Single-Player Games: Introduction to a New Solving Method Combining State-Space Modelling with a Multi-Agent Representation

François Van Lishout   Pascal Gribomont

University of Liège, Montefiore Institute, B28, 4000 Liège, Belgium
{vanlishout, gribomont}@montefiore.ulg.ac.be

Abstract

Classically, games are described as search problems. Each game situation is considered as a node in a graph. The arcs represent legal moves from one position to another. Solving a single-player game requires to solve a state-space problem, i.e. find one path that leads to a solution-state of the game. The heart of this paper consists in exploring the idea that most single-player games can also be modelled as multi-agent systems. The agents are no longer the players of the game as for multi-player games, but primitive game elements depending on the particular game. Furthermore, instead of facing each other, the agents collaborate to achieve a common objective. This new representation leads to new interesting resolution techniques, even more when both modelling methods are combined. It is demonstrated on the game of Sokoban, a challenging one-player puzzle for which mankind still dominates the machine.

1 Introduction

Games have always fascinated mankind and attracted the attention of the AI research community. Writing game-playing programs is not just a diverting activity, it also has many applications in real-life problems. Some people consider even life as a big game, where each living creature tries to maximize its well-being.

In many games, the machine has become stronger than the best human players. Machines have already beaten the human World Champion in famous games like Checkers, Chess, Scrabble and Othello. However, mankind has not been humbled by chips in all games. The best human players are still stronger than computers in games like Go, Poker, Chinese Chess and Hex [8].

This paper introduces a new solving method for single-player games, which will be demonstrated on the game of Sokoban. This choice is not innocent. First, it is a one-player puzzle which has not been solved yet by the AI community. Furthermore, man still dominates machine in this domain. For now, the best documented solver called *Rolling Stone* uses single-agent search techniques with a lot of problem-dependent improvements, and is able to solve 59 problems of a difficult 90-problem test suite [4].

Second, an unlimited set of different starting positions can be created by varying the size and the difficulty of the component problems. Different solving methods are therefore easy to compare, as there will always exist a test-set that can highlight the limits of the solving strategies. In this paper, we will compare our results to *Rolling Stone*'s ones on the same 90-problem benchmark.

Finally, Sokoban is a very challenging game. The main reason for this is the size of the search-space: it has been estimated at 10^{98} for 20×20 Sokoban mazes [3]. Other reasons are the large branching factor, the presence of deadlock states, the fact that the problem is PSPACE-complete, the length of the solutions, etc. For more details about the properties that make Sokoban a challenging research domain, consult [1, 2, 3]. Let us just add that Sokoban is not only challenging, it is also of practical interest as an instance of robot motion planning problems [2].

This is still an ongoing research, and further problem-dependent enhancements in the implementation are planned. On the other hand, problems that are unsolvable by *Rolling Stone* with single-player methods without other enhancement become solvable with the new strategy presented here.

2 The Game of Sokoban

A Sokoban maze is a grid composed of unmovable walls, free squares, exactly one man, and as many stones as goal squares. The player controls the man and the man can only push stones (not pull). Furthermore, only one stone can be pushed at a time. The objective of the game is to push all stones on goal squares.

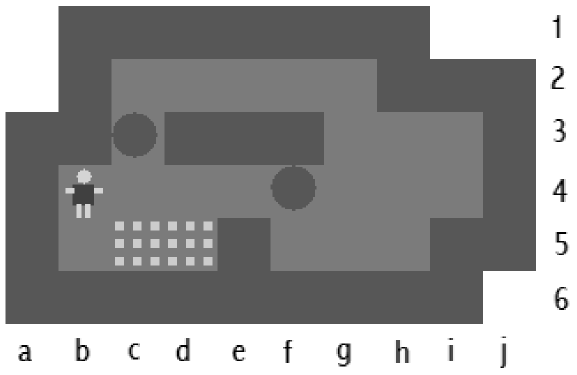

Figure 1: A Sokoban maze and a particular solution: b4-c4-d4-e4-f4-g4-g3-g2-f2-e2-d2-c2-c3-c4-b4-b5-c5-c4-d4-e4-f4-g4-g3-h3-i3-i4-h4-g4-f4-e4-d4-e4-f4-g4-g3-g2-f2-e2-d2-c2-c3-c4. The corresponding stone-moves notation of the solution: f4-g4-h4, c3-c4-c5-d5, h4-g4-f4-e4-d4-c4-c5.

