View on GitHub

AI-for-rush-hour-game

Rush Hour Game Solver Using A* with Different Heuristics

Download this project as a .zip file Download this project as a tar.gz file

0:00 / 0:57

Abstraction

Finding initial configurations(puzzles) for rush hour game and implementing A*algorithm with different heuristics to find out which one is better in term of CPU time and number of expanded nodes. Finding an optimal solution to rush hour puzzle is easy but finding best algorithm or heuristic to solve these puzzles is difficult.

Introduction

Rush hour game is sliding block puzzle game. It invented in the 1970s by Nob Yoshigahara. The board size of the game is 6x6 and contains only two types of vehicles. First vehicle type has length of two blocks and second

vehicle type is a truck that has length of three blocks. Each type can move either horizontally or vertically and none of vehicles can move both direction. The target vehicle is a horizontal car in row 2. The goal is to move the target vehicle to the exit (out of the board). Where each puzzle has its own configuration, so finding a way to move the target vehicle to the exit is not an easy because you have to figure out how to clean the way to the exit. My project focuses on implementing A* with different heuristics to solve the game and make a comparisons between these heuristics.

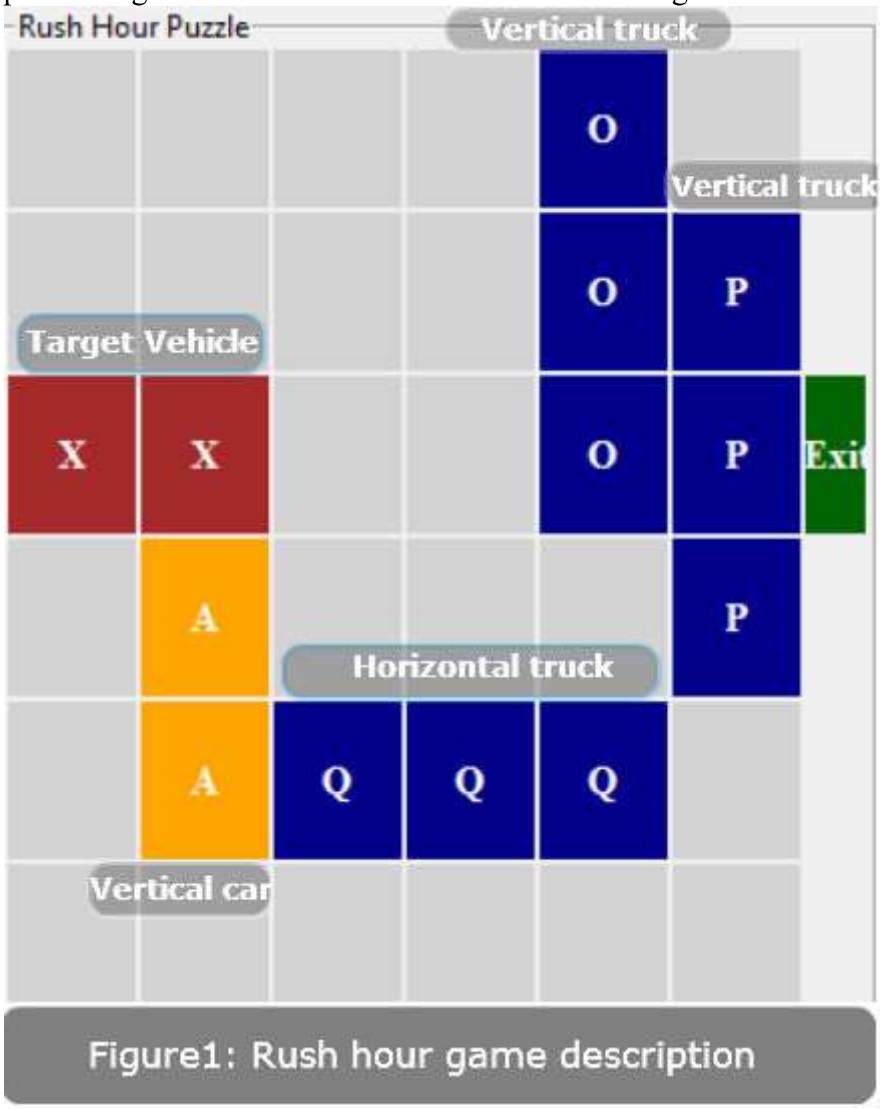

Figure1: Rush hour game description

Motivation

The motivation of this project is finding an optimal solution for puzzles like rush hour is an easy job but finding faster and cleverer search is difficult. As well learning Artificial intelligence techniques through games that we are interested in is really fun way to master things.

