with emphasis on macros and subgoals. It is shown how pws may be used to overcome some of the difficulties associated with macro-tables and lead to shorter solution paths without replanning. An algorithm is given for converting a macro-table to a more powerful pw set. Superiority over the Squeeze algorithm is demonstrated. It is also shown how pws provide a mechanism for achieving dynamic subgoaling through the combination of knowledge from multiple alternative subgoal sequences. The flexibility and execution time reasoning provided by pws may have significant use in reactive domains. The main cost associated with pws is the cost of applying them at execution time. An associative retrieval algorithm is given that expedites this matching-evaluation process.

Keywords: heuristic search, knowledge representation, macro-operators, feature construction, pattern retrieval, reactive planning, subgoals, tile puzzles

> When a philosopher invents a new approach to reality, he promptly finds that his predecessors saw something as a unit which he can subdivide, or that they accepted distinctions which his system can name as unities. The universe would appear to be something like a cheese; it can be sliced in an infinite number of ways–and when one has chosen his own pattern of slicing, he finds that other['s] cuts fall at the wrong places.
>
> –Kenneth Burke [17]

1 Motivations

Korf [19, 20, 21] has done an excellent job of classifying and describing the traditional forms of search knowledge. This was facilitated by the recognition that planning with perfect information can be viewed as heuristic search. The difference in nomenclature refers to the types of search knowledge being used. For planning, subgoals, macro-operators and abstraction spaces are used; for heuristic search, heuristic evaluation is used. In examining each of these knowledge forms, one wonders about their interaction in problem-solving and whether there is a more basic form that captures the essence of each of these. It is these questions that this paper begins to address.

The pattern-weight formulation that will be used to help answer these questions was developed as part of a project to develop methods by which intelligent systems can improve their search efficiency and accuracy through experience, in particular, by using pattern formation and associative recall [27, 28, 30]. We will purposely not be focusing on the learning issues in this paper, but instead will concentrate on the relationship of this form of search knowledge to more traditional forms and to their interaction.

In particular, it will be shown that *pattern-weight pairs* (pws) due to their low granularity provide the following advantages over macro-operators and subgoals:

- They can lead to shorter solution paths.
- Can support management of multiple alternative subgoal sequences.
- Are more amenable to reactive and execution time planning.

The main disadvantage of pws over traditional forms is that more computation is required to make full use of them. It will be shown how this additional computation may be managed efficiently.

The goal of the pattern-weight approach is to provide a more uniform problem-solving mechanism at a lower-level of granularity than in other approaches [19, 22, 38]: no distinctions will be made between subgoals and non-subgoals, nor is knowledge about actions explicitly stored. Instead, we shall take the view that if states can be evaluated properly (by determining their distance from a goal state) that the actions will take care of themselves. *It is hoped that the uniformity provided by pws will lead to more efficient and flexible problem-solving schemes and also be more consistent with current cognitive models that emphasize pattern matching over symbolic processing* [44].

2 Preliminaries

We will abide largely by the definitions and understanding set out by Korf [21]. We shall define a problem space (or state space search problem) as a set of states (the "state space") and operators where operators are partial mappings from states to states. Problem instances are composed of a problem space with an initial state and a goal where a goal is a predicate over states. A solution to a problem instance is a sequence of operators that map the initial state to a goal state where a goal state is any state that satisfies the goal predicate.

We will assume that the goal remains fixed over the problem instances for a particular problem space, and thus may be considered a third part of the problem state definition, along with states and operators. For many practical problems this assumption is realistic, since usually either the goal state or the initial state remains fixed. In the latter case we can solve the corresponding problem with operators going in the opposite direction.

By using a fixed goal state an origin is established for computing distances. However, this restriction may be removed [29].

Now we define a pattern as a predicate over states. Thus, the goal predicate is a pattern. We will say that a pattern P occurs in a state S if P(S) is true. For simplicity of notation we will use the symbol for a state ("S" in the previous sentence) to refer interchangeably to the state itself or to the pattern that is true in that state and in no others. Typically, patterns for a particular problem space will be represented in some fixed representation language and we will only be interested in the patterns expressible in that language. Often, the pattern representation language will be the same as the state representation language except that patterns can be partial state descriptions. Patterns are partially ordered by the relation more-general-than. Pattern A is more general than pattern B, if for all states S, if B(S) is true, A(S) is true. What we call patterns have also been called features [43], equivalence classes and schema [14] in the literature. We prefer the term "pattern" as they are most useful when they occur regularly. We will see that they are also related to what have been called abstract states and when coupled with a weight serve much the same purpose as subgoals (Section 7).

Search control knowledge can be maintained by storing a subset of the patterns that occur in the state space. With each pattern P is assigned a weight, w(P), which is to be a least upper bound on the shortest distance from any state in which P is a subpattern to any goal state. Each state is evaluated as the minimum of the weights of the patterns that occur in it. Thus the evaluation of a state S, e(S), is a least upper bound on the shortest distance from the state to any goal state. Since if pattern A is more-general-than pattern B then $w(B) \leq w(A)$, e(S) is simply the minimum of the weights of S's most specific subpatterns. In general, $e(S) \leq w(S)$ and when no two state patterns hold the more-general-than relation to each other, e(S)=w(S). Here evaluations are pessimistic (being least upper bounds) whereas those that traditionally go with A* and other search algorithms are optimistic (lower bounds).

Examples in this paper will be taken from the $n \times n$ tile puzzles. The traditional small tile puzzle [15], called an "eight puzzle," is a 3x3 matrix of squares. Eight of the squares are occupied by sliding tiles which are numbered 1-8. The ninth square is empty, allowing the other tiles to be moved. The object is to arrange the numbers in a predefined pattern from a random starting point. The puzzle can be extended to much larger sizes. The largest commonly used size is 6x6, for a puzzle with 35 tiles and one empty square. The general procedure for solving the puzzles is the same at any size, but the problem of finding optimal solutions (those which require a minimum number of moves) is NP-hard [40].

Patterns for tile puzzles will be states that may leave zero or more tile positions unspecified. A major theme of the paper is that useful patterns can be derived from macrotables [19]. But before proceeding further it is worthwhile to consider how patterns (with weights as least-upper bounds) might enhance traditional heuristic functions. To determine this we have done an analysis of the entire eight puzzle state space, considering all partial state descriptions as patterns. For each pattern we consider all solvbable full states that match a given pattern to determine that pattern's statistical profile. Table 1 illustrates the average and range of patterns of a given size where size is defined in terms of number of tiles specified. Tables 2 and 3 report results of an experiment in which the least upperbounds of patterns of size 1 and 2 are summed and normalized (as opposed to taking the minimum as we do in the remainder of the paper) to produce a heuristic that is more efficient and/or more accurate than Manhattan Distance. Victories are said to occur when an optimal solution is found and fewer states are expanded than with the others.

Size of Pattern	Population	Ave.	Min.	Max
6	4,896,814	21.45	19.37	23.45
5	1,905,111	21.50	17.15	25.14
4	381,024	21.50	14.55	26.67
3	42,336	21.50	11.62	28.03
2	2,592	21.50	8.27	29.27
1	81	21.50	4.35	29.95
0	1	21.50	0.00	30.00

Table 2.1: Statistical profile of patterns, grouped by size

	Manhattan	all size 1 pats	all size 2 pats
average solution length	21.50	21.84	21.71
ave. no. states expanded	948.20	451.60	227.40
percentage optimality	100.00	86.10	90.00
percentage victories	7.80	5.60	86.60

Table 2.2: Cost/benefit comparison between A* heuristics

	Manhattan	all size 1 pats	all size 2 pats
average solution length	21.50	21.84	21.71
ave. no. states expanded	948.20	451.60	794.90
percentage optimality	100.00	86.10	100.00
percentage victories	4.60	67.40	28.00

Table 2.3: Cost/benefit comparison (with admissable size-2 heuristic)

A planning strategy that we shall only touch on here is the use of abstraction in hierarchical planning [19, 18]. In a common use of this strategy, certain states are collected together to form "abstract states." Interestingly, here too, a direct mapping to the pattern-weight formulation is possible. As there is always a 1-1 correspondence between patterns and subsets of the state space, the terms pattern and abstract state may be viewed similarly: traveling from one abstract state to the next is equivalent to traveling from some state in which one pattern is true, to another in which another pattern is true. Traveling within an abstract state means to traverse states in which one particular pattern remains true. Due to efficiency considerations, in this framework, patterns must have simple descriptions to facilitate matching and not be complicated disjunctions of individual states as abstract states may be in general.

