Extra-sliding puzzles: Experiments in Cognitive Modeling and Representation for Problem Solving

Elpida S. Tzafestas

[1] Institute of Communication and Computer Systems
National Technical University of Athens
Zographou Campus
15773 Athens, Greece
brensham@softlab.ece.ntua.gr

Abstract. We are studying the well known problem of the sliding puzzle or N-puzzle and we are modeling a number of consecutive extensions and their solution based on the known algorithmic solution. The usual approach to date has been to find optimal solutions (move sequences) to the N-puzzle with the aid of sophisticated special-purpose search methods. Our goal is to study it from a bottom up perspective and to do cognitive modeling of the scalable algorithmic solution that finds suboptimal move sequences but that can do so consistently for a large number of extensions with practically no increase in polynomial complexity. This approach claims both to model for fundamental cognitive skills that are exerted on top of existing intelligent problem-solving methods as well as to identify modules of reasoning that should be included in any scalably intelligent system.

Keywords: Sliding puzzle, cognitive modeling, problem solving, problem decomposition, planning, representation, mental rotation.

1 Introduction

The usual artificial intelligence approach to puzzle solving consists in describing and representing the puzzle knowledge in a way that allows one of the general search methods to apply, possibly modified appropriately in order to exploit the particularities of the puzzle [1]. The application of the general or modified search method then yields optimal or near-optimal solutions, i.e. move sequences that drive the puzzle from its initial state to the final solved state. The axiom behind this approach is that a general method for (intelligent) problem-solving should be able to solve all imaginable problems: thus, heuristic search should be able to solve all those problems that like puzzle or perfect knowledge games involve reasoning on a sequence of states or moves.

This approach is not however universally accepted. Doubts were cast even in the early days of AI ; for example, David Marr [2] stated that "*A result in Artificial Intelligence thus consists in the isolation of a particular information processing*

problem, and the statement of a method for solving it. Some judgement has to be applied when deciding as to what constitutes a method – something based on exhaustive search for chess would clearly not qualify. […] The important point is that when a method has been established for a particular problem it never has to be done again, and in this respect a result in A.I. behaves like a result in mathematics or any of the hard natural sciences". The alternative to exhaustive or heuristic search is to model exactly what the human puzzle solver does, i.e. to do human cognitive modeling and reproduce the human subject's problem-solving behavior as much as is possible, instead of relying on a general but artificial method to solve the problem. Apart from the obvious human modeling target, cognitive modeling of this sort may also allow the reuse of its results to solve other analogous problems.

Our belief is that many of our problem solving skills involve a number of discrete or "geometric" manipulations of knowledge that is either basic or acquired through instruction and experience with other more or less similar problems. Thus our medium term goal is to identify the kinds of "bricks" and "glue" that may or should be used to build scalable artificial intelligence systems, i.e. to solve increasingly complex problems with minor or no increase in complexity. To this end, we are studying a number of well-known puzzles and how a human problem solver scales his/her behavior to more complex cases by exploiting the knowledge from previous simpler cases.

In this article we are studying the sliding puzzle (N-puzzle) for which a search-derived optimal solution exists for small values of N [3] as well as an algorithmic suboptimal solution. We are adopting the latter and we are scaling it to a number of more complex versions by adding specific skills at each level or "simple heuristics that make us smart" [4]. The algorithmic solution has so far been disregarded by AI for not producing optimal solutions. However, for human-level intelligence, evaluation of intelligence is almost never done on the basis of optimality. In the case of puzzles, all human competitions center around the speed of solution, which given the physically limited human dexterity translates mostly as reasoning speed to find one solution quickly enough. The quest of the optimal solution is a completely different mathematical problem, generally uninteresting from an every-day bounded resources point of view.

Our study serves as an illustration of what it should take to devise and solve more complex and difficult problems without involving higher complexity skills, i.e. by representation manipulation and problem decomposition. Our claim is then that human-level intelligence involves precisely those skills that tame and deconstruct complexity rather than those that ensure optimality and our aim is to both identify the skills and develop ways of incorporating them to real-world problem solving and to practical AI programming.