For a better understanding of the game, a solution to the problem of Figure 1 is given. It is first described as an ordered list of coordinates of the squares that the man must pass by, to bring all the stones on goal areas (starting with the initial square of the man). A shorter notation giving only the lists of the stone-moves is also provided. Note that this notation makes the implicit hypothesis that each stone-move of the solution is valid, i.e. that the man can reach the square adjacent to the stone to effectively make the push.

State-of-the-art solvers model Sokoban as a state-space problem. The states of the graph are all the possible states of the game which can be obtained by varying the position of the man and the stones. The arcs represent legal one-square stone-moves from a position to another. *Rolling Stone* uses the Iterative Deepening A* (IDA*) [5] as basis for exploring the state-space.

In [4], the fact that only trivial problems can be solved by using the IDA* without other enhancements is demonstrated (even with a clever heuristic function, no maze of the difficult 90-problem test suite used can be solved). A lot of enhancements like transposition table, move ordering, deadlock tables, pattern search, and further problem-dependent improvements where implemented to achieve good performances (59 problems solved). However, this strategy seems to have reached its limits as the most difficult instances are still far from being solved by such methods.

3 The New Multi-Agent Modelling Approach

The idea of the new multi-agent modelling approach is that every stone of the maze can be seen as an agent whose aim is to reach one of the goal squares, and the global goal is to find a solution for which everyone achieves his objective. It is also possible to impose some kind of optimality like minimizing the global number of agent moves. In this view of the problem, the man is only a puppet which can be called by the stones when they want to be pushed.

The classical state-space techniques can be reused in this approach, as tools which can be used by agents to solve sub-problems of the game. Agents can for example use the A* algorithm [6, 7] to determine if they can reach some of the goal squares without needing the collaboration of the other agents. However, an agent that is solvable cannot start moving without the permission of the other agents. Otherwise, it could reach a goal square but generate a configuration in which other agents are in deadlock states. The difficulty of the problem is therefore reported to the communication process between agents.

3.1 Solving a Particular Subclass of Sokoban Problems

3.1.1 Definition of the Subclass

Let us define a particular subclass of Sokoban mazes which have been completely solved by the new method with a rather simple protocol. To be more precise, we will write a protocol that solves mazes if (but not only if) they belong the class. We can thus apply this protocol to any Sokoban maze but we have the assurance to find a solution only if it is in the class. Intuitively, a maze is in this class if the stones are solvable one by one. To be more precise, the maze must satisfy the following conditions:

- *Goal-ordering-criterium:* it must be possible to determine in advance the order in which the goal squares will be filled without introducing deadlocks, independently from the position of the stones and the man.
- *Solvable-stone-existence:* it must be possible to bring at least one stone to the first selected goal square without having to move other stones.
- *Recursive-condition*: for each stone which satisfies the previous condition, the maze obtained by removing that stone and replacing the selected goal square by a wall must also by in the class.

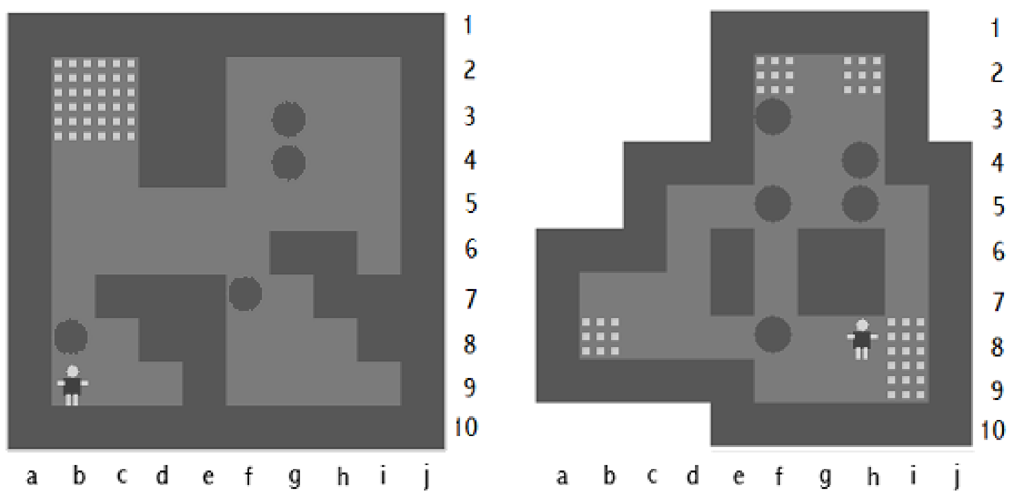

Figure 2: The Sokoban maze on the left side is an example of maze that is in the defined subclass. The one on the right side is a counterexample.