3 Depth-First Hill-Climbing

Much of the following discussion will assume the use of pattern-weight sets (pws) with the standard depth-first hillclimbing algorithm (DFHC) that always does only 1-ply search and greedily moves to the adjacent state with best evaluation. We will be constructing pattern-weight sets that insure that the algorithm will terminate and return a solution path if a solution exists.

```
DFHC:
Begin
    Let current-state = initial-state.
    Let solution-path = empty-sequence.
    While not goalp(current-state) do
        Let S = the set of 3-tuples (s,o,d) where s is a legal state
            reachable from current-state using operator o and receives
            an evaluation of d.
        Let (s*,o*,d*) be that tuple with smallest d value.
        Let current-state = s*.
        Let solution-path = solution-path followed by o*.
    Return (solution-path).
End
```

4 Macro-Operators

Macro-operators (macros) are sequences of primitive operators. Such sequences can naturally be represented as sets of pattern-weights. Sequences can be represented by following the preconditions of the macro through the successive operator applications of the macro and giving decreasing weights to each successive pattern. DFHC will then naturally reproduce the operator sequence at execution time (though we will see that sometimes it may not complete the sequence, but instead opt for something better!) since after each operator application there will be a state with the next pattern (and, hence, lower evaluation).

For problems with serially decomposable subgoals (i.e. there exists some ordering of subgoals for which the effect of each operator on each subgoal depends only on that subgoal and previous subgoals in the ordering), Korf has shown how a table of macros can be constructed that guarantee solution paths from all initial states. Later we will use such a set of macros to produce a set of pws that, when coupled with DFHC, is guaranteed to produce solution paths from all initial states that are shorter on the average than those produced by the macros. The technique is best illustrated by an example:

EXAMPLE 1:

A macro-table is a completely general solution to the tile puzzle problem, providing a clear, unambiguous solution for every solvable puzzle configuration [19] For instance, a macro-table for a 2x2 puzzle – three tiles, one empty square – might look like the one in Table 4.

Position	*Tile*			
	0	1	2	3
0	*	*	*	*
1	r	*	*	*
2	dr	lurd	*	*
3	d	uldr	*	*

0	1
3	2

Table 4.1: A macro-table for solving the 2x2 tile puzzle.

To read this macro-table, the system first locates tile number 0 (the empty space.) The entries in column number 0 corresponding to the empty tile's location indicates the sequence of moves necessary to properly position the space given it is at positions 0,1,2 and 3 respectively.

Given that the 0 subgoal has been solved, we only need to specify macros for tile 1 when it is in positions 2 and 3. These macros not only move tile 1 to its destination, but also return the empty square to its proper place. In general, a macro must ensure that all previously positioned tiles are returned to their proper locations (though they may be moved during the macro's operation.) If this condition is met, then when the last macro is executed, the puzzle will be solved – the last tile, and all previous ones, will be in their proper positions. The last two columns are always empty in all tile puzzle macro tables, no matter what the size of the puzzle. This is due to a property of tile puzzles [15]: if any two non-blank tiles in a solved board are swapped the puzzle cannot be solved by any legal manipulation.

4.1 Weaknesses in the Macro-table Technique

Despite the obvious strengths of the macro-table technique in terms of space, correctness and execution time efficiency there are some weaknesses [3]. The most serious weakness is that the solution paths produced by macro-tables are often far from optimal (as depicted in Table 4.2). Tables 4.2 through Table 5.3 are a running example based on states selected for illustrative purposes. For a more statistically representative set of examples see Tables A.1 and A.2 in the Appendix.

EXAMPLE 2: Non-optimality of macro solutions.

		Solution Length	
Size	Initial state	Macro(Mt)	Optimal
3x3	024813567	50	14
3x3	847035126	50	22
3x3	013724658	24	16
3x3	782134560	31	11
4x4	c953086b41f2ad7e	131	33
4x4	c130492a8e7fb5d6	132	22
4x4	851426a739b0fdec	120	20
4x4	b3c7651280a4f9de	142	32

Table 4.2: Demonstration of the non-optimality of macro-tables

For higher-order puzzles, the problem gets worse. (The letters a-f represent tiles numbered 10-16.) (these optimal solutions were generated using A* with a Manhattan-distance heuristic.)

That macro-tables should lead to inefficient solution paths is not surprising. This is almost always the case when a general heuristic strategy is applied to specific problem instances. The difficulties are that macros operate at a high-level of granularity that prevents adjustment while a macro is executing and that a macro-table depends on a particular subgoal sequence (order in which state variables are to be solved) that may be inconvenient for a given problem instance. Pws can be used to address these difficulties by exploiting *macro-crossover* and *subgoal sequence crossover*, respectively. Further, pws allow the knowledge from multiple macro-tables to be usefully exploited. The following sections bring out these issues in detail.

Importance of Subgoal Ordering

The ordering of subgoals can have a profound effect on the efficiency of macro-based solutions. This is due almost entirely to the side effects resulting from the positioning of each tile. Tiles whose positions have not yet been fixed are shuffled around to undefined places while a subgoal is being reached. Ordering the subgoals differently will change the side effects, sometimes placing the remaining tiles closer to their final positions and reducing the number of moves required to solve the puzzle. It should be clear that different subgoal sequences will result in very different macro-tables.

Name	Sequence
top	0 1 2 3 4 5 6 7 8
right	0 3 4 5 6 7 8 1 2
bottom	0 5 6 7 8 1 2 3 4
left	0 7 8 1 2 3 4 5 6
interleaved	0 2 4 6 8 1 3 5 7

Initial state	Top(Mt)	Right(Mr)	Bottom(Mb)	Left(Ml)	Interleaved(Mi)
024813567	50	16	40	16	66
847035126	50	42	38	32	72
013724658	24	24	24	50	40
782134560	31	17	43	29	21

Table 4.3: Subgoal sequences and the importance of ordering

The effect of changing the ordering of subgoals can be seen in Table 4.3. Clearly, the ordering of subgoals makes a difference. The trouble is that no single sequence is consistently better than all the others. There is no known easy way to tell which sequence will be best for an arbitrary tile configuration, other than to simply try several sequences and compare the results.

Macro-crossover and Squeeze

Because of the high-level granularity of macro-based solutions, all the already-positioned tiles must be returned to their proper places before the next tile can be positioned. That is, assuming the ascending subgoal sequence (as above), the positioning of tile 3 must end

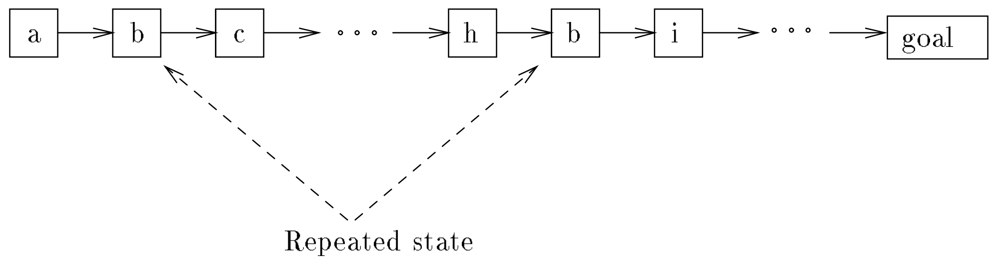

Figure 4.1: Repeated states in a macro solution.

with tiles 0, 1, 2, and 3 in place before tile 4 can be evaluated. Tile 0 is obviously always the last to be returned to its rest position. Because of this restriction, duplicate states often occur. For example, suppose that the macro for a subgoal is “lurrd.” If the macro for the next subgoal begins with a “u,” two moves have been wasted doing essentially nothing.

The “Squeeze” algorithm described briefly in [39] removes such inefficiency by removing the paths (cycles) between two identical states and hence makes some improvements to overall solution lengths (see Table 4). We will show in the next section that pws obviate the need for Squeeze.

Size	Initial state	Optimal	Macro(Mt)	Squeeze
3x3	024813567	14	50	46
3x3	847035126	22	50	48
3x3	013724658	16	24	16
3x3	782134560	11	31	29
4x4	c953086b41f2ad7e	33	131	111
4x4	c130492a8e7fb5d6	22	132	124
4x4	851426a739b0fdec	20	120	112
4x4	b3c7651280a4f9de	32	142	130

Table 4.4: The effects of Squeeze on macro length

5 Patterns

Recall that a pattern is defined as a predicate over states.

5.1 Patterns and Subgoals

Subgoals are intermediate goals to be achieved before reaching a final goal. Korf [21] formally defines a subgoal as “a set of states, with the interpretation that a state is an element of a subgoal set, if and only if it has the properties that satisfy the subgoal.” This is, clearly, equivalent to our definition of pattern. Korf has shown that subgoals are useful even if they need to be undone later since they indicate a direction in which the search or plan should proceed. Well-chosen patterns fulfill precisely this purpose! Korf has also shown that the best subgoals are not necessarily partial-descriptions of a goal-state [19]. Algorithms based only on ordering partial goal states, perform no better than brute force in the worst case - additional knowledge is needed [16].

