Documentation for Sokoban solving

using Beam Search and Learning Real Time A\*

Overview

For both algorithms I tried enhancing the heuristics in order to avoid pull moves. I made multiple observations during solving process which I am going to detail further.

For both algorithms, I tried two different heuristics regarding the distance between the boxes and the targets.

But first of all, the problem was, which target belongs to which box?

For the first draft of the algorithms, I tried mapping them randomly. The problem was, if I mapped them from the beginning, maybe there was no possible solution for that mapping. Thus, I mapped them randomly at each step when I was calculating the heuristic value of a state. To match the randomness, when having to make a choice between states with same cost, I chose randomly as well (for LRTA\* algorithm especially). Needless to say, the solutions varied a lot, and the number of explored states was huge.

Thus, I changed the mapping to be more intelligent and after some research, I used Hungarian algorithm, where the assigned cost was the shortest path between a box and a target (calculated using BFS algorithm). I did this mapping at the beginning and maintained it throughout.

Then, I observed that just taking into consideration the distance from a box to a target was not enough, because there were a lot of states with the same cost, states where the player had different positions. So I thought about adding to the cost, the distance of the path between the player and the boxes. At first, I added all distances between the player and the boxes, but it was not very productive because there were too many variables to take into consideration, so if the player got closer to a box he got farther from another and the total distance may have remained the same. Thus, I needed to concentrate on one box at a time.

So here came the discussion, which box to consider?

There were multiple variants to solve this, and I thought of two: either take into account the closest box at that specific moment, or I could set a predefined order to put the boxes in place.

Having a predefined order sounds useful because as I have researched, the sokoban solution more often than not consists of multiple sequences of moves applied to the same box. However, it’s hard to calculate an order good enough to be possible to be solved. Whereas, using the closest box at each moment was a more flexible approach, as the “followed box” could dynamically change, and the distance between the player and the box would still follow a decreasing curve.

That’s why I found that following the closest box was a better approach.

Beam Search

Beam Search is like a BFS, for which you maintain only the best k results on every level.

For beam search, using the closest box approach worked very well. Also, for this distance between the player and the box I used BFS, as I wanted a more accurate result, because here there was no interaction between player and box to be taken into account, like there is when pushing boxes to targets.

For the distance between the boxes and the targets I tried two different approaches, using Manhattan distance and BFS distance. From my experience, BFS distance gave better results, as it was closer to the reality (less states explored and most importantly less pull moves).

To improve the number of pull moves, I implemented a BFS variant that calculates paths in form of “tunnels”, that is paths where a box could be pushed by the player. This worked really well with beam search.

Also, for the same reason, I added to the heuristic a term that took into account the number of pull moves performed from the start state to current state. I used a factor of 10 \* no\_pull\_moves, in order to give more weight.

Another thing I observed, was that there were moments when on the path to the solution, a box that was placed on a target was moved again. So I thought how to avoid this. Thus I subtracted from the heuristic value of a state the number of boxes on the map for each box that was placed on a target. My idea was that this term I subtracted should be proportional with the total number of boxes on a map, rather than being a constant. This way, the issue was solved.

For beam search, the heuristic does no need to be admissible, so there were no worries about over evaluating the cost. Also, the exploration has the form of a tree with mostly k nodes on each level and the solution is the path from the root (start state) to a leaf.

Another important aspect was the value of k. A smaller k gives better performance, whereas a larger k gives the possibility of finding / finding a better solution.

Here is the analysis for different values for k. Note that a -10 number of pull moves means no solution was found.

* O imagine care conține text, captură de ecran, diagramă, Interval

  Conținutul generat de inteligența artificială poate fi incorect.K = 50

O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.

* O imagine care conține text, captură de ecran, diagramă, Interval

  Conținutul generat de inteligența artificială poate fi incorect.K = 25

O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.

* O imagine care conține text, captură de ecran, diagramă, Interval

  Conținutul generat de inteligența artificială poate fi incorect.K = 100

O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.

On super hard map, the expected behavior appeared, the larger the k, the better the solution.

