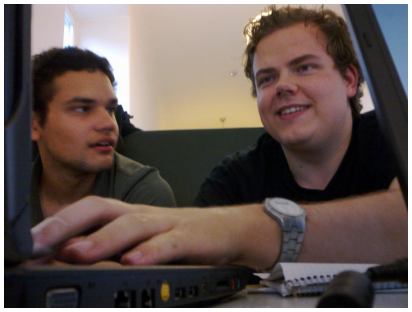
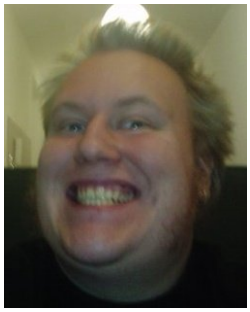
We Called Him Jens

The Solver of Sokoban

JOHANS HJÄLTAR
TIM LENNERYD, RISHIE SHARMA, JACOB NORDGREN, JOHAN ÖHLIN

倉庫番

Abstract

Contents

Chapter 1

Background

1.1 About Sokoban

Sokoban is a type of transport puzzle originally from Japan that involves pushing boxes into their goal positions on a maze-like board. It was first created in 1981 and has since been used both as a computer game and a testbed for artificial intelligence. While there have been a number of different implementations and variations of the game, the basics remain the same, the character is the same size as the boxes, and the boxes can only be pushed from the four different directions: up, down, left and right. To push the boxes the character needs to be on the opposite side of the box. That is, the boxes can not be pulled or pushed from the angles (see Fig 1).

(Fig 1. Sokoban board with arrows on how to push boxes. Or a better image. Maybe two pictures.)

Chapter 2

Theory

2.1 Complexity

The game of Sokoban belongs with a number of other games and puzzles in a family called Motion Planning Problems, or more specifically Motion Planning Problems with Movable Obstacles. The time and space complexity of the games and puzzles in this family vary, some being in P, some being NP. Sokoban however have been shown by for example Joseph C. Culberson to be P-Space Complete[1] . He has done this by showing that a Turing Machine can be emulated in linear time using a specially constructed infinite version of the puzzle. In this version, there are only a finite number of out-of-storage containers, and he uses a number of special constructions of the puzzle to emulate different parts of the Turing Machine. He then restricts the tape to finite length, which in turn shows that the puzzle is PSPACE hard. We will not go into this further, since the reasoning for this are fairly advanced. We refer to the report by Joseph C. Culberson instead, which can be found in the references[1].

Dorit Dor and Uri Zwick has concluded that the Sokoban puzzle is in fact NP hard in terms of time complexity.[2] They do this in a similar, but not identical way to Culberson's proof above. They make special constructs within the rules of the game to make the game behave a certain way. This is to be able to reduce the problem to a known NP hard problem, which in the case of their proof is P3SAT, ie. the Planar 3SAT problem which is a variation of the original 3SAT problem. They do this first for a generalized version of the game, in where the pusher can push k blocks at once, and pull n blocks at once. In the case of their first proof, k = 5 and n = 1. By being able to reduce this constructed Sokoban game to 3SAT, they are then able to use this as a grounds to prove that the Sokoban game we are considering here is also NP-Hard, by making some changes to the proof and using k = 1, n = 0, which means that the pusher can push one block at the time, and pull 0 blocks. These proofs are however quite advanced, and if more info is wished for, the article used is in the references.[2]

2.2 Unique states and search space

The Sokoban game has a very large search space, as well as a very big branching factor, as the complexity analysis referenced above shows. This can be easily shown just by thinking of how one may define a single state. To be able to provide a unique state, as a basis for eliminating repeated states in the solver, all the box positions need to be included. But that's not all, the position of the pusher needs to be included as well, since the boxes may need to be pushed from a certain direction for the successive states not to result in a deadlock. If the pusher can get from his current position to that direction without touching a box, that is however enough to provide a unique state in terms of the theory.[3] This is however a fairly complex thing to keep track of, and it's usually just easier to keep track of the specific position of the pusher as well as the boxes, and call this a unique state.

This means that the branching factor of a specific board depends on the number of boxes, since every box can be in every position in theory, and also on the number of directions the player can move, which is 4 in Sokoban. A normal game with for example 6 boxes may then have a branching factor of 24, since every stone has 4 possible legal moves depending on the direction it's pushed. This means the search space will be huge, since in level k of the search space tree, there will be 24 000 nodes.[3]

2.3 Finding solutions

Due to the perfect information single-player nature of Sokoban, an informed search algorithm that is proven to be Optimal in such situations can be used, despite the search tree being very big. One example of such an algorithm is A*, an algorithm using best-first search and that will find the least-cost path from an initial node to a goal node. The initial node in this case is the untouched box-positions and the player position, and the goal nodes are the nodes where all the boxes are placed in the goals, regardless of the permutation among the boxes and the position of the player. A* is an extension of the Djikstra algorithm that achieves better performance by using certain heuristics, and can still be proven to be Optimal if certain constraints to the heuristics used are met.[4]

A* considers both the cost to get to a particular node in the tree from the root (initial node), as well as the estimated cost to get from the current node to the goal node. Due to the fact that the cost to get from current to goal is only estimated, heuristics must be used to do the estimation, which in turn places certain constraints on the heuristics, as mentioned above. The main constraint is that the heuristic (or combination of heuristics) may not overestimate the cost to get from the node to the goal. This is an important, since this means that the A* algorithm will never overlook the possibility of a lower-cost path. The algorithm is therefore optimal in the sense that it will find the least-cost path in terms of the costs placed upon the node transitions by either the heuristic or the problem itself.

