Using an Algorithm Portfolio to Solve Sokoban

By – Navneet Gupta, Sahil Singhavi, Abhay Bhardwaj

Abstract— As the world is giving a new birth daily to the technology and advancing it. The biggest role in advancing the technology is the artificial intelligence which is a science that makes computer think like people, act like people, think rationally and act rationally. As in every process we have a state where we need a machine to act like human and find out the solutions to come out of the state with his own rational power. Here in this paper we have taken the state of a game which is Sokoban which is a very interesting game where we have applied the algorithm.

Index Terms— deadlocked, branch factor

1 Introduction

As the world advances NP-hard problem become hard and hard problem become easy to solve by the computer and the human brain. The game Sokoban was first proven to be NP-hard in 1996 and then PSPACE-complete in 1997. The state we have taken have simple rules ,even easy level can require a lot of computation power to solve. To reduce that parallelization and we need good algo to solve the problem.

This work focuses on algorithm portfolios – a parallelization concept in which each processor solves the whole problem instance, using a different algorithm, random seed or other kind of diversification.

Sokoban

It is a puzzle game where each level consists of a 2-D rectangular grid of square. In which it is occupied by walls (static obstacle),boxes (movable obstacle), goal locations or the player. The goal of the game is to push each box into a goal location by controlling the player which can move in four directions (up, down, left and right). The player is only allowed to push a single box at a time.

2 Problem Modeling of Sokoban

To create the state of Sokoban puzzle the input we have taken is a text file which consist of walls, obstacle, boxes and the final position where the blocks to be placed which can be converted into a 2D map as shown in figure (1) below for understanding the map "#" means walls, "@" means the person who will move the boxes (Placer) , "$" means boxes, "." Means the destination of the boxes.

```
-------------Initial Map---
The map is as follows:
# # # # # # # # # #
#             #   #
#         # #     #
#   #       #     #
#                 #
# #               #
#             #   #
#   $ # @       # #
#     #         . #
# # # # # # # # # #
```

And the output will be in string form that represent the movement of the placer by which it can place all the boxes in place.

```
# # # # # # # # # #
*************** Solution Found ! *****************
ulllddrulurrrrurddldrr
greedy(manhatten): 2 ms
This Map works: (10Ã—10) walls: 10 boxes: [Point(8,3):251]
```

As the moves taken by the placer is one at a time. Since every state is determined by a move of the placer, a state potentially has four neighbors/successor nodes that can explore in the next move. The move that will be valid is when placer move to an empty place within the wall limit. Or th placer move and pushes a box to a new place or the destination. Both the placer and box can not violate the restriction of the walls.

As in the paper we are going to apply algorithms based on graph so we have to model our problem as a graph search problem which is composed by an initial state, actions, end state, goal test and the path cost. So we need to many constraints for our problem to set up. Lets begin,

1. Initial State:- it is a position of the placer and boxes.
2. Action:- the change of position in boxes and the placer.
3. Goal test:- the system to execute a matching between boxes and their targets or destination points.
4. Path cost:- the number of moves starting from the original state to current state.
5. End state:- this is the state that will come as the output as we are not using any bi-directional graph search algorithm.

As the state are defined which will be fitting in our puzle.

2.2 Algorithm discussed in the paper

The algorithm describe in the paper is of a parallelization concept in which each processor solves the whole prob-lem instance, using a different algorithm, random seed orother kind of diversification. And over all we see in the simple is the algo uses disatance measurement for obtaining the final goal.

In brief the algorithm described is We use a heuristic to estimate the minimum number of pushes necessary to solve a Sokoban level from a given state. This is done by assigning a goal to each box and then sum-ming up the distance of each box to its goal. In this process we use different metrics to calculate the distance between box and the box final position with the help of the Man-hattan distance, the Pythagorean distance and the Goal Pull Distance, which is the distance a box has to be pushed fromeach square to a goal if no other boxes were present and theplayer could reach every part of the level.

In order to assign a goal to each box we first compute the distance of each box to each goal. Then we use one of the following assignmentprocedures: Closest Assignment – each box is assigned tothe goal closest to it. The Hungarian Method (Kuhn 1955) –minimizes sum of box goal distances. Greedy Assignment –approximate a minimal perfect matching by choosing pairsin ascending order by their cost. This how the algoritm is described and they also uses the deadlock recognizer to find the deadlocked.

we tried to apply their algoritm as in the later phase of the paper they tried to say about the comparison of the algorithm time use by them. So we apply the BFS, DFS, A*(Manhattan and Eucledian) and other algorithm like UCS and IDS. So we can solve a puzzle with different algorithm and then compare our time interval within our algorithm portfolio.

As I am not aware of deadlock so I have not tried it.

And the parallelization they are talking about is running the solving a algorithm and the other one manages communication with other solvers in the portfolio. While the solving thread runs uninterruptedly until a solution is found, the communication thread will only run periodically. The second level of parallelization is running mutiple instances of the program on different CPUs. They com-municate using the Message Passing Interface (MPI). So what I understand from this is they are trying to compute evey algorithm in the puzzle and trying to compare ing with the time required in instance or in the different cores. As we have tried to solve our algorithm in background and compare them and tell the solution. The complete time of the solution exceed as I was not able to solve them in different cores and don't have the different CPU.so I created a portfolio where we apply the algorithm and see which algorithm runs fast.

2.3 literature review

For solving the sokoban puzzle we created a algorithm portfolio so that we can fnd the optimal algorithm for the given state Breath First Search (BFS), Depth First Search (DFS), Uniform Cost Search (UCS),Greedy Search and A* search algorithm. To find the optimal algorithm we compare the no. of steps taken for competion and the time for the analysis.