The difference between patterns and subgoals is that subgoals are totally-ordered in the order in which they are to be achieved. It is this ordering information that give subgoals much of their power [41]. The optimal orderings are those that produce few impasses, where impasses are said to occur when a previously solved subgoal must be undone to achieve a current subgoal. Working towards well-ordered subgoals one at a time reduces the cost to find a solution path (by focusing the search) at the cost of finding solution paths of much greater length than optimal. *We shall see that patterns can be used similarly, but with more flexibility. Without imposing a total ordering, when coupled with weights they are used as "signposts of progress."*

5.2 Decoupling macros into pattern-weight pairs

The primary weaknesses of all of the above techniques are that they either treat a state as a whole, in the case of the Squeeze algorithm described above, or treat the board as a series of single tiles in the case of the macro-table technique.

With the pattern-weight technique, the system no longer matches entire boards (as in Squeeze) or looks at the positions of individual tiles. Instead partial board matches, or "pattern matches" are searched for.

A macro-table that guarantees solution of an nxn puzzle from all states [19] can be reformulated as patterns and weights as follows:

1. For each macro, create a pattern out of the preconditions of the macro. Let "key tiles" stand for the tiles and blank that are specified in the preconditions. Then for each successive state the macro goes through, create a new pattern depicting where the key tiles occur in this state. Assign a weight recursively to each pattern equal to the number of remaining operators in the macro plus the maximum weight of the pattern in which the key tiles are all in their correct place (the post-conditions for the original macro). Thus, the subgoals 0, 01, 012, 0123, 01234, 012345, 0123456, 01234567, 012345678 get weights of 62, 50, 40, 26, 18, 4, 0, 0, 0 respectively. For example: Assume a goal-state for the 8-puzzle of:

1	2	3
8	0	4
7	6	5

   where 0 is the blank space.

   The macro for the state in which tiles 0 and 1 are in the correct place and tile 2 is in position 4 is "ldru" The macro reaches a state in which tiles 0,1 and 2 are in their correct place. From this macro we would create the following patterns where "-" may be anything. The weights assigned are shown below:

1	-	-
-	0	2
-	-	-

   44

1	-	-
-	2	0
-	-	-

   43

1	-	0
-	2	-
-	-	-

   42

1	0	-
-	2	-
-	-	-

   41

1	2	-
-	0	-
-	-	-

   40

2. Pattern-weight pairs now stand for "states that contain this pattern as a subpattern are at most a distance equal to the weight from the goal" thus corresponding with our specification of weight.

3. As always, each state's evaluation is the minimum of the weights of the most specific subpatterns that apply to it.
4. This construction insures that DFHC will find a solution path no longer than the weight of the minimum pattern occurring in the initial state. This follows directly from Theorem 1 below.

Theorem 1: When DFHC is applied to the pws generated from a macro-table, given a solvable initial state a solution path is generated. Further, the evaluations of states on the solution path will be monotonically decreasing.

Proof: Each state S is evaluated as the minimum of the pws that apply to it. Call this extreme pattern p(S) with w(p(S)) = e(S). Now if p(s) was a pattern generated from intermediate conditions in a macro sequence, there is some operator application (namely the one to be applied at that point in the macro) that will lead to a state S' with a pattern with weight less than w(p(S)) and thus S''s evaluation e(S') is less than e(S). Now if p(S) came from the terminating condition for a macro M, S must be matched by some pattern P' which is the initial state of some macro in the column immediate to the right of M in the macro-table (and by our construction has weight = e(S)). By executing the operator to be applied at P' a state of lesser evaluation will be reached, arguing as above.

The only state for which the above paragraph does not hold is the goal state since it does not have any macros to the right. But it is at a goal state that DFHC terminates.

The reader may have noticed from the above proof that it is not actually necessary to calculate and store a pw for the terminating conditions of a macro (except for perhaps the goal state) since they will be subsumed by the patterns containing the initial conditions of the macro to the right of them in the table. For example, in Table 7 the second pattern top left need not actually be stored.

EXAMPLE 3:

- 0 - -	0 - - -	- - - 0	- - 0 -	0 - 1 -	0 - - 1	- 0 - 1
5	4	6	5	4	4	3
- 1 - 0	- 1 0 -	0 1 - -	1 - 0 -	1 - - 0	1 0 - -	
2	1	0	3	2	1	

Table 5.1: Pws created from the 2×2 macrotable

The patterns in Table 5.1 were created by using the algorithm given above on the 2x2 macro-table.

The pw reformulation of a macro-table uses storage of the same order (though using more bytes per macro-step) and produces 7-8% shorter solution paths in practice for the 8-puzzle and about 23% shorter paths for the 15-puzzle. (See tables A.1 and A.2 in the Appendix.

Macro-crossover opportunities are not recognizable until execution time since they depend on the identity of those tiles not specified by the macros. It is true for a single macro-table (as specified by Korf), however, that an intermediate pattern P_a of macro A can only satisfy an intermediate pattern, P_b, of macro B if P_a is more-general-than than P_b. This condition is necessary, but not sufficient for macro-crossover. In Section 7, we will be combining the knowledge from two or more macro-tables and then even this condition need not be true.

An advantage of this technique over such algorithms as Joint and LPA* [39] is that no potentially exponential searches of the state space as a whole are required. These techniques apply A* in an attempt to shorten segments of existing solution paths. However, the efficient solution paths produced by pws can be generated at execution time, *no re-planning is necessary.*

5.3 Patterns and Squeeze

After conversion to pws there is no need to run the solutions through a Squeeze routine:

Theorem 2: Solution paths based on a pw version of a macro-table never contain duplicate states.

Proof: This follows directly from Theorem 1 since each state is given exactly one evaluation and the evaluations of states in the solution path are monotonically decreasing.

In fact, *patterns are stronger than Squeeze* (except in rare instances, see next section). Squeeze can only find a way to reduce the solution length when one state in the sequence exactly matches another state further on in the sequence. Patterns, on the other hand, can recognize a state as being further along based on less information. The result (see Table 5.2) is shorter overall solution lengths.

Size	Initial state	Macro(Mt)	Optimal	Squeeze	Pattern(Pt)
3x3	024813567	50	14	46	46
3x3	847035126	50	22	48	48
3x3	013724658	24	16	16	16
3x3	782134560	31	11	29	29
4x4	c953086b41f2ad7e	131	33	111	93
4x4	c130492a8e7fb5d6	132	22	124	68
4x4	851426a739b0fdec	120	20	112	82
4x4	b3c7651280a4f9de	142	32	130	94

Table 5.2: Comparison of different solving strategies

5.4 When Macros or Squeeze May Be Better Than Patterns

Occasionally, for a given problem instance, it is possible to produce a solution path using macros with or without Squeeze that is shorter than that produced by pws. These cases can occur when a macro executed by the macro technique happens to solve more than one subgoal but patterns, by skipping this macro, may miss this opportunity. These anomalies do not occur because of any superiority of the macro knowledge but due to fortune alone. We observe these occurring 3% of the time in the 8-puzzle. For example, see states 348576021, 341568072, 281576043 in the appendix. On the 15-puzzle although Patterns dominate, SQUEEZE comes out ahead about 6% of the time.

In those cases in which the macros must be used to solve each subgoal (not skipping any) the pattern-weight formulation is guaranteed to produce solution paths no longer than those produced by the macros since in the worst case (by Theorem 1) the same path is traversed.

5.5 Patterns and Dynamic Subgoaling

We pointed out earlier (Section 4.1) that macro-tables are restricted in the sense that they are based on exactly one subgoal sequence that may be far from optimal. Ideally, we would like a system that could choose among several alternative subgoal sequences and even correct itself in midstream by taking advantage of *subgoal sequence crossover.* To achieve this with pws, one first generates macrotables for each desired subgoal sequence, converts them to pws and combines these pw sets by taking their union, with repeated patterns receiving their minimum weight. Once the knowledge is in this form, DFHC can be applied as before to produce solutions that are shorter on the average than those based on any single subgoal sequence. (See Figure 2)

Size	Initial state	Macro(Mt)	Optimal	Squeeze	Pattern(Pt)	P5
3x3	024813567	50	14	46	46	14
3x3	847035126	50	22	48	48	32
3x3	013724658	24	16	16	16	16
3x3	782134560	31	11	29	29	11
4x4	c953086b41f2ad7e	131	33	111	93	113
4x4	c130492a8e7fb5d6	132	22	124	68	68
4x4	851426a739b0fdec	120	20	112	82	30
4x4	b3c7651280a4f9de	142	32	130	94	76

Table 5.3: The effect of using pws from multiple subgoal sequences

Here, one sees the biggest advantage in using pws: to mix subgoal sequences. Tables A.1 and A.2 (see Appendix) were produced from solving 10,000 different random instances of the 8-puzzle and 100 different random instances of the 15-puzzle. Each instance was solved once using each of five different macrotables associated with a different subgoal sequence, and then from pattern sets derived from the macros, once with four pattern sets combined and once with five pattern sets combined.

As expected, the average solution lengths using any one of the subgoal sequences were about the same (except for the interleaved sequence) and the pattern formulation of these sequences produced path lengths that were about 9-10% shorter than those formed by the macros for the 8-puzzle, and about 2% better on the 15-puzzle. Compared to SQUEEZE the solutions were 2-3% and 15-16% on the 8-puzzle and 15-puzzle respectively.