2 NM sliding puzzle

The original N-puzzle (which is an instance of a sliding puzzle) is defined as follows: A square of KxK tiles is given ($N=K^2$) where each tile is numbered from 1 to N as in fig. 1 (right). The last tile is missing and instead there is a blank position where an

adjacent tile may slide into. A random configuration is provided as initial state and the solution of the puzzle consists in finding a sequence of moves that brings the puzzle to the solved position. From here on and for the sake of simplicity, we will call N-puzzle the puzzle with NxN positions.

2.1 Basic algorithm

The algorithmic non-optimal solution is the following:

```
Solve the top row.
          {Foreach tile in turn (left to right): if
          righter than its  target position move up and
          left until position is found, else move right
          to wall, then up and left}
Solve the remainder of the left column.
          {Foreach tile in turn (top to bottom): if
          lower than its target position move left and
          up until position is found, else move down to
          wall, then left and up}
The order of the puzzle is now reduced by 1.  Recurse
as necessary (N-1 puzzle).
```

Figure 1 shows an example run. The complexity of the algorithm is $O(N^3)$, because each tile needs at most 2N steps to be correctly placed.

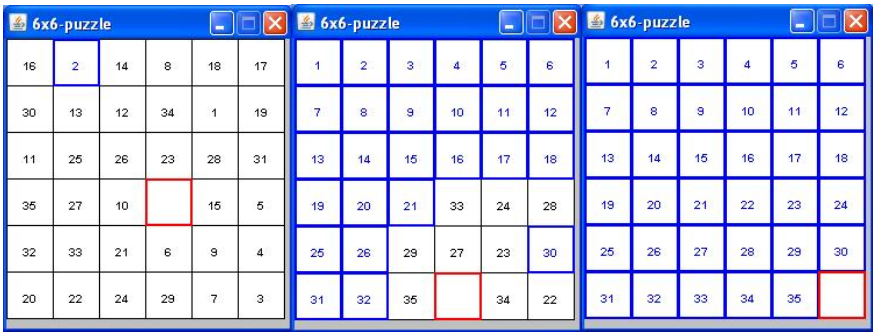


Fig. 1. Initial state, an intermediate state, final state. The tiles that are currently correctly placed are heavily marked in blue, while the current position of the blank is marked in red. One can clearly see the progression from top row, left column toward bottom and right.

2.2 NM Puzzle

The puzzle formulation as well as all solutions (search-based and algorithmic alike) actually do not depend on the puzzle being an exact square and can work equally well conceptually on a NxM puzzle with different lengths for the edges. This is the definition we have adopted in what follows (the square puzzle is a special case of the

NM puzzle). Figure 2 shows an example run. The complexity of the algorithm in this case is $O(X^3)$, where X=max(N,M), because each tile needs at most M+N steps to be correctly placed.

Fig. 2. Initial state, an intermediate state, final state.

2.3 Corner rule

The only intricate case in the algorithmic solution of the N puzzle or the NM puzzle is the handling of the last two tiles in a row (or column) because once the penultimate is correctly placed, the last one cannot be placed without disturbing the previous one. To solve this anomaly, a special pre-planned sequence is applied. First, we bring the penultimate and ultimate tiles to the positions shown below and we apply the sequence shown for the end of rows (a similar rule is used for the end of columns):