The maze on the left side of Figure 2 satisfies all the conditions:

- Filling the goal squares in the following order is adequate: $b2$, $c2$, $b3$ and finally $c3$.
- The $b8$ stone is the only one that can reach the square $b2$ without requiring other stone-moves.
- The maze obtained by removing the stone $b8$ and replacing the square $b2$ by a wall is obviously also in the class. This comes from the fact that each remaining stone can reach each remaining goal. The order in which the stones will be solved can thus be chosen freely.

The maze on the right side of Figure 2 is not in the class. It is a good compilation of non-satisfied conditions. The goal squares are divided into several goal areas and it is not possible to determine in advance the order in which the goal squares should be filled without considering the position of the stones and the man. Furthermore, in the initial position, only one stone can reach some goals without other stone-moves (the stone $f8$ can reach $b8$ and $i8$). However, removing the stone and replacing the square $i8$ by a wall would make the goal square $i9$ unreachable. Even by replacing the square $b8$ by a wall, the corresponding maze would still not be in the class. Indeed, none of the four remaining stones can reach a goal without other stone moves.

3.1.2 Protocol for Solving the Mazes of the Subclass

Due to the *recursive-condition* of the defined subclass, solving an agent cannot introduce a deadlock situation. This facilitates the elaboration of the algorithm. Indeed, we do not have to write a communication protocol that an agent must initiate when he can reach the first goal square to determine if it could cause damage to other agents. From this, a simple sequential program can be used for finding a non-optimal solution:

1. If no agent (stone) exists, the maze is already solved and the solution path is the empty list.
2. Else, select the first goal square.
3. Select an agent that was not chosen before and use the A* to compute if it can reach the selected goal square.
    - If it cannot, return to step 3.
    - If it can, move the agent to the goal square, replace the goal square reached by the agent by a wall and delete the agent. Go back to step 1. The solution path is the solution path of the deleted agent appended to the recursive solution path produced by the result of step 1.

In the first version of *Talking Stones* presented here, a very simple rule for ordering the goal squares has been chosen. The first goal square will be the one that has the maximum adjacent walls (in case of a tie any of them). The next goal squares are chosen by following the same strategy, but by considering that the previously chosen goal squares are walls. Thus, the goal squares which can only be reached from a single adjacent square will be filled first and we will avoid to block their entrance. This simple criterion has been chosen because it is easily implemented and works for most of the mazes that satisfy the *goal-ordering-criterion.* However, counterexamples can be constructed and more complicated ordering rules will be considered in later versions.

This protocol is easy to modify for obtaining shorter solutions at the cost of more computation time. In step 3, the first solvable agent will no longer be chosen directly. Instead, the solvability of all the agents will be computed and the one that has found the shortest path to the goal square be selected. By this means, a situation in which a stone has to make a detour around another stone to reach a goal square cannot occur anymore. The stone that would have been on the way would have already been solved. This strategy does not guarantee optimality (counterexamples can be constructed easily) but practice shows that in most real cases it leads to optimal or near-optimal solutions.

The termination of the algorithm is assured for mazes that are in the subclass, as the *solvable-stone-existence* condition assures that it will always exist at least one agent that can reach the first goal square at step 3. The mazes of the subclass can thus all be solved by this protocol.

The algorithm could also find an answer for mazes that are not in the class. Consider for example a maze for which the stones are solvable one-by-one but only in a precise order. The algorithm could try this order first by chance and find the solution.

It is easy to modify the protocol so that it terminates also for mazes that are not solvable by this protocol. We can simply answer that no such easy multi-agent exists when no more agents that have not been selected before can be chosen at step 3.

3.2 Results

Only one problem of our 90-problem test suite is in the subclass just defined (problem 78) and can be solved by this protocol. This is not surprising, as problems that are directly solvable stone by stone are uncommon in difficult benchmarks. Instead of elaborating other protocols for solving more general problems with the pure new multi-agent modelling approach, another idea has been developed.

As it is often the case in AI, trying to understand how the human player solves problems helps to find new algorithms. In the case of Sokoban, one of the talents of the human player consists in recognizing very soon in the resolution process that he can reach a configuration that is easy to solve (stone-by-stone). This suggests a new solving method for difficult games.