That being said, this does not mean that A* is the best algorithm to use out of the box for Sokoban, since due to the nature of the game the search tree can be pruned in a number of ways by both simple heuristics and quite advanced ones, by finding deadlocks in different ways. This is discussed in more detail below. Many of the heuristics that can be implemented to find deadlocks can however be used with both A* and other search algorithms, since they mostly just prune the tree in different ways, which would make A* a very good choice even after all these heuristics were implemented, since A* is a search algorithm that has very good performance provided a good (more global) heuristic can be found to provide it with estimates.[4] Using a simple Move-Toward-Box + Push-Box-Toward-Goal heuristic could give A* a big edge over algorithms like depth-first and regular greedy-best-first (since greedy-best-first doesn't consider the distance already travelled).

2.4 Finding deadlocks

A deadlock in the sense of the game is a position in the game from which it is impossible to win. In the most obvious case, this is when a box is pushed into a corner and therefore cannot be pushed from any meaningful direction. But deadlocks can be much more subtle than that, containing a number of walls in different configurations as well as boxes. As soon as a deadlock is found, that part of the search tree can be pruned, which means that every single deadlock found will help in the effort of reducing the size of the tree.

As an example, if there are 2 boxes on a 10x10 board, then each of the boxes can be in one of a 100 different positions. The total number of positions for the two boxes are then 100*99 = 9900. Add to that the pusher, who can be in any one of the free positions, and we have 970200 unique states. Now, this is a completely constructed example, where the whole 10x10 board is open with no walls except around the edge. And if we only consider the 4 corners to be deadlocks, this means that we have 96*95 = 9120 positions for the boxes, and 98 for the pusher, to a total of 893760 unique states. Due to removing the 4 deadlocks we had considered, we have removed 970200-893760 = 76440 states.

In theory the tree could be reduced to only having valid (if long, in some cases) solutions, if all the deadlocks were to be found. This is an unreasonable expectation however, and would need far too much time and effort in even fairly small cases. The most naive and intuitive solution for finding deadlocks is to sit with a pen and paper, and write down all the different deadlocks that one can think of, be it corners, two blocks beside each other pushed up to a wall etc, but only a fraction of the deadlocks possible can be found in this way. That said, every single deadlock pruned away helps when it comes to finding a solution, but there are other ways. If one uses a pattern based method, deadlocks can be saved as patterns to a big database or table beforehand, which will allow for offline calculation of said deadlocks in for example a brute force manner. Using brute force prevents larger patterns due to the time complexity of brute force calculations, but smaller patterns (2x1, 2x2, 3x2, 3x3 etc)

can then be saved to a database and would then allow for a simple lookup where the positions of the walls, boxes, goals and the pusher within the area are being considered, and returns whether the position results in deadlock or not.

Chapter 3

Implementation

3.1 Algorithm Basics

The fundamental part of the algorithm is a depth first search that from each node, loops through the possible directions and chooses a direction based on two different criteria. The first criteria that is checked is whether it is possible to move in the direction at all, and this is done by enforcing a series of rules that are required for the move to be possible without violating the rules of the game. This criteria also checks that the exact same position haven't been evaluated by the solver previously, this to make sure that any repeated states are removed.

The second criteria that is evaluated is that of the cost to move in the requested direction, This is done by applying a series of heuristics and local optimizations to give a cost based on the attractiveness of a certain position. As an example, pushing a box into a corner is a very unattractive as every move after that will be a dead end, the box can no longer be moved out of the corner and therefore the game can't be won from that position.

3.2 Architecture

3.3 Local optimizations

Chapter 4

Result

Chapter 5

Performance comparisons of local optimizations

References The site citeseerx.ist.psu.edu is a repository-like site for science report abstracts and the reports themselves.

Bibliography

[1] J. C. Culberso, "Sokoban is pspace-complete," 1997.

[2] D. Dor and U. Zwick, "Sokoban and other motion planning problems (extended abstract)," 1995.

[3] A. Junghanns and J. Schaeffer, "Sokoban: Evaluating standard single-agent search techniques in the presence of deadlock," in *Advances in Arti Intelligence*, pp. 1–15, Springer Verlag, 1998.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, third ed., 2009.