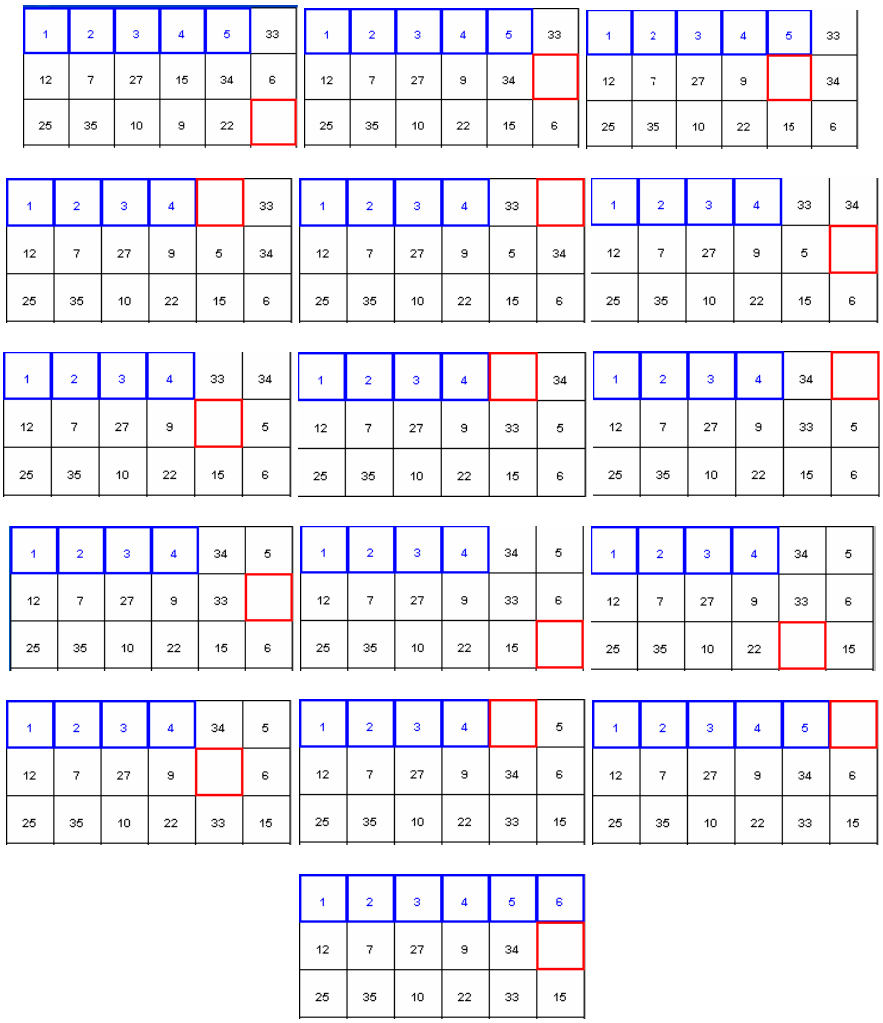


Fig. 3. Sequence of moves that solves the corner anomaly.

2.4 Lessons

While we have not touched on the issue of how the initial algorithm was found, we observe that it scales directly to the generic NxM puzzle case. We also observe that for a human problem solver, the directive to solve the top row and left column first and then the rest of the puzzle recursively is not sufficient. One has to discover a sequence of moves that solves the corner anomaly, whereas a search-based method does not distinguish betwwen these moves and the rest of the moves. This is an indication, that the original geometric knowledge and inspiration to solve the puzzle

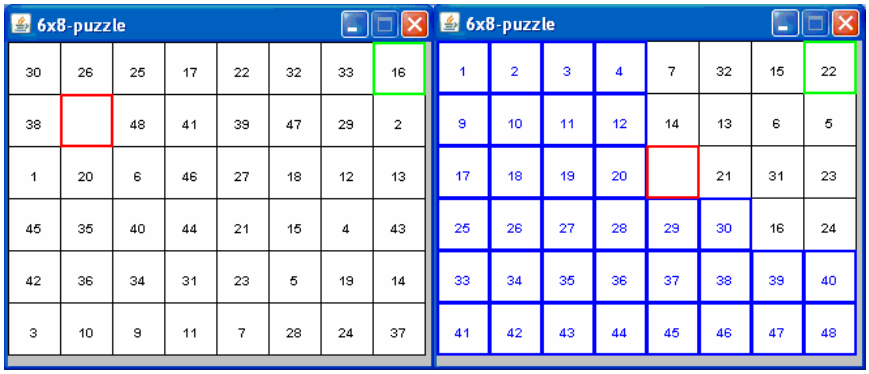
recursively has to be combined with some local search and planning method that is of an order tractable for a human mind. Inspection of the execution of the algorithm has also shown that the machine solver exploits unexpected opportunities (configurations that were expected later during solving) more easily than a human user. This is because the human generally focuses on the current step of the algorithm s/he is executing and may miss these opportunities. On the contrary an experienced human solver can find on the fly small efficient if not optimal sequences for certain limited positions s/he encounters and behaves thus "creatively" in front of some novel local opportunities.

3 Any corner sliding puzzle

The NM puzzle can be solved without a complexity penalty for the case of a blank that is not the NM-1 tile but one of the other three corners (tiles 1, M or NM-M+1). The only difference resides in the orientation of the puzzle; whereas for a human problem solver the handling of the puzzle will be straightforward once he starts thinking about it as a rotated one, a search-based method would in general involve a more intricate reinitialization of the state space reflecting the mental rotation.