4 Embedding the Multi-Agent Modelling Approach into a Classical State-Space Algorithm

The method consists in using a classical state-space algorithm, but one in which the nodes whose corresponding state of the game is solvable by the new multi-agent modelling approach are defined as success nodes. This means that when the search reaches such a node, the search terminates successfully. The solution is then obtained by appending the solution path found by the state-space algorithm to the solution found by the multi-agent modelling approach.

The offspring of success nodes are no longer reachable and can be considered to have been pruned out of the state-space. In practice, the size of the state-space will decrease substantially. On the other hand, more computation time is needed at each node as the multi-agent modelling approach is called for each node to determine whether it is a success node. However, the time lost by these calls is largely compensated by the time won by having less nodes to visit. Our program *Talking Stones* implements this idea.

4.1 Choosing the Right State-Space Algorithm

The size of the search space remains huge for difficult problems. As the branching factor is rather high, a good memory management is required. Therefore, an iterative deepening algorithm is a good choice. We have thus to choose between the pure iterative deepening algorithm and the IDA* algorithm. The latter is naturally the best choice if we can find a good heuristic.

However, the heuristics commonly used for games are functions that try to minimize a particular distance to the objective. For Sokoban, it is generally the distance between the stones and the goals (using a minimum matching algorithm to assign the stones to the goals). This does not help much in the context of the new method. Indeed, almost every Sokoban mazes start in a configuration where the situation seems to be nearly blocked. The initial strategy consists therefore in finding a few moves to make more space for the man. Those moves have no reasons to be moves that diminish the global stone-goal distance. Defining a good heuristic function is thus far from trivial. For this reason, we have simply used the original iterative deepening algorithm in our first version of *Talking Stones*. We decided to limit the number of generated nodes to 20000.

4.2 Results

The algorithm seems rather naive; it simply tests all the possible moves until a position in which the stones are solvable one-by-one is reached. Surprisingly, this coarse strategy is already able to solve 9 problems of the 90-problem test suite.

Problem	Generated Nodes	Depth of the multi-agent solution	Time
1	76	3	6 sec
2	7155	4	8 min 23 sec
3	31	2	5 sec
5	67	2	25 sec
6	2849	4	4 min 20 sec
51	972	4	31 sec
54	2761	3	19 min 4 sec
78	0	0	< 1 sec
82	173	3	12 sec

Table 1: Results obtained by Talking Stones on the 90-problem benchmark

4.3 Comparison with ***Rolling Stone***

Rolling Stone has demonstrated that a pure IDA* approach without other enhancements cannot solve any problem of the benchmark and that immense progress can be obtained with them. Implementing some of those enhancements within *Talking Stones* looks thus promising for solving more instances.

One enhancement needs a special comment: Move Ordering. In *Rolling Stone*, moves which preserves the *inertia* of the stones are favored. This means that if the previous move was performed on a particular stone, moving this stone again will be considered first. From this, if *Rolling Stone* reaches a node that is solvable by the multi-agent modelling approach, it will find a solution similar to the one found by *Talking Stones* (solving the stones one-by-one, but possibly in another order).

However, the big difference is that our program will in this case require much less time and space to find the solution. Indeed, if we consider a maze composed of N stones, *Talking Stones* will simply call the A* algorithm at most N times to find a first solvable stone, $N - 1$ times for a second one, and so on. This gives a complexity of N^2 calls to the A* algorithm. Each call is relatively fast as the size of a Sokoban maze is small and so is the number of possibilities. To be more precise, for a 20×20 maze the stone can be placed on at most 18×18 squares (the borders are walls). A stone can move in at most 4 directions. The number of possibilities is thus clearly bounded by $18 \times 18 \times 4 = 1296$. A smaller bound can even by found by considering that a stone cannot move if it is in a corner and can at most move in 2 directions if it is against a wall.

An implementation based on the IDA* will do much more. At each step, it will generate all the possible moves for the N stones. For each position that can be obtained by performing one of those moves, it will compute its heuristic distance to the final goal. Then, it will select the most promising move not tried so far (in case of a tie one that preserves the inertia) and continue from that point. As the branching factor of the game of Sokoban is important, the IDA* algorithm will rapidly have to keep a large amount of candidate nodes to expand in memory. In the case of a maze solvable stone-by-stone, this is not needed as most of the candidate moves will never been tried.