When the pattern sets were mixed (P4,P5), solution paths that were nearly 16-17% shorter on the 8-puzzle and 26-27% better on the 15-puzzle on the average than those from the macros. Further, solution paths were about 10% and 1% shorter on the average than those produced by any one pattern set. The explanation of this gain comes from the two reasons cited in the previous paragraph: The system chooses a favorable subgoal sequence to start with and can switch to another sequence if it becomes more favorable. These sequence shifts are recognized by noting from which pw set a new state's evaluation

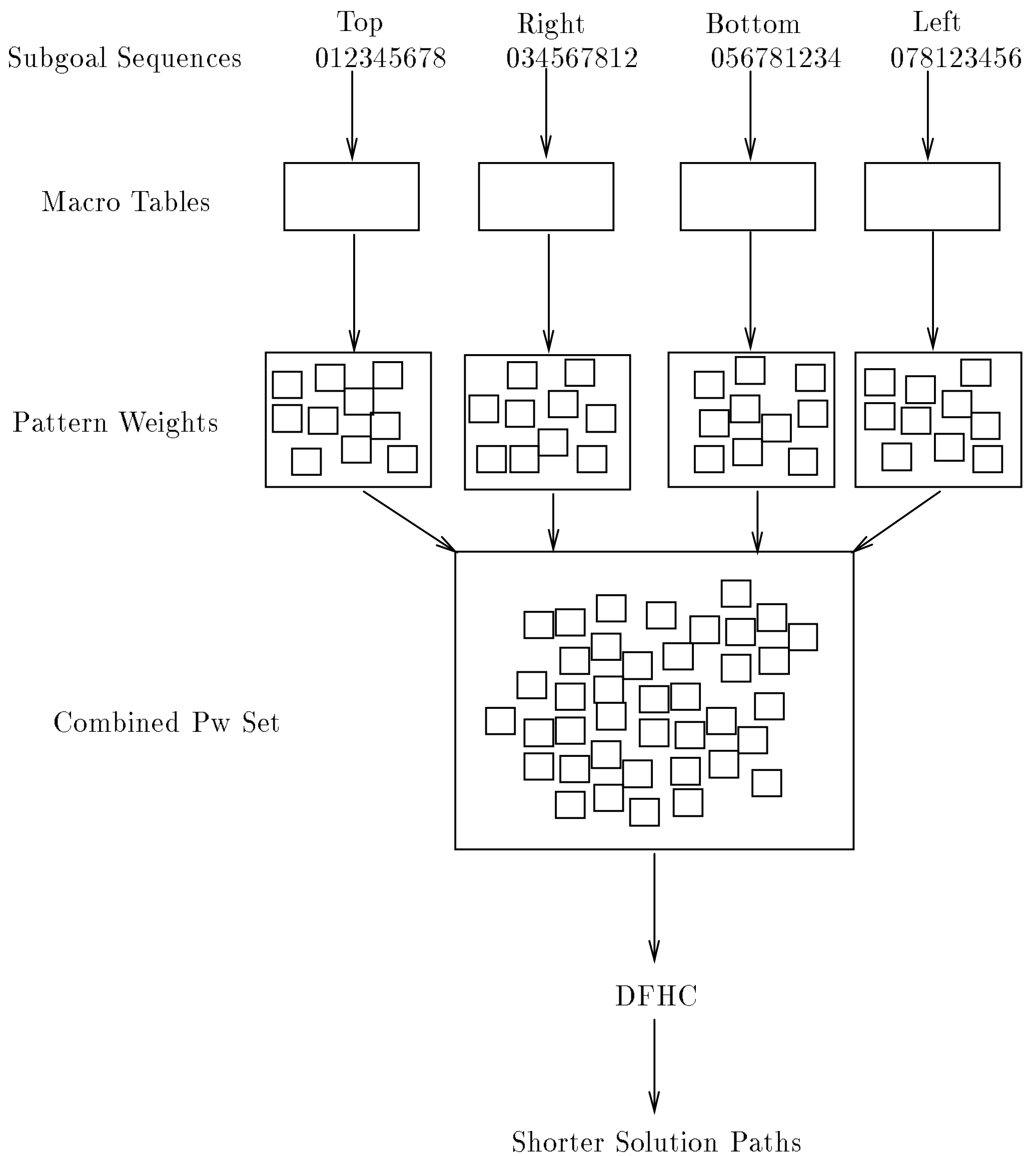

Figure 5.1: Combining macro-table knowledge

is based. Although the system actually switched sequences infrequently, when it did so the gains were significant.

Since P4 and P5 did so well one might wonder why Pm returns solution paths that are about 15% and 9% shorter on the average than P4 or P5 (and 29% and 33% better than the Macros)? The explanation is that while P4 or P5 might choose a good subgoal sequence initially it may not turn out to be the best. The pattern sets for Top, Right, Bottom, and Left and Interleaved are not close enough to generate enough subgoal sequence crossovers to make up for this.

Table 5.3 gives a final example of using the patterns generated from multiple subgoal sequences on solution length. The *P5* column lists those solution lengths. The subgoal sequences are the five subgoal sequences of Table 4.3. The first line represents a state that was solved optimally by P5 and not close to optimally using a macrotable, Squeeze or a single pattern set. An optimal sequence for this state is llurrdldlurdrul. In generating this sequence P5 used patterns from the "bottom" sequence for 9 states and then switched to "right" patterns for the next 4 states and completed using a pattern from "interleave".

For the case of multiple subgoal sequences, theorems analogous to Theorems 1 and 2 (of Section 6) can be proved.

6 Implementing Efficient Pattern-Based State Evaluation

In the previous section it was shown that pattern-weight sets can produce shorter solution paths than from the macros from which they have been derived. The disadvantage of using pattern-weight pairs over macros is that at execution time computation is required at each step to evaluate the states adjacent to the current state and to choose the proper operator. Whereas, while executing a macro little computation is required until the selection of the next macro. However, state evaluation can be done efficiently by using an associative retrieval algorithm that takes advantage of a partial ordering of patterns by more-general-than [6, 10, 24, 25, 34].

Suppose for example that pattern R is known to be a generalization of pattern S. Now once we determine that R is a specialization of a "query" pattern Q, we know that S is also without performing further comparison tests! Similar reasoning applies to negative information: If pattern X is a generalization of pattern Y, then if X is found not to be a generalization of Q, than clearly Y isn't either.

In this database organization all patterns are placed in a partially-ordered hierarchy by the relation more-general-than (See, for example, figure 6.1). Because of transitivity only the immediate predecessor (generalizations) arcs and immediate successor (specialization) arcs need be stored (as in the Hasse diagram of any po-set). In essence, an object is indexed by its predecessors in the ordering and indexes its successors!

Notice that to evaluate a state Q it is sufficient to find where it fits in the partial order (i.e., Q's immediate predecessors and immediate successors). The predecessors of Q in the ordering are its generalizations and its successors are its specializations. Thus the retrieval/insertion operation is essentially the same as an insertion operation. The immediate predecessor and immediate successor sets are found in two consecutive phases. Phase II makes use of the immediate predecessors found in Phase I to find immediate successors. Both phases attempt to use the information in the hierarchy to minimize the number of pattern comparison tests. For state evaluation without insertion only Phase I is necessary.

Ordering the database objects by size produces a topologically sorted list, i.e. a total ordering that embeds the original partial ordering by more-general-than. Since all database objects will be preceded by their predecessors in the list they only will make it to the front of the list if their predecessors have been found to be predecessors of Q. Thus, the proper elimination of comparisons is taking place.

If we actually wish to insert Q into the hierarchy, the IP and IS sets of other objects have to be updated. This is done in Phase III.

Thus, Phase II does not do a comparison test on a database pattern unless it contains each member of IP(Q). Note how the original database objects are being used as screens in Phases I and II. The big savings of this retrieval method comes from the fact that only the immediate successors of Q need to be determined using comparison tests. All other successors (specializations) are determined for free. Since the patterns eliminated in this way are usually the most complex, many expensive tests have been eliminated.