3.1 Rotation rule

The mental rotation rule can be easily implemented in the artificial puzzle solver as an embedded renumbering of the tiles, the solving algorithm remaining intact. Figure 4 shows an example run.



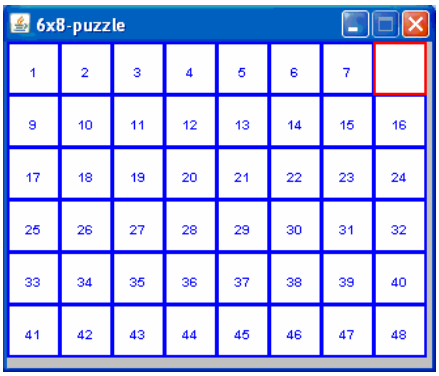


Fig. 4. Initial state, an intermediate state, final state. The target blank position is heavily marked in green. One can clearly see the (rotated) progression from bottom row, left column toward top and right.

3.2 Lessons

To implement the mental rotation rule one needs both recursion and re-representation. Change of representation (here: encoding) is important and is one of the tools that should come in the problem solver's bag. Note also that any one of the tools has to have an easy way to be integrated with others. Part of the criticism against traditional symbolic AI approaches is targeted toward the difficulty of implementing such "declarative" pattern-oriented rules within AI problem solvers. While the criticism may be theoretically correct, we believe that the difficulty is not conceptual, but rather it stems from the inherent limitation of usual symbolic, logic or, worst yet, procedural programming languages to handle directly such relational pattern-oriented information. The issue of the direct relation between image representations and cognition has been also put forth by Aisbett and Gibbon [5].

4 Any position sliding puzzle

The obvious extension of the NM puzzle is to have an arbitrary blank position. In this case, the human user mentally decomposes the puzzle into the part that can be directly solved to leave a smaller possibly rotated puzzle that can be solved as before. Again, a human problem solver would handle the puzzle in a straightforward recursive way, while a search-based method might involve another reinitialization of the state space reflecting the new blank position.

4.1 Decomposition rule

The solver first easily determines the minimum number of rows and columns to be solved before reducing the remaining of the puzzle to a regular puzzle with a corner blank. The rest of the algorithm is as before and the complexity of the solver does not change. Figure 5 shows an example run.

Fig. 5. Initial state, two intermediate states, final state. One can clearly see the two steps involved: first the filling of a few rows and columns from the top and the left, respectively, then the (rotated) progression from bottom row, right column toward top and left.

4.2 Lessons

To implement the decomposition a solver uses a criterion for decomposition that leads recursively to a puzzle it can solve. In the general case, we would expect human-like puzzle solvers to try various criteria to find out which one works -or works best- in order to develop next level algorithms. These criteria are unlike the ones used in

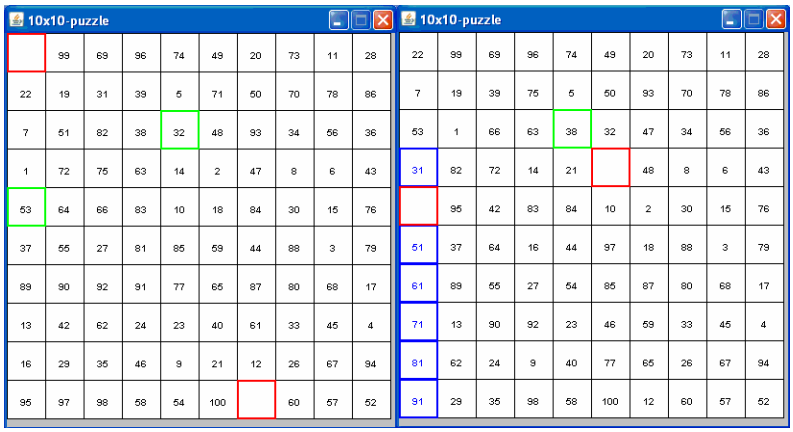
pattern database approaches [6] that seek to discover the optimum by introducing search "cuts" rather than our tractable routes to non-optimal quick solutions. Decomposition thus should be a cognitive "heuristic" appropriately implemented within a problem solver that serves not optimality but tractability [7].