However, on some easier cases, a k = 25 did unexpectedly well. The reason would be that the heuristic function is not that well informed, especially the Manhattan one where we see that the “proportions” we expected are not maintained. However, the number of pull moves are very close (at difference of one or two), in our game this difference is significant, but the heuristic was not projected to take this weight into account.

O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.Here are the execution time results, only for K = 50, as for other cases they are proportional to the number of explored states.

For further analysis, I chose k = 50, because the results were good enough, especially considering BFS distance and that all maps were solved.

LRTA\*

For LRTA\* I chose the same two heuristics, Manhattan distance and BFS distance.

At first, I ran LRTA\* just once, and I considered the path being all the explored states. However, LRTA\* has its learning component, where it explores a path and then it comes to the conclusion that it’s better to return to a previous path. Thus, I used a stack to eliminate the cycles in the path from the start state to the end one. This made the results comparable with the ones produced by Beam Search.

For the function that calculates the returning cost, I added 1 if the return move would be a push/normal move and 5 otherwise, in order to discourage returning by pull moves. 5 was chosen because I thought about avoiding a pull move, which usually means getting on the other side of the box, which in best case takes 5 moves.

With LRTA\* I saw that the algorithm had a hard time getting closer to any box, it would firstly explore the open space. Thus, I modified the weight of the distance between the player and the box, making it 2 instead of 1. This accelerated the results, but the cost would be that the heuristic may not be admissible. It would be admissible if we consider that, for every box, the player pushes it to the target and then it returns to the initial position and then gets to another box. Then the weight of 2 would be justified by the returning path. But that would imply an heuristic that would make the player return every time, which in this case is not implemented.

Moreover, I observed that Manhattan distance, although less informed and with worse results, would solve all maps, whereas BFS distance with tunneling did not. That is because BFS with tunneling may return “inf” cost when target is not reachable from specific state (that is it is not accessible without pull moves), and from there it’s more difficult to find a solution.

The test that would fail with BFS distance was a medium one, where I observed that the boxes were very close initially to each other. I figured out that the player would toggle between states when I used the “closest box now” approach. That is why I implemented the predefined order. It solved this problem, but there were other more difficult tests that would remain unsolved, so it was kind of a trade off.

Then I thought about running LRTA\* multiple times, because ideally it gets better by learning. It worked, sometimes it gave better results, but the curve of them was not increasing as I had expected. Most probably the problem comes from the heuristic function, that as I have said is not necessarily admissible.

These are the LRTA\* results after 5 iterations for each heuristic, storing the best result. For the number of explored states, I added them all for the 5 iterations, although not very suggestive, as there are iterations where no solution is found and the 30000 states limit is reached.

O imagine care conține text, captură de ecran, diagramă, linie

Conținutul generat de inteligența artificială poate fi incorect.O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.

O imagine care conține text, diagramă, captură de ecran, Interval

Conținutul generat de inteligența artificială poate fi incorect.

Execution time cumulates execution time for the five iterations. I have observed that solutions are usually found within 30000 states, so I stopped the algorithms at this point.

These graphics merely show that the solution usually contains multiple sequences of moves that target specific box, thus the number of pull moves is smaller when considering a predefined order. However, we can see there are maps that are not solved this way.

That’s why for future analysis I chose “closest box now” approach.

Comparison

Note that for lrta\* the number of explored states is the one taken for the best iteration. That is beacause convergence for lrta\* is not known. Even so, clearly lrta\* is much more computationally dependent, which can be seen from the execution time graphic as well.

O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.O imagine care conține text, captură de ecran, diagramă, Interval

Conținutul generat de inteligența artificială poate fi incorect.

On average, beam search finds better results. A reason for this would be that learning real time A\* is suitable for situations when the searching area is not know or maybe it changes dynamically.

Thus, for mazes of moderate dimensions like the ones used, beam search is a much faster approach.

As it implements a BFS, it finds a shorter paths that does not contain cycles, whereas LRTA\* is more of a DFS, which explores in a more unordered manner.