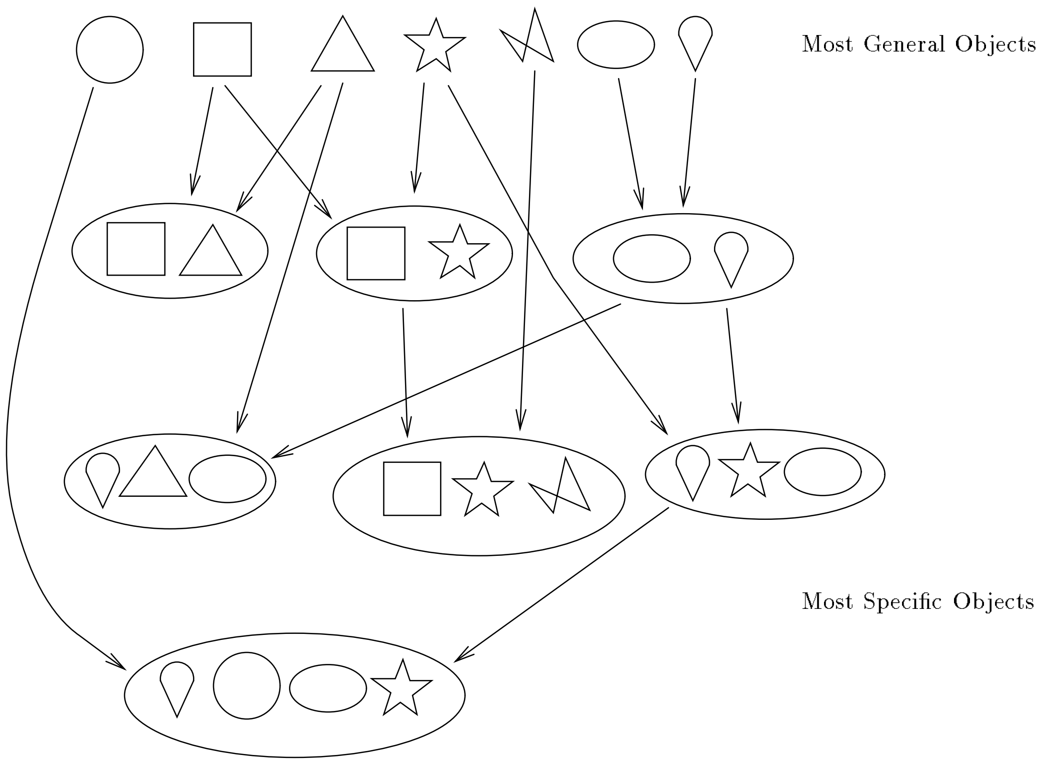

Figure 6.1: A hierarchy of objects ordered by "more-general-than"

Retrieval Phase I: (find IP(Q), the immediate predecessors of Q)

(1) List all patterns from smallest to those of the same size as Q.
(2) S := $\emptyset$.
(3) While there is a member X in the list
    If X is a predecessor of Q (comparison test) then
        S := S $\cup$ {X} − IP(X)
        Remove X from the list.
    Else
        Remove X and all successors of X from the list.
(4) Return S.

The efficiency of this algorithm can be improved by, in addition to the hierarchy, maintaining a linked list of database objects sorted by size. Space for new links is reserved so that the sublists needed by the algorithm may be formed dynamically. Further enhancement can be achieved through incremental updating.

There are other algorithms for insertion of objects into partially-ordered sets, we recommend the one here due to its simplicity and efficiency. *In practice only a small fraction of the database objects need to be compared with Q using comparison tests*: 10 or 20 structures at the most on a database of 680 objects for example. Further, we have seen that as database size grows the increase in retrieval time is sublinear and probably logarithmic [25, 34, 10].

This algorithm has been used for associative retrieval of organic molecules and reactions [24, 25, 45], chess patterns [25, 28] and radio signals [26] in addition to the application

Retrieval Phase II. (find IS(Q), the immediate successors of Q)

```
(5) S := ∅.
(6) Y := some element of IP(Q)
(7) I := intersection of the successor sets of each element of IP(Q) except Y
We suggest the following implementation of step 7:
(7') For each z in IP(Q) except Y do
         For each successor s of z do
             Increment count(s)
         For each item s do
             If count(s) = |IP(Q)| − 1 then I := I ∪ {s}

(8) For each successor X of Y in order by size do
         If X is in I and X is a successor of Q (comparison test) then
             S := S ∪ {X}
             Eliminate successors of X from the rest of the for loop.
(9) Return S.
```

Phase III. (update immediate predecessor and successor sets of other items)

```
(10) For each x in IP(Q) do
         S(x) := IS(x) ∪ {Q} − IS(Q)
(11) For each x in IS(Q) do
         P(x) := IP(x) ∪ {Q} − IP(Q)
```

here. Other applications are possible [34]. For simply represented states and patterns such as tile-puzzle states even more efficient schemes are possible [9, 12]; the same is true for a set of structured pattern-weights that are fixed since heavy precompilation can enable the structure screening and comparison operations to be done simultaneously [35].

7 Extending to Real-World Planning and Search

The tile-puzzles have provided a good framework in which to illustrate the fundamentals of the pattern-weight formulation. But how might these methods extend to real-world domains? We shall consider two aspects of real-world problems:

1. Reactive domains in which the agent does not have absolute control of the world, i.e. the state resulting after an operator application is not fully-predictable.
2. Domains in which the structure of the state-space is not fully known or knowable due to high combinatorics or insufficient knowledge available to the agent.

7.1 Reactive Domains

We are suggesting that it is possible to use macro-operators and subgoals as they are currently being used, but by converting them to pws, more flexibility and robustness is achieved. In reactive domains the effects of each operation and the state of the world are not fully predictable. Attempting to execute a macro in such an environment becomes a risky operation. The longer the macro the less likely it is of succeeding because at some point the preconditions of the next operator to execute may not be satisfied.

It is always possible to respond in these cases using a macro-table since an action is recommended in every situation. But the macro-table, being confined to produce a solution based on a particular subgoal sequence may lead to tremendous inefficiencies, since during execution of a macro it is quite possible to purposely and successfully undo previously achieved subgoals. For example, in a "reactive" tile puzzle in which the effects of operators are unpredictable a certain percentage of the time a macro-table may be endlessly returning the blank to the center since this is always the recommendation when the blank is not in its goal position.

The basic reactive planning techniques such as STRIPS triangle table representation of macros [36] and Universal Planning [42] recover gracefully from interruptions but suffer from one of the two weaknesses of macro-tables by being limited to a single subgoal sequence or not taking advantage of *macro-crossover*.

In PET [38], macro-operators are decoupled into the form *PRE state description* $\Rightarrow$ *POST state description* with the interpretation:

IF the current state S matches PRE and
  the state resulting from applying OP to PRE matches POST
  such that the relations in the augmentation hold
THEN OP is recommended in S.

Similarly, in SOAR [22] macros are decoupled into situation-action rules. Both these forms are functionally equivalent since the *PRE*and *POST* conditions with the augmentation can be viewed collectively as the situation.

Situation-action schemes due to their lower granularity lead to more fruitful responses to reactive domains than macro-tables. Converting a macro-table into a situation-action rule base is similar to the conversion into pws (intermediate state patterns are generated), except that with each pattern rather than a weight being stored the operator to be applied when that pattern occurs is stored. The effect is that if a macro is interrupted in processing, as always the current state is matched against memory to see which operators apply, conflict resolution takes place in the case of multiple pattern matches and potentially the same macro or another useful one could be started up again from the middle. However, pws seem to provide a more useful scheme than this on several accounts:

1. Conflict resolution is handled naturally by DFHC as the weights induce a prioritization of patterns.
2. As DFHC evaluates "post-conditions" one level of lookahead is achieved. (Further levels are also possible with pws but may be less useful in reactive domains due to uncertainty.) In a situation-action framework it becomes more difficult to use lookahead since it is difficult to choose between post-conditions. Of course, if a good heuristic evaluator existed to choose amongst these the discussion becomes moot as the evaluator itself rather than situation-action rules could guide the search process.
3. Pw-sets from multiple macro-tables can be combined more easily and fruitfully than situation-action rules.

7.2 Incomplete Knowledge

The above uses of pws have assumed that complete knowledge of the structure of the state space is available and that the problem is serially-decomposable. Now we will consider the case in which such knowledge is not readily available. We shall break the discussion into two parts:

1. Only a subset of the pws are available and there are gaps. That is, a state with a known pattern P is no longer assumed to be adjacent to a state with a pattern of weight one less than P's weight, as was the case when pws were derived from a macro-table.
2. The structure of the state space can only be learned through experience and thus it is not possible to assign accurate weights to patterns.

Case 1 corresponds to the case in which Korf's subgoal distance is greater than 1. Subgoal distance, $D(S,T)$ = the maximum shortest distance between a state that satisfies S to a state that satisfies T, is defined as:

$$D(S,T) \equiv \max_{s \in S} \min_{t \in T} d(s,t)$$

Since patterns and subgoals are defined identically, the above formula can be used to establish a pattern distance function. Thus the same methods that are used to solve problems in which the maximum subgoal distance [19] is greater that 1 can be applied here: The problem defined by moving from one subgoal to the next is solved using some technique such as breadth-first search, A* or means-ends-analysis that is exponential in the subgoal distance in the worst case. However, as we saw before, the weights associated with patterns provide greater flexibility than a simple subgoal sequence: from the current state S one can do breadth-first search until reaching a state with a pattern whose weight is less than the evaluation of S. This state becomes the new current state. Thus, essentially performing DFHC with lookahead. The advantage of this approach over subgoals is that it is not necessary to establish ahead of time which subgoals will be solved and in which order. Patterns are not used as conditions to be achieved, but as signposts of progress. As long as progress is being made, which patterns are being achieved is unimportant. Of course, as with subgoals, it is possible to choose a particular pattern as the "next goal" and use A* or means-ends analysis to achieve it.