5 Multi-objective sliding puzzle

The final extension of the NM puzzle is to have many arbitrarily placed blanks. In this case, the human user mentally plans the order of placement of blanks and decomposes the puzzle to parts that can be directly filled (rows and columns possibly with blanks) and at least one part that is a puzzle with an arbitrary or corner blank. Once more, a human problem solver decomposes very easily, while a search-based method would involve many reinitializations of the state space reflecting the various blanks configuration.

5.1 Rule for two blanks

The solver first easily determines the blank with the minimum euclidean distance to any of the corners and fills this part of the puzzle (including the corresponding row or column of the selected blank) reducing the remaining of the puzzle to a regular puzzle with an arbitrary blank. The rest of the algorithm is as before and the complexity of the solver does not change. Figure 6 shows an example run.



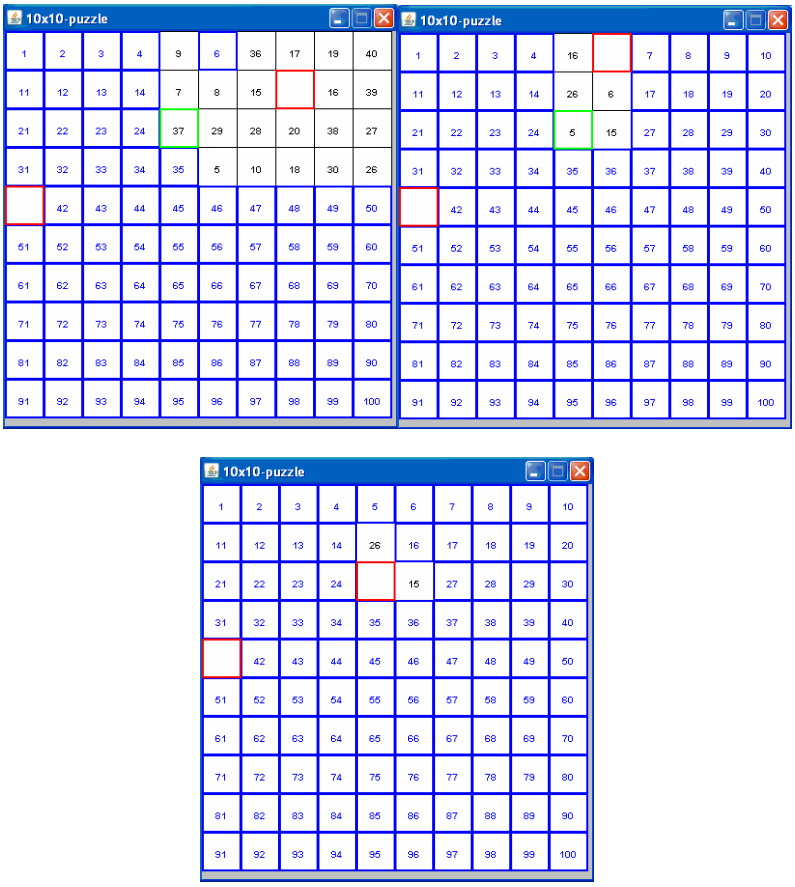
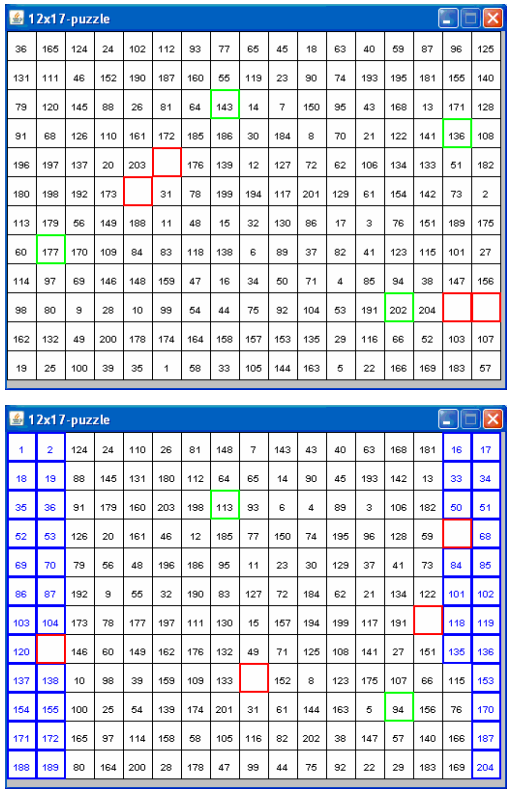