A. Breadth First Search (BFS):

   it is an algorithm for searching a tree which satisfies a given property for a node. It starts at the tree root and explores all nodes at the present depth prior and to the next level also.
   Time complexity is $O(b^{d+1})$ where b is the "branching factor" of the graph and d is the distance from first node. Here branch factor of the graph is 4(up, down,left,right).

B. Depth First Search (DFS)
   DFS is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking.Time complexity is $O(b^m)$ time but occupies only O(b*m) space, which is linear. Here m is max. search depth. DFS is fater than BFS but consume too many moves.

C. Uniform Cost Search (UCS)
   It is a variant of BFS. A uniform cost search keeps a priority queue which puts states with fewer costs in front.BFS will provide solution in less step but UCS will provide it in lowest cost. But the cost in our puzzle is same in both but we result shows that UCS is faster than BFS due to the inner structure of a LinkedList and a Priority Queue.

D. Greedy Best First Search
   It always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. For Sokoban puzzle, we adopted two possible heuristics: (1) Euclidean distances and (2) Manhattan distances. Euclidean distances heuristics calculates the sum of Euclidean distances between boxes and targets, whereas the other calculates the absolute values of the differences between boxes and
   their targets.

E. A* Algorithm
   It is a graph traversal and path search algorithm, which is often used in many fields of computer science due to its completeness, optimality, and optimal efficiency. In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a fitness number.
   F(n)=g(n)+h(n)
   F(n)-estimated cost of the cheapest solution
   G(n)- cost to reach node n from start state
   H(n)-cost to reach from node n to goal node

3 CONCLUSION

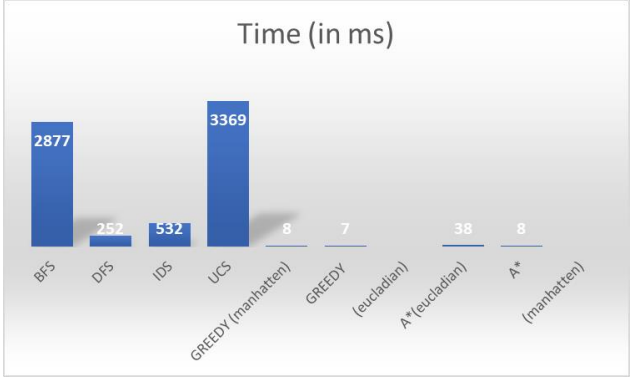
Sokoban is a typical and accessible puzzle model for testing AI algorithms. Our practices have proved that it can be well-formed as a graph search algorithm, Overall in the portfolio algorithm majorly greedy algorithm with manhatten distance win as it is computation time is very low and it worked in the most of the puzzle. Some of my puzzle were deadlocked which I got to know when I was solving the puzzle as they start giving me error
"formal_inputs\input_10_10_1_3.txt (The system cannot find the path specified)"
So to run my model fast we close the model by giving out put model failed.

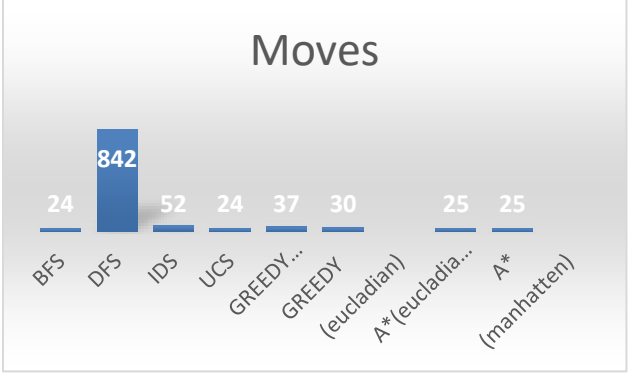
4 FUTURE AND DISCUSSION

Our knowledge was not good enough to rewrite the same algorithm as stated in the paper.
In future we can try to use deadlock recognizer to solve our failed model and we can also use different cores which was the main function of the paper that we chosen. learning is the thing we can do and solve the puzzle and help.

5 RESULT

As the time complexity of many algorithm are same but the some of the algoritm runs fast some slow which is because of the inner structured and the process they follow in iteration. The results are shown below and we can chose the perfect algorithm with comparison and find the solution.





S no.	Algorithm Used	Moves	Time (in ms)
1	BFS	24	2877
2	DFS	842	252
3	IDS	52	532
4	UCS	24	3369
5	GREEDY (manhatten)	37	8
6	GREEDY (eucladian)	30	7
7	A*(eucladian)	25	38
8	A* (manhatten)	25	8

```
------------Initial Map---
The map is as follows:
# # # # # # # # # #
#             #   #
#         # #     #
#   #       #     #
#                 #
# #               #
# # # # # # # # # #
**************** Solution Found ! *****************
ullldrulurrrrurddldrr
greedy(manhatten): 2 ms
This Map works: (10Ã—10) walls: 10 boxes: [Point(8,3):251]
```

6 REFRENCE

I. Culberson, J. 1997. Sokoban is pspace-complete.

II. AI Sokoban Solver: CS271 Project Report Zhenhan Li (44129905), Yue Zeng (36487716), Yang Jiao (59085058) University of California, Irvine

III. Junghanns, A., and Schaeffer, J. 1998. Sokoban: Evaluatingstandard single-agent search techniques in the presence ofdeadlock

7 GIT HUB REPOSITORY LINK

HTTPS://GITHUB.COM/NAVNEET1395/AI-PROJECT-