Case 2 is the critical case and is a challenging problem for all planning systems. Here, the structure of the domain can only be learned through experience. It is not possible to assign accurate weights to patterns. The weights may be too large because one does not yet know how to solve the problem optimally, or they may be too small because all states that contain that pattern as a subpattern may not have been considered yet. Still there is room for research progress here. Perhaps, the weights associated with pws should be ranges (lower and upper bounds) and search algorithms built up around them as in B* [4]. These ranges associated with patterns provide a useful mechanism of encoding only partial knowledge of a domain. With subgoals and macro-operators more preciseness is required.

8 Conclusions and Ongoing Work

Let us summarize what has been learned:

1. We have outlined the difficulties with the macro-table technique that cause solution paths to be far from optimal, namely, due to high granularity, *macro-crossover* is not exploited and macros are based on one subgoal sequence.
2. We showed that Squeeze can improve solutions by looking for duplicate states, but can not take advantage of partial matches.
3. We give a construction that takes a macro-table and converts it into pws.

4. The pw version of a macro-table when used with DFHC produces significantly shorter solution paths on the average than from the macro table (alone or with Squeeze) by exploiting *macro-crossover*.
5. Dynamic subgoaling can be achieved by using the pw sets from multiple macro-tables (and hence multiple subgoal sequences). By taking advantage of *subgoal sequence crossover*, solution paths significantly closer to optimal are achieved.
6. We give an efficient method for organizing pattern-based state evaluation.
7. Pw versions of macro-operators and subgoals may lead to more robust execution in reactive domains.

We are also studying methods by which pws can be learned from experience. Similar work is also being pursued by others [41]. Morph is a self-learning pattern-oriented chess program that attempts to create pws that are useful in evaluating chess positions. We have seen that macro-like behavior arises automatically from the pws. Efforts are underway to bring Morph to as strong a level as is possible using a shallow search depth. [27, 28, 30, 32, 33]. It is our hope that this pattern-oriented, low-search approach will be more consistent with cognitive models of human chess performance[13].

Acknowledgements

Steve Grimm wrote the initial program code and improved the descriptions of the algorithms. Richard Snyder and Ira Pohl helped with an early draft. Sherri Shahrokhi wrote the initial table generation routines for the 8-puzzle. Thanks to the systems staff for giving us adequate disk space to carry out the experiments.

References

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D., Data Structures and Algorithms, Addison-Wesley, 1983.

[2] Bagley, J.D., The behavior of adaptive systems which employ genetic and correlation algorithms, PhD Thesis, University of Michigan, Dissertation Abstracts International, 28(12), 5106B. University Microfilms No. 68-7556, 1967.

[3] Banerji, R. B., Developments with GPS, in Search in Artificial Intelligence, Kanal, L. and Kumar, V.(ed.), pp. 268-286, Springer-Verlag, New York, 1988.

[4] Berliner, H. "The B* Tree Search Algorithm: A Best First Proof Procedure," Artificial Intelligence, vol. 12, No. 1, May 1979

[5] Christensen, J. and Korf, R., A unified theory of heuristic evaluation functions and its application to learning, in Proc. AAAI-86, 1986.

[6] Colin, C. and Levinson, R., Partial order maintenance: a tutorial, in Special Interest Group on Information Retrieval (SIGIR) Forum. vol. 23, numbers 3,4 pp. 34-59, 1990.

[7] de Groot, A. D., Thought and Choice in Chess, The Hague: Mouton, 1965.

[8] Dreyfus, H., and Dreyfus, S. Mind over Machine, New York: The Free Press, 1986.

[9] Forgy, C. L. On the efficient implementation of production systems. PhD Thesis, Carnegie-Mellon University,1979.

[10] Pattern Associativity and the Retrieval of Semantic Networks In the Special Issue of Computers and Mathematics on Artificial Intelligence and Semantic Networks, Fritz Lehmann, editor. Pergammon Press. 1990.

[11] Gibbons, A., Algorithmic Graph Theory, Cambridge University Press, 1985.

[12] Hayes-Roth, F. and Mostow, D. J., An automatically compilable network recognition network for structured patterns. In Proc. IJCAI-75, 246-251, 1975.

[13] Hearst, E., "Man and machine: chess achievements and chess thinking", in Chess Skill in Man and Machine, edited by Frey, P.W., Springer-Verlag, 1977.

[14] Holland, J., Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, 1975

[15] Johnson, W. and Store, W. "Notes on the '15' puzzle," A American Journal of Mathematics, vol. 2, 1897, pp 397-404

[16] Joslin, D.E. and Roach, J. W., A Theoretical Analysis of Conjunctive-Goal Planning, Artificial Intelligence, Volume 41, Number 1, November, 1989, pp. 97-106.

[17] Klir, G. J., Architecture of Systems Problem-Solving, Plenum Press, New York, 1985

[18] Knoblock, C. "A Theory of Abstraction for Hierarchical Planning," Change of Representation and Inductive Bias, Kluwer Publishers, 1990

[19] Korf, R. E., Learning to solve problems by searching for Macro-Operators. Research Notes in Artificial Intelligence 5, Pitman Advanced Publishing Program, 1985.

[20] Korf, R. E. Planning as search: a quantitative approach, Artificial Intelligence, 1987.

[21] Korf, R. E., Optimal path finding algorithms, in Search in Artificial Intelligence, Kanal, L. and Kumar, V.(ed.), pp.223-267, Springer-Verlag, 1988.

[22] Laird, J., Newell, A., and Rosenbloom, P., "SOAR: An Architecture for General Intelligence," Artificial Intelligence, Vol. 33, pp. 1-64, Elsevier Science Publishers, Amsterdam, 1987

[23] Lee, K. F. and Mahajan, S. "A Pattern Classification Approach to Evaluation Function Learning", Artificial Intelligence, Vol. 36, 1988, pp. 1-25.

[24] Levinson, R., A self-organizing retrieval system for graphs, In Proceedings of AAAI-84. 1984.

[25] Levinson, R., A self-organizing retrieval system for graphs., PhD Thesis, Technical Report AI-85-05, from Artificial Intelligence Laboratory, University of Texas, Austin, 1985.

[26] Levinson, R. Intelligent signal analysis and recognition using a self-organizing database, in Proceedings of The First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, ACM. 1988.

[27] Levinson, R., "Pattern formation, associative recall and search: a Proposal", Technical Report available from Baskin Center of Computer Engineering, University of California, Santa Cruz, 1989.

[28] Levinson, R., A self-learning, pattern-oriented, chess program", in International Computer Chess Association Journal, pp. 207-215, Dec. 1989, Vol. 12, No. 4.

[29] Levinson, R., A Pattern-Weight Formulation of Search Knowledge, Technical Report UCSC-CRL-89-05, available from the Baskin Center of Computer Science, University of California, Santa Cruz, 1989.

[30] Levinson, R., Snyder, R., Beach, B., Dayan, T., and Sohn, K., Adaptive-Predictive Game-Playing Programs, submitted to Journal of Experimental and Theoretical AI.

[31] Levinson, R., Pattern Associativity and the Retrieval of Semantic Networks, in Special Issue of Computers and Mathematics on Semantic Networks in Artficial Intelligence, Fritz Lehmann, editor. Pergamon Press, 1991.

[32] Levinson,R. and Snyder,R. Adaptive Pattern-Oriented Chess, in *Proceedings of AAAI-91*, pp.601–605, Morgan Kaufman, 1991.

[33] Levinson,R. Experience-Based Creativity, in Proceedings of Symposium on Artificial Intelligence, Reasoning and Creativity, Brisbane, Kluwer Academic Press, to appear in the book, "Artificial Intelligence and Creativity", Terry Dartnall, ed., Kluwer 1994.

[34] Levinson, R. A Self-Organizing Pattern Retrieval System in *Int'l Journal of Intelligent Systems*, vol. 6.,717-738, John Wiley and Sons. Also appears as Technical Report UCSC-89-21, University of California, Computer Research Laboratory, Santa Cruz, CA 95064.

[35] Nagy, M. Z., Kozics, S., Veszpremi, T. and Bruck, P., Substructure search on very large files using tree-structured databases in Chemical Structures: The International Language of Chemistry, Wendy A. Warr (ed.). Springer-Verlag, 1988.

[36] Nilsson, N. J. Triangle tables: a proposal for a robot programming language. Tech Note 347, AI Center, SRI International, 1985.

[37] Pohl, I., Bi-directional search, in Machine Intelligence 6, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, pp. 127-140, 1971.

[38] Porter, B. and Kibler, D., "Experimental Goal Regression: A Method for Learning Problem-Solving Heuristics," Machine Learning 1, pp. 249-286, Kluwer Academic Publishers, Boston, 1986

[39] Ratner, D. and Pohl, I., "Joint and LPA*: Combination of ap- proximation and search," Proceedings of the AAAI-86, pp. 173-177, 1986.

[40] Ratner, D. and Warmuth, M., "Finding a shortest solution for the NxN extension of the 15-puzzle is intractable," Proceedings of the AAAI-86, pp. 168-172, 1986.