Fig. 6. Initial state, three intermediate states, final state. One can clearly see the three steps involved: first the filling of the left row that has a blank, then the filling of a few rows and columns from the bottom and the left, respectively, finally the (rotated) progression from top row, right column toward bottom and left. This solution only involved 2787 moves. Note that because the puzzle's initial configuration is not derived through tile shuffling but through random tile initialization, it is possible to come up with a variant of the (unsolvable) Loyd's puzzle [8], which is the case in this run.

5.2 Generalization

It is possible to extend this problem solver to accommodate larger blank sets. We have implemented a generalized version of the solver that solves for an arbitrary (but small, on the order of N or M) number of blanks, leaving for the future the case where the blanks are so many that it is easier to manipulate the numbered tiles within the blank space and direct them to their positions than to plan the solution as before. In this case as in the two blanks case, the human user mentally plans the order of

placement of blanks and decomposes the puzzle to parts that can be directly filled (rows and columns possibly with blanks) and parts that are puzzles with an arbitrary or corner blank. Once more, a human problem solver decomposes very easily, while a search-based method would involve several reinitializations of the state space reflecting the various blanks configurations, and this recursively in every successive number of blanks reduction. The solver decomposes recursively as in the case of two blanks and the complexity of the solver does not change. Figure 7 shows an example run for four blanks.



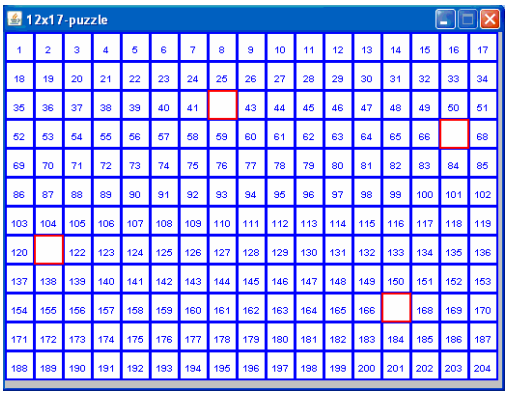


Fig. 7. Initial state, an intermediate state, final state. This solution only involved 8778 moves.

5.3 Lessons

To implement the decomposition for an arbitrary (small) number of blanks, a solver uses a recursive criterion for decomposition that leads recursively to smaller puzzles until it finds puzzles with a unique blank. In the general case, we expect human-like puzzle solvers to adopt the same criterion as in the previous case (any position blank). However, given the multitude of decomposition and planning possibilities available, a real human subject does not always choose the optimal criterion, but choices are differentiated across subjects and sometimes for the same subject across runs. Therefore the criterion selected is most often not the best but the fastest to retrieve, as dictated by experience, personality and context.

6 Conclusion: On representation and problem solving

We have designed and implemented a generalized N-puzzle solver that handles with no complexity cost a number of extensions of the puzzle where a search-based method would have at least to reinitialize. Our experience and results with the various stages of design and development are summarized here and are targeted toward the design and development of other specific solvers and of a more general bottom up solver. The current version of the solver may be found at: http://users.softlab.ece.ntua.gr/~brensham/NPuzzle/.

6.1 Representations

Our approach purports to treat classical “symbolic” problems in ways that are scalable, i.e., using problem solving methods for which the problems remain tractable as their scale/dimension rises. For this, we study specific solutions in an attempt to

build “general methods for building special-purpose *ad hoc* systems” and not “general representations with general inferential procedures” [7]. The overall lesson from our study of the sliding puzzle is that a scalable algorithmic representation should not rely on the state-space concept because variants or extensions of the problem would need a reconstruction or recomputation of the state space. The representation should use the situated information of the problem (puzzle) in a way that is directly perceivable by the solver and manipulable (“*the world is its own best model*”, [9]) and all actions would be direct actions within this world (the puzzle). Such representations have been found to be recursive to allow straightforward scaling (puzzle within puzzle) and this is very much dissimilar to compressed state-based representations such as “patterns” [6]. The latter are used to channel search away from non-optimal search subspaces rather than toward quick solutions and they do not introduce any other structure to the state-space, even if they are used for associative recall as in the original work by Levinson et al. [10] The pattern databases operate as a meta-level search space presenting the same inefficiency features at several orders of magnitude below usual search spaces, so that various extensions attempt to speed up or further channel the process [11][12]. On the contrary, our representations are intended to be of the order of complexity of the physical (symbolic) organization of the problem at hand and do not rise exponentially as the problem gets more complex.

