The Sokoban Challenge: An Analysis on Past, Present, and Trends in Algorithms and Heuristics for Automatic Solving of Sokoban Problems

Juan Antonio Martín Checa

Computer Science Department
Campus Teatinos
University of Malaga
29071 Málaga, Spain
jamcheca@telefonica.net

Abstract. Games have been since far long a matter of curiosity for Humanity, Most of known games are nowadays immediate or easy to solve by mankind and/or computers. However, some of them still remain a mystery, a challenge to be accomplished. Sokoban is one of these fascinating games from which we still have much to learn. This work introduces the game of Sokoban, analyses the different algorithms used so far in an attempt to solve it, explains the heuristics applied, and its learning, and last, gives an overview of the current situation as well as upcoming research lines oriented to solve the game.

Keywords: Artificial Intelligence, Deadlock, Decomposition, Difficulty, Games, Goal Cuts, Goal Macros, Goal Scheduling, Hash Tables, Hierarchical Learning, IDA*, Minsmatching, Move Ordering, Overestimation, Pattern Searches, Planning, Rapid Random Restart, Relevance Cuts, Rolling Stone, Search Strategies, Single-Agent Search, Sokoban, Talking Stone, Tunnel Macros.

1 Introduction

Games have always been seen as perfect subjects for exploring human reasoning capabilities within the field of Artificial Intelligence [2]. As explained by remarkable AI researcher Susan Epstein, there are two main reasons why research on games should continue: human fascination with game playing, and the fact that some games still remain unsolved [1]. One of these games, and focus of this paper is called Sokoban.

The term *Sokoban*, which in Japanese means *pusher* [8], is commonly translated into English as *warehouse keeper* or *storeman* [9]. The term also applies to a game created in1981 by Hiroyuki Imabayashi [12], which worldwide success relies on its simple rules coupled with the significance of the player's intellectual challenge [5].

1.1 Rules of the game

As described in [12], “the object of Sokoban is to push all stones (or boxes) in a maze […] to the designated goal areas.” The stones can only be pushed (not pulled) by the *warehouse keeper,* for what he has to get behind them. Also, the *storeman,* who equals the stones in size, is only capable of moving one stone at a time, making the game even more challenging [9].

2 Applied algorythms

The problem of solving Sokoban is very attractive to AI researchers because of its similarity with the design of a robot responsible for moving boxes in a warehouse. The main issue, however, when solving Sokoban puzzles (which have been demonstrated to be *PSPACE-complete*) is their *NP-hard* complexity [9].

Traditionally, the most extended approach to solving single-player games has been seeing them as *state-space* problems. This state-space is structured as a graph in a way that each of its nodes is a game state. A node’s successors represent those game states reachable in one *movement* (or *push*, in the particular case of Sokoban). Then, the natural method to solve the problem was relying on any of the so-called *single-agent search algorithms*, such as *IDA* (Iterative Deepening A*)* [2].

However, *classical search algorithms* such as IDA* have been proved not to be capable of solving non-trivial Sokoban problems by themselves. The key factor here is the level of complexity of the specific Sokoban problem under study.

A number of factors affect directly the complexity of Sokoban problems. First, the *branching factor,* meaning the number of options given a game state, is extremely high (probably over 100). Second, and because of this, the final *solution* can be particularly long. Third, the state-space can be unmanageable in size. Forth, is that unlike in other games, *deadlocks* (game states from which the final solution game state is not reachable anymore) do exist in Sokoban. Last, the enormous *variety* of problems makes it necessary to adopt different strategies when solving them [2].

For all of the above, in order to be capable of solving complex Sokoban games, the IDA* heuristic search algorithm is combined with a a number of techniques which make use of domain-specific knowledge [9].

3 Heuristics and learning

Once we have commented on the IDA* algorithm, we will introduce two worldwide recognized programs for solving Sokoban problems: *Rolling Stone* and *Talking Stone,* analyzing the heuristics and strategies used on each.

3.1 *Rolling Stone*

Rolling Stone, developed by the University of Alberta, is considered as the best documented Sokoban solver ever. Based on the IDA* algorythm, it is capable of solving up to 59 complex problems from a pool of 90, when coupled with a set of domain-specific enhancements.