[41] Ruby, David and Kibler, Dennis Learning subgoal sequences for planning, Proceedings of Eleventh International Joint Conference on Artificial Intelligence, Vol. 1, pp. 609-614, 1989.

[42] Schoppers, M. J., Representation and automatic synthesis of reaction plans, PhD Thesis, Tech Report UIUCDCS-R-89-1546, Department of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[43] Readings in Machine Learning, edited by Shavlik, J., and Dietterich, T. Morgan Kaufmann Publishers, Inc, 1990

[44] Watanabe, S. Pattern Recognition Human and Mechanical, John Wiley and Sons New York, 1985

[45] Wilcox, C.S. and Levinson, R.A., A self-organized knowledge base for recall, design, and discovery in organic chemistry, in Artificial Intelligence Applications in Chemistry edited by T. H. Pierce and Hohne, B. A., ACS Symposium Series, No. 306. 1986.

A Appendix

Tables A.1 and A.2 below summarize experiments run on the 8-puzzle and 15-puzzle, respectively. Table A.1 illustrates 100 samples out of a 10,000 sample where solvable initial states were generated by a random number (between 50,000 and 100,000) of tile swaps from the goal state where swaps did not correspond to legal moves.

The columns for Table A.1 are as follows:

- Init. state = the initial state.
- Opt= Optimal solution length determined using A* with Manhattan Distance.
- Mt = Macro table solution length based on "top" subgoal sequence: 012345678.
- Sqz= Squeeze applied to Mt solution.
- Mr = Macro table solution length based on "right" subgoal sequence:034567812.
- Mb = Macro table solution length based on "bottom" subgoal sequence:056781234.
- Ml = Macro table solution length based on "left" subgoal sequence:078123456.
- Mi = Macro table solution length based on "interleaved" subgoal sequence:024681357.
- Mm = Minimum of Mt,Mr,Mb,Ml,Mi.
- Pt = Solution length based on Pws derived from Mt macrotable.
- Pr = Solution length based on Pws derived from Mr macrotable.
- Pb = Solution length based on Pws derived from Mb macrotable.
- Pl = Solution length based on Pws derived from Ml macrotable
- P4 = Solution length based on combined Pws from Mt,Mr,Mb,Ml.
- P5 = Solution length based on combined Pws from Mt,Mr,Mb,Ml and Mi.

Table A.2 illustrates 49 samples out of a 100 samples where solvable initial states were generated for the 15-puzzle as was done for the 8-puzzle.

The columns for Table A.2 are as follows:

- Init. state = the initial state. Using hexadecimal representation of tile numbers. Given time and resource constraints optimal solutions were obtained for just the two states * = 38 with 74307 states expanded and ** = 44 with 48320 states expanded using standard A* with Manhattan distance. All other states required more than 10,000 expansions to solve.
- Mt = Macro table solution length based on "top" subgoal sequence: 0123456789abcdef
- Sqz= Squeeze applied to Mt solution.
- Mr = Macro table solution length based on "right" subgoal sequence: 048c37bf26ae159d
- Mb = Macro table solution length based on "bottom" subgoal sequence:0fedcba987654321 (These solutions not shown in table).
- Ml = Macro table solution length based on "left" subgoal sequence: 0d951ea62fb73c84 (Also not shown).
- Mi = Macro table solution length based on "interleaved" subgoal sequence:02468ace13579bdf.
- Mm = Minimum of Mt,Mr,Mb,Ml,Mi.
- Pt = Solution length based on Pws derived from Mt macrotable.
- Pr = Solution length based on Pws derived from Mr macrotable.
- Pb = Solution length based on Pws derived from Mb macrotable (Not shown).
- Pl = Solution length based on Pws derived from Ml macrotable (Not shown).
- Pi = Solution length based on Pws derived from Mi macrotable.
- Pm = Minimum of Pt,Pr,Pb,Pl,Pi.
- P4 = Solution length based on combined Pws from Mt,Mr,Mb,Ml.
- P5 = Solution length based on combined Pws from Mt,Mr,Mb,Ml and Mi.

Init. state	Opt	Mt	Sqz	Mr	Mb	Ml	Mi	Mm	Pt	Pr	Pb	Pl	Pi	Pm	P4	P5
612504873	18	28	20	36	34	48	78	28	20	36	20	48	64	20	20	20
254613870	20	30	22	26	42	30	80	26	20	26	30	28	56	20	26	26
726481530	22	38	30	50	38	54	52	38	30	48	30	52	70	30	52	52
436178052	24	40	38	36	48	40	36	36	38	48	46	36	46	36	48	48
068315724	20	30	30	42	42	50	68	30	30	42	40	48	58	30	40	40
726305841	24	34	32	40	50	36	68	34	32	38	48	34	66	32	34	30
542703186	18	18	18	34	22	42	68	18	18	32	22	42	50	18	32	32
756104328	26	32	32	46	50	30	70	30	32	42	48	30	60	30	32	32
348576021	22	48	46	52	44	34	56	34	48	46	44	34	54	34	44	44
472136085	20	46	44	50	44	48	60	44	44	40	38	46	56	38	46	46
370682541	22	50	50	46	60	52	66	46	34	44	56	50	56	34	44	44
041267358	26	44	42	56	52	30	76	30	40	54	52	28	76	28	28	28
257864013	24	38	38	26	44	52	58	26	38	30	32	42	50	30	42	50
258307614	22	30	28	34	48	46	44	30	26	34	46	46	42	26	34	34
186572430	18	42	42	40	40	44	38	38	42	40	38	38	36	36	42	42
341568072	24	32	32	50	32	30	44	30	48	52	32	26	64	26	32	32
426873150	20	52	50	52	30	30	76	30	50	52	20	30	38	20	30	30
281576043	18	42	42	52	34	26	78	26	48	40	34	26	40	26	34	34
817205643	22	38	34	32	44	38	44	32	34	22	44	30	44	22	34	34
548367210	26	36	36	46	46	44	76	36	36	38	46	42	76	36	38	38
832461570	26	46	44	32	50	46	48	32	44	42	46	44	48	42	40	40
482706135	24	50	48	42	36	32	66	32	48	42	36	32	52	32	32	32
320741865	20	48	44	46	48	22	58	22	44	36	48	30	54	30	30	30
540612738	24	44	40	48	42	54	62	42	40	48	36	48	60	36	36	36
513824067	16	44	42	38	56	44	20	20	16	36	50	20	16	16	36	36
246817530	18	46	44	28	46	20	76	20	42	28	42	18	64	18	42	42
274536081	26	56	56	42	50	40	60	40	34	40	28	38	40	28	40	40
348276150	16	34	34	30	24	16	32	16	34	28	24	16	32	16	20	20
371604825	20	46	44	38	36	38	30	30	44	36	32	36	30	30	36	36
042135876	10	26	24	50	32	32	60	26	20	48	10	10	42	10	10	10
538206417	22	36	36	46	52	44	56	36	36	44	50	44	54	36	44	44
582167340	24	44	34	50	46	40	54	40	34	42	46	36	52	34	34	34
576431082	28	46	46	38	40	44	68	38	46	44	46	54	60	44	44	44
730146258	22	34	34	42	58	28	56	28	32	38	58	44	44	32	38	38
830624715	16	32	30	24	16	36	38	16	30	22	16	24	30	16	22	22
146307528	22	32	30	46	32	38	60	32	30	38	32	38	42	30	30	30
621703458	24	46	44	48	46	44	52	44	44	46	46	44	52	44	46	46
064752318	26	48	44	52	38	42	70	38	44	40	38	40	60	38	38	38
524786130	26	36	32	48	34	52	68	34	32	38	30	52	64	30	30	30
872306415	26	52	52	42	30	34	56	30	52	42	30	34	46	30	34	34
230415678	20	48	46	30	32	38	32	30	46	30	44	36	30	30	30	30
438572061	26	46	46	48	48	36	72	36	42	44	38	36	72	36	44	44
845271360	20	40	38	38	28	28	60	28	38	38	20	36	44	20	20	20
170862435	20	40	40	50	36	42	58	36	40	40	28	32	56	28	32	32
072135864	18	42	42	36	48	46	54	36	38	36	46	42	42	36	38	38
826703451	22	50	48	46	34	46	72	34	48	44	32	36	70	32	46	46
128406753	14	36	36	28	38	26	30	26	36	28	36	26	28	26	36	36
132674580	20	38	34	50	48	34	80	34	34	22	48	30	70	22	24	24
538476210	26	56	52	48	40	46	78	40	52	44	40	46	76	40	44	44
140863572	20	44	42	34	36	32	42	32	38	26	38	32	60	26	38	38
number of trials, this page: 52								Goal State: 123804765								

Table A.1: Comparing Pattern and Macro Strategies for 8-puzzle.