6.2 Planning

A scalable system should support planning at a higher level of representation (such is in our case study, the planning about the order of placement of blanks). Planning sequences could be produced by search, even by local evolutionary search if possible (as could be the case of the corner sequence). However planning of detailed long sequences should be discouraged and it actually would hardly work, because, in general, problems (especially puzzles) are not “flat” enough for planning to be able to solve efficiently. Thus the subgoal formation and evaluation [6] is misleading from a cognitive standpoint in the sense that subgoals do not have a hierarchical relation to the overall goal, but they represent potential milestones and not subtasks of lower complexity. Rather than that, planning should be used to coordinate the various stages of lower-complexity reasoning [13].

6.3 On scaling and development

We are not studying how the algorithmic solution appeared in the first place (the Stepping Stone system [14] finds the subgoal sequence that corresponds to more or less the algorithmic solution, but it does not identify this as an algorithm). We have shown that for scaling to be possible, i.e. for complexity not to rise in case of extensions, a number of cognitive skills should be developed in humans and implemented in human-like solvers. These skills include recursion, minimal planning, mental rotation (or other re-representation) and problem decomposition and should replace detailed planning and search that should remain limited. The representation of the problem (puzzle) is not state space or graph based, but it is rather the geometrical

information of the puzzle itself. Another important result is that to be able to talk about scalable human-like problem solving, the above and other necessary skills have to be available as primitives within the problem solver and/or within the programming system (language or environment) but, unlike Korf's macro operators, they shouldn't always apply. We have found extremely difficult to implement programmatically these "skills" and "representations" in a way that is as elegant as one would expect from an AI system for a symbolic problem of the sort and we have been obliged to invent various programming tricks that did our job without for the least solving the general (meta-)programming problem. This observation about a cognitive skill based programming toolkit constitutes a different research direction that we plan to undertake together with the study of other puzzles some of which are in process [15].

References

1. Nilsson, N.: Principles of artificial intelligence. Morgan Kaufmann, San Francisco (1980)
2. Marr, D.: AI, a personal view. MIT AI Lab AI Memo 355 (1976)
3. Korf, R.E., Taylor, L.A.: Finding optimal solutions to the twenty-four puzzle. In: Proceedings AAAI, 286—291, AAAI Press, Portland (1996)
4. G. Gigerenzer, P.M. Todd and the ABC Research Group: Simple heuristics that make us smart. Oxford University Press (1999)
5. Aisbett, J., Gibbon, G.:A cognitive model in which representations are images. Journal of Cognitive Systems Research, 6, 333-363 (2003)
6. Culberson, J.C., Schaeffer, J.: Pattern databases. Computational Intelligence, 14(3), 318-334 (1998)
7. Bylander, T.: Tractability and artificial intelligence. Journal of Theoretical and Experimental Artificial Intelligence, 3, 171-178 (1991)
8. Loyd, S.: Mathematical puzzles of Sam Loyd. Dover, New York (1959)
9. Brooks, R.A.: Intelligence without reason. Artificial Intelligence, 47, 139-159 (1991)
10. Levinson, R., Beach. B., Snyder, R., Dayan, T., Sohn, K.: Adaptive-predictive game-playing programs. Journal of Theoretical and Experimental Artificial Intelligence, 4, 315-337 (1992)
11. Holte, R.C., Newton, J., Felner, A., Meshulam, R., Furcy, D.: Multiple pattern databases, International Conference on Automated Plannning and Scheduling (2004)
12. López, C.L.: Multi-valued pattern databases, European Conference on Artificial Intelligence (2008)
13. Pitrat, J.: La Métaconnaissance. Hermès (1990)
14. Ruby, D., Kibler, D.: Learning subgoal sequences for planning. In: Proceedings IJCAI, 609—614, AAAI Press, Portland (1989)
15. Tzafestas, E.: Sudoku intelligence. In preparation