The first is *minmatching*, a lower bond for the number of pushes needed to solve the maze, based on an algorithm that solves the *minimum cost perfect matching in bipartite graphs*. *Hash tables* are also used to eliminate identical subtrees, while *move ordering* produce savings in the last iteration, and *deadlock tables* remove parts of the tree leading to deadlocks. *Tunnel macros* and *goal macros* reduce the tree depth. On their side, *goal cuts* and *relevance cuts* reduce the branching factor. *Pattern searches* is of special importance since they produce vast reductions in the search tree. *Overestimation* is also oriented in the same direction. Finally, *rapid random restart* avoids getting stuck in critical situations [12].

3.2 *Talking Stone*

Talking Stone, is a more recent Sokoban solver, created in the University of Liège. Unlike *Rolling Stone, Talking Stone* is not based on a *single-agent* algorithm such as IDA*, but on a *multi-agent* representation of each specific Sokoban problem [2], adopting a new approach based on the concepts of *decomposition, goal scheduling,* and *leaning from deadlocks*.

Decomposition refers to dividing the final solution into a number of sequences of pushes, each of which can be subdivided into two phases: *extrication* (which implies the move of different stones) and *storage* (pushing the stone to its final location).

*Goal schedulin*g determines the order in which the *goals* (destinations) will be filled by their corresponding stones. Basically, it could be seen as a permutation of the list of the ‘n’ *goals* associated to the problem. There are two types of goal scheduling: *effective scheduling* (guarantees that the goals can be filled) and *consistent scheduling* (assures that no goal will ever made unreachable by those stones located on filled goals). The *goal scheduling algorithm* used in *Talking Stone,* starts from a situation in which all goals are filled by stones (the final solution game state) and evacuates all of those stones annotating the pushes needed. Once all the stones are evacuated, the algorithm takes the reverse order to solve the problem [2].

Learning from deadlocks is still another essential characteristic of *Talking Stone,* based on a classical search algorithm, leading to a significant reduction on the branching factor.

Finally, the solving protocol consists of four search functions. The first function in in charge of calculating the goal scheduling based on the initial state, which is passed to the second function, whose objective is filling all the goals as described in the goal scheduling. A third function is the one which actually fills the actual goal. The last function controls potential deadlocks that may arise.

4 Current status

As we said before, *Rolling Stone* is capable of solving 59 of the 90 Sokoban problems in the reference suit. However, it is to be highlighted the fact that this Sokoban solver (and so, its heuristic approach) seems to have reached its limits. Indeed, no further research is being done on it.

Talking Stone, on its side, is currently capable of solving 54 of those problem, this is, 5 less than *Rolling Stone,* with the difference that further research regarding the rearrangement of the disposition of the goals could push that number up to 61, and thus, defeating *Rolling Stone* as the best Sokoban problem solver.

Nowadays, the current situation is that some of those 90 Sokoban problems still remain unsolved by either humans or machines [11]. Recent research [3] focuses on the notion of difficulty in an attempt to understand how humans solve Sokoban puzzles. Many online sites such as [6] offer executable programs for solving Sokoban problems, while others such as [4] and [10] allow curious users to enjoy this fascinating game. Even the *Apple Store* offers Sokoban for the *iPhone* [7]. There is no doubt that Sokoban is more *alive* than ever.

References

1. AI Topics/Games, http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/Games
2. Demaret, Jean-Nöel. Hierarchical Planning and Learning for Automatis Solving of Sokoban Problems. University of Liège, Liège.
3. Jarusek, Petr. Human Problem Solving: Sokoban Case Study. Masaryk University, Brno (2010)
4. Juegos de Sokoban , http://www.juegosjuegos.com/juegos_reflexion_sokoban_1.shtml
5. Junghanns, A. Sokoban: a Challenging Single-Agent Search Problem. University of Alberta, Edmonton.
6. Resolver Sokoban, http://jose-juan.computer-mind.com/jose-juan/Resolver-Sokoban.php
7. Smart Sokoban Free para iPhone, iPod touch y iPad en la App Store de atunes, http://itunes.apple.com/es/app/smart-sokoban-free/id294794649?mt=8
8. Sokoban, http://www.rodoval.com/heureka/sokoban/sokoban.html
9. Sobokan - Wikipedia, http://en.wikipedia.org/wiki/Sokoban
10. Sokoban 3- Juego de Análisis y Estrategia para Lograr la Secuencia Óptima de Movimientos, http://www.jugarjuegos.com/juegos/java/sokoban3/index.htm
11. Takes, F.: Sokoban: Reversed Solving Bachelor Thesis. Leiden University, Leiden (2008)
12. The University of Alberta Games Group, http://webdocs.cs.ualberta.ca/~games/