(table continued on next page)

Init. state	Opt	Mt	Sqz	Mr	Mb	Ml	Mi	Mm	Pt	Pr	Pb	Pl	Pi	Pm	P4
740358216	24	52	48	36	40	46	72	36	48	34	38	40	72	34	40
018367245	24	46	46	36	38	42	82	36	46	36	36	38	62	36	38
387164250	22	24	24	48	30	40	46	24	24	36	30	40	38	24	40
170236584	22	26	26	32	46	48	42	26	26	24	38	44	34	24	26
621408537	26	42	38	50	50	42	70	42	38	48	50	42	70	38	38
378615042	20	26	24	54	42	42	54	26	20	28	52	40	36	20	40
087413562	26	44	42	44	54	48	48	44	42	42	50	40	48	40	42
154736820	22	40	40	30	36	32	42	30	40	26	34	32	42	26	40
372541680	26	32	32	48	52	38	54	32	32	38	34	38	54	32	38
037625841	22	36	34	32	58	34	30	30	34	28	56	32	30	28	32
014832765	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
780146235	22	46	46	42	54	48	56	42	44	38	54	50	46	38	50
075312864	22	32	32	42	54	52	74	32	32	40	54	48	74	32	32
671524830	22	38	36	48	50	46	44	38	36	48	54	38	64	36	42
162573084	12	22	14	22	14	34	22	14	14	14	12	32	14	12	12
081563427	20	46	46	34	42	46	64	34	44	32	26	44	64	26	32
140785263	16	44	42	42	36	52	56	36	42	30	36	48	56	30	36
827406153	22	44	44	46	46	40	70	40	42	44	44	40	60	40	40
514276380	24	34	32	44	48	42	52	34	36	44	48	42	48	36	32
472803165	16	16	16	40	26	22	46	16	16	30	24	20	32	16	20
735421086	18	50	46	24	48	40	56	24	46	26	24	44	52	24	26
415768320	24	44	44	38	48	54	84	38	42	38	46	44	76	38	46
613407825	20	42	40	40	44	42	30	30	40	38	38	38	30	30	38
681503724	20	44	42	44	36	36	46	36	42	34	36	36	46	34	36
086217354	24	36	28	46	24	32	42	24	28	36	24	40	32	24	28
054173268	22	42	42	42	42	56	76	42	38	40	38	48	74	38	38
018526347	24	40	40	40	44	42	50	40	40	42	44	42	48	40	40
067581324	26	42	42	26	52	50	68	26	42	28	42	50	60	28	50
018452376	24	24	24	56	40	42	44	24	24	30	46	32	44	24	24
247306518	24	42	40	56	40	38	74	38	38	56	40	38	60	38	56
620715843	18	30	20	34	46	50	74	30	20	40	36	48	70	20	20
047521386	26	44	42	50	42	38	56	38	42	40	32	36	48	32	36
160372854	16	24	24	30	42	48	42	24	24	30	40	28	34	24	24
012748536	16	36	34	22	38	50	30	22	32	22	38	48	22	22	32
472163058	20	22	22	50	34	50	28	22	22	50	54	44	34	22	36
356402817	22	36	36	40	46	30	42	30	36	40	44	30	40	30	32
730584612	20	44	44	34	46	44	50	34	44	32	44	42	32	32	32
730645182	22	46	44	30	46	32	64	30	44	28	46	32	38	28	28
562817034	22	46	36	50	36	40	50	36	32	42	24	50	46	24	32
283506174	12	12	12	20	12	12	48	12	12	12	12	12	48	12	12
852476130	20	46	28	38	36	58	76	36	28	36	28	56	66	28	28
153268047	24	36	36	40	54	36	70	36	38	34	52	38	68	34	34
081642357	24	44	40	44	38	36	60	36	38	46	30	32	52	30	32
286507134	24	38	30	30	24	42	60	24	28	30	24	42	60	24	28
524817360	24	42	38	42	50	36	76	36	36	34	48	34	44	34	36
451803672	22	44	40	32	46	36	74	32	40	32	42	26	66	26	42
168457023	24	44	44	50	52	36	58	36	40	48	52	34	56	34	40
524106738	20	32	30	44	30	50	66	30	28	42	22	48	62	22	22
428763015	20	48	46	48	42	36	44	36	46	46	38	34	46	34	34
Average	21.6	39.9	37.7	40.1	40.1	40.2	57.6	31.5	36.9	36.8	36.9	36.7	52.3	28.4	33.3
number of trials: 10000								Goal State: 123804765							

Table A.1: Comparing Pattern and Macro Strategies for 8-puzzle.

100 out of 10000 trials shown; averages correspond to all trials.

Init. state	Mt	Sqz	Mr	Mi	Mm	Pt	Pr	Pi	Pm	P4	P5
069dbef4c58a7213	178	132	136	174	136	158	114	174	112	112	112
8dc652f3b479e0a1	142	128	160	190	112	118	152	144	118	122	122
e718d2c6ab3f5094	164	150	168	138	138	112	152	134	112	112	112
6f5dc013a4eb2798	170	156	166	188	166	168	120	136	106	106	106
1fe24a869703d5cb	134	118	136	126	120	104	114	104	66	66	66
f63d8a24bc07519e	150	146	192	170	118	128	148	152	74	74	74
cbe2d7a6f8435190	158	152	172	226	154	144	166	178	118	118	118
d20b3794f5ae68c1	138	134	150	140	138	100	128	140	100	124	124
392658cb0e17fda4	178	162	150	168	142	130	106	150	96	96	96
8b0cdf239a45e167	160	128	152	174	148	90	148	138	88	88	88
78bd564c1903a2ef	152	138	152	186	136	102	104	114	102	156	156
750fa1b28c6d43e9	178	174	214	208	144	120	170	176	102	102	102
e6fa873b912d4c50	174	158	182	198	168	128	168	188	128	138	138
0148e237b6dcf5a9*	140	130	130	150	130	92	108	106	92	92	92
9bf8dae7c1065234	152	144	176	200	126	112	124	192	112	114	114
a20f897e6d54c3b1	142	128	152	162	142	114	170	152	114	138	138
b9a480f2537d6e1c	138	124	166	156	138	118	122	152	118	132	132
1fb8d2a03e57c496	172	156	140	192	138	130	166	162	118	130	130
a7e95dbfc1284360	134	124	166	226	128	126	138	180	120	120	120
a4081d69cbef5732	170	166	138	144	138	116	94	152	94	120	120
b4026a13f9e578dc	148	122	188	172	132	108	122	120	108	122	122
6a3729f4cde81b50	116	96	146	178	116	116	76	146	76	80	80
f2ea3b76d1485c90	190	164	138	170	138	118	140	176	116	116	116
431c5e976abdf820	122	110	162	190	122	98	110	132	98	106	106
4c925016efba738d	152	146	192	172	142	154	168	186	104	154	154
f39ace56d8217b40	158	154	168	200	158	120	156	174	120	146	146
3a482916e5dc70fb	188	166	148	182	138	98	140	144	68	68	64
8f5617bdeac34290	124	114	170	206	124	138	154	148	112	136	136
93ab5270d6ecf148	156	146	116	184	116	118	90	146	90	100	100
b48c3ed906a715f2	162	138	182	204	162	144	108	150	108	162	162
d4b780f56a9213ec	156	140	162	152	120	140	124	140	92	92	92
0ae51c84d72fb963**	158	136	142	150	142	104	94	130	94	104	104
e5bd9f81620ac473	146	134	182	190	124	130	166	158	106	106	106
329871af5e4d60cb	160	154	190	144	136	74	134	154	74	86	86
9ca62815df0e7b43	154	136	210	180	140	96	94	178	94	126	126
a34b6dcf192875e0	176	158	156	182	138	100	118	160	100	114	114
14095fbc672e3a8d	112	100	194	162	112	96	106	142	96	96	96
463a90f18cd72be5	148	142	172	174	148	138	172	162	124	128	128
d60815afc73429eb	148	138	160	202	148	142	120	164	120	92	92
8f072ca9e153d64b	150	134	178	212	150	106	162	184	106	126	126
c487ed1af52b9036	142	124	164	210	134	104	154	176	104	108	108
e3b1a57d968c4f20	124	110	164	170	124	154	140	130	116	116	116
d79fb3e645a2108c	168	150	150	198	136	150	122	172	96	96	96
b49afe2501c8637d	122	122	164	180	122	116	128	120	116	136	136
28b7e9fa04cd1635	148	146	158	152	96	142	144	174	96	96	96
657efa4b32c9d810	162	158	164	170	144	134	148	168	92	92	92
3a9712ce6f4b508d	140	110	130	166	108	120	82	134	82	92	92
0cdb534917a86f2e	170	164	142	200	142	142	130	166	130	134	134
08fa41de3b7562c9	182	172	202	196	182	164	126	152	126	164	164
Average	158.2	144.3	166.0	179.8	138.4	122.5	134.1	139.8	106.6	116.8	116.6
number of trials: 100						Goal State: 123456789abcde0					

Table A.2: Comparing Pattern and Macro Strategies for 15-puzzle.

49 out of 100 trials shown; averages correspond to all trials.