5 Generalization of the Method

We can decompose the new solving method for difficult single-player games in three layers:

- The high-level layer is a classical state-space algorithm where a node is a success node if the medium layer can solve it. The choice of the algorithm depends on the precise game. If the branching factor is important an iterative deepening algorithm is appropriate. The IDA* should be preferred to the pure iterative deepening algorithm when expert-knowledge of the game is provided and a good heuristic function can be constructed. If the branching factor is small, the memory is not critical and the A* algorithm is indicated. When no good heuristic function can be constructed, the breadth-first search algorithm is the best suited one at this level. Consult [9] for more information on the most adapted state-space algorithms according to the branching factor.

- The medium-level is a protocol based on a multi-agent representation of the game. The agents are primitive game elements depending on the particular game. The agents have to communicate together to find a common solution. They can use the algorithms of the low-level as tools for solving sub-problems.

- The low-level is a set of algorithms for solving subproblems of the game. Classical state-space algorithms can be used but not exclusively.

We believe that it is possible for almost all games to determine primitive game elements that have to reach some goal. In puzzles like the 24-tile puzzle, the agents could be defined as the tiles. In this representation, each tile aims to reach its final destination but cannot move without altering the position of other agents. In the game of Sokoban, all the agents are instances of stones of the maze and have thus the same characteristics. For other games however, we could define agents that have their own personality. For the game of solitaire for example, the agents could be the 52 cards. Each agent is now unique. Note that for such imperfect information game, we must consider that only a subset of the agents is visible. The other agents can thus be seen as being in an unknown queue, waiting for entering into play. The new multi-agent modelling method could also be extended to multi-player games. In this case, teams of agents would face each other. Take the game of Go as an example. Here the goal of the white and black teams of stones would be to control as many areas of the game as possible.

6 Conclusion and Future Work

In this work we have presented a new modelling method for single-player games. Our idea has been to model the game as a multi-agent system where the agents are primitive game elements depending on the particular game. We have demonstrated the method on the game of Sokoban. It is important to note that the multi-agent notion which has been introduced is only conceptual and that it does not imply multi-agent programming. Our program is a solver for a single player puzzle. The method that has been presented requests a central solver to decide the order of the stones to be solved. This work presents thus solely a new way to view the problem which leads to new interesting resolution ideas.

We have defined a particular subclass of Sokoban mazes that has been completely solved by a protocol based on our pure multi-agent representation. It is of course possible to write an algorithm which determines efficiently if a maze is in the class. This has not been done here since it was not necessary. Our protocol solves indeed all the mazes of the subclass but can also solve some out of it. Furthermore, we have discovered that even when a Sokoban problem is not in this very particular class, it can often become so after a few moves.

We propose thus a new method. It consists in using a classical state-space algorithm, but one in which the nodes whose corresponding state of the game are solvable by the multi-agent modelling approach are defined as success nodes. This means that when the search reaches such a node, it terminates successfully. The solution is then obtained by appending the solution path found by the state-space algorithm to the solution found by the multi-agent modelling approach. Thus, we have not to test if a maze is in the class, but merely search for a configuration that is solvable by an algorithm based on a multi-agent representation.

At the time being, our program *Talking Stones* solves only 9 mazes of the benchmark whereas *Rolling Stone* solves 59. However, the latter is based on the IDA* algorithm with a lot of really interesting problem-dependent enhancements. These are presented in [4] and it is well explained why each of them contribute to a substantial decrease of the search-tree size. On the other hand, the fact that no problem of the benchmark can be solved with the pure IDA* approach without these enhancements, even with a clever heuristic, is also demonstrated in [4]. We have not implemented these enhancements yet and we plan to inject them within our new method in future works. As *Rolling stone* has enjoyed tremendous progress by adding them to its initial pure IDA* approach (from 0 mazes solved to 59), we hope to benefit from the same kind of progression.

References

[1] J. Culberson. Sokoban is pspace-complete. In *Informatics 4 Fun with Algorithms*, pages 65–76, Waterloo, 1999. Carleton Scientific.

[2] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry: Theorie and Applications*, 13(4):215–228, 1999.

[3] A. Junghanns. *Pushing the limits: New developments in single-agent search.* PhD thesis, University of Alberta, 1999.

[4] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(2):219–251, 2001. Republished in [8].

[5] R.E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 26(1):35–77, 1985.

[6] N.J. Nilsson P.E. Hart and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] N.J. Nilsson P.E. Hart and B. Raphael. Correction to [6]. *SIGART Newsletter*, 37:28–29, 1972.

[8] J. Schaeffer and J. van den Herik, editors. *Chips Challenging Champions.* Elsevier, 2002.

[9] P.H. Winston. *Artificial Intelligence.* Addison-Wesley, third edition, 1992.