Sokoban Solver

General Design

Since Sokoban is PSPACE-complete, the design of the solver had to be lightweight as the search space for non-trivial puzzles is huge. Thus, it was important to only save the necessary data for each Node as to not run out of memory during a complicated puzzle. Each Node N consisted of a state, the moves to get to that state, a pointer to its parent, and the cost of the path from the start node to N. A state *only* consisted of the location of the boxes and the player, as these were the only variables that could change between states (The walls and goals cannot be moved).

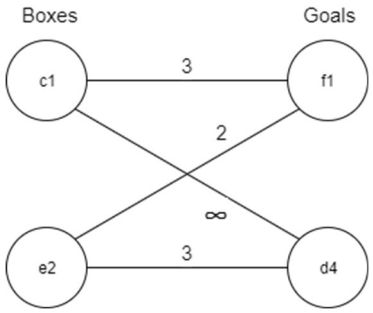
Algorithm

Keeping with the memory-saving design, the solver uses IDA* search. Like A* search, it is complete and optimal, but contrary to A* search, it has a polynomial space complexity. In each scan of the search, the memory required is only the path we are currently scanning without its surrounding nodes, as oppose to A* which must store the set of tentative nodes we would like to visit.

Heuristics

Min-Cost-Matching

	a	b	c	d	e	f	g
0	#	#	#	#	#	#	#
1	#		$			.	#
2	#	@			$		#
3	#						#
4	#			.			#
5	#	#	#	#	#	#	#



Boxes on	goals on	
	f1	d4
c1	**3**	∞
e2	2	**3**

For each puzzle, there is pairing between each box and goal. This pairing forms a complete bipartite graph. The problem then turns into the min-cost-matching or assignment problem (see). We need to find a perfect pairing of boxes and goals as to minimize the total number of steps required to push each box to its goal. In this solver, I use the Hungarian algorithm (see) to find the perfect pairing and return the minimum steps required.

The example above illustrates why the perfect pairing is so important opposed to the simple Manhattan distance. Although the box on e2 is physically closer to the goal on f1, the Hungarian algorithm does not choose that pairing since the box on c1 *must* go to that goal as it cannot reach the goal on d4. Therefore, the min-cost-matching for this is 6 steps.

Lastly, we must factor in the distance from the player to his nearest box. This is a just a simple Manhattan distance calculation resulting in 2 steps from b2 to c1. The total heuristic value of the state above is $6 + 2 = 8$. It is important to note that this heuristic is admissible and never overestimate the cost of the reaching the goal state.

Backtracking

When determining all possible valid moves in the current state, we do an additional check to see if the player is backtracking. Backtracking occurs when the player does a 180, either going from up to down or left to right. We can omit these moves from the possible move pool as the optimal solution will never have the player turn around (unless the previous move was a push).

Deadlock

Deadlocks occur whenever the state becomes 'unsolvable', or more explicitly, when it is no longer possible to push every box to a goal. Detecting a deadlock state ranges from being a simple check to extremely complex checks and possibly more computationally expensive than solving the puzzle itself. This solver only checks for the trivial deadlock when a box is pushed into a corner and can no longer be pushed.

Biggest and Smallest Problems Solved

I have included several test levels in the zip. The 'test_levels' directory contains some trivial levels that I made myself, including one that has no solution as to illustrate that the solver can recognize unsolvable levels. I have also included several levels of the well known microban2 level set under the 'microban2' directory.

Defining 'complex' in terms of sokoban puzzles is not a simple definition. A good metric I found was the more boxes and goals there were with the goal tiles being very close to each other (in a cluster), the more complex the puzzle. The difficulty in small area puzzles usually arises from the goals being close together, which means the distances from the boxes to their goals are quite similar. It also means that the solution probably requires boxes to be pushed *through* goal tiles, which is highly unfavourable.

- The biggest and most complex puzzle that I could find that it solved in under 5 minutes was microban2 level 16. This one was quite complex due the number of times a box had to be pushed *through* a goal tile in the actual solution.
- The smallest puzzle that I could find that could not be solved in under 5 minutes was microban2, level 18. It would consistently be solved in just over 5 minutes.

(Testing done on a laptop with an i7-6500U CPU @ 2.50GHz)

References

- Junghanns, Andreas. (1999). Pushing the Limits: New Developments in Single-Agent Search.