Method

Step1: generated random puzzles with different number of vehicles.

Step2: Implementing Hill Climbing method to find harder puzzles (e.g. puzzles take longer steps to solve).

The following the pseudo code for hill climbing:

```
generateHarderPuzzlesUsingHillClimbing(startPuzzle)
   currentPuzzle = startPuzzle;
   While True:
```

```
nextPuzzle = None
for neighbor in currentPuzzle.getNextPossibleMoves():
  if NumOfStepsToSolve(neighbor) > NumOfStepsToSolve (nextPuzzle):
     nextPuzzle = neighbor
if NumOfStepsToSolve (nextPuzzle)<=NumOfStepsToSolve(currentPuzzle):
   return currentPuzzle
currentPuzzle = nextPuzzle
```

Step4: Apply A* star algorithm with different heuristics to solve the puzzles

Heuristics:

H0: Zero heuristics. Here the A* acts like BFS.

H1: Distance from target vehicle to the exit.

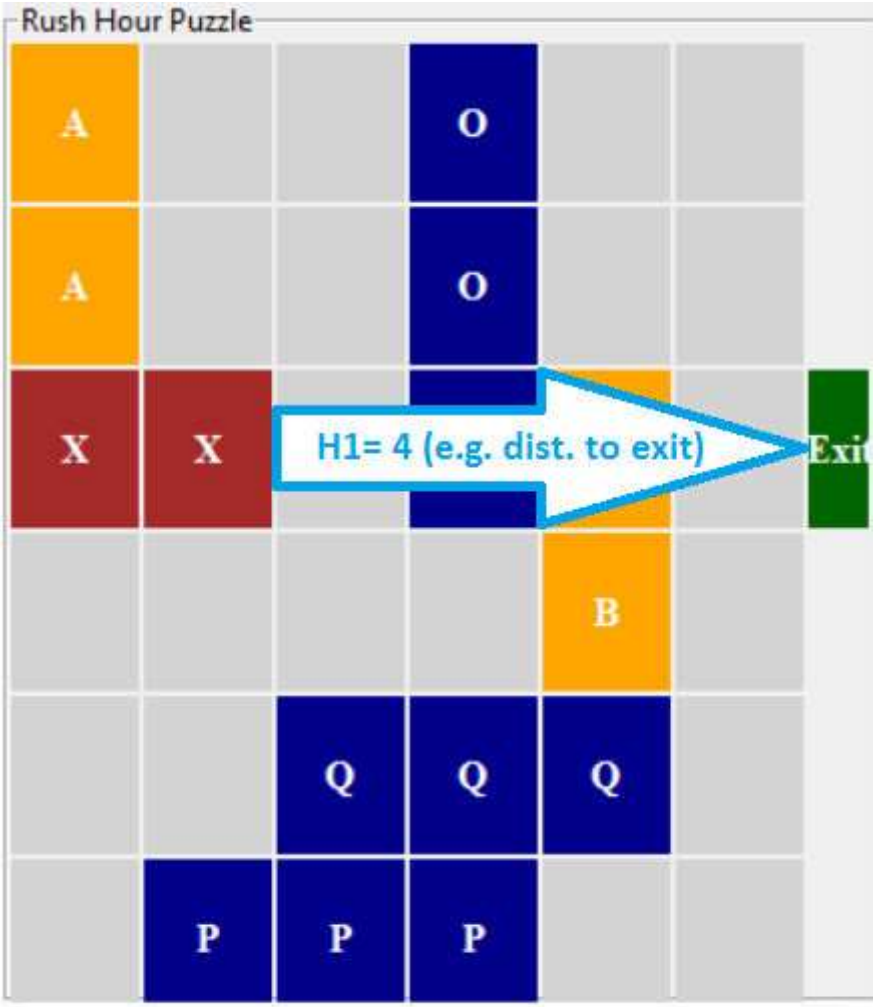

Figure2: H1 description

H2: blocking heuristic e.g. count number of vehicles that block the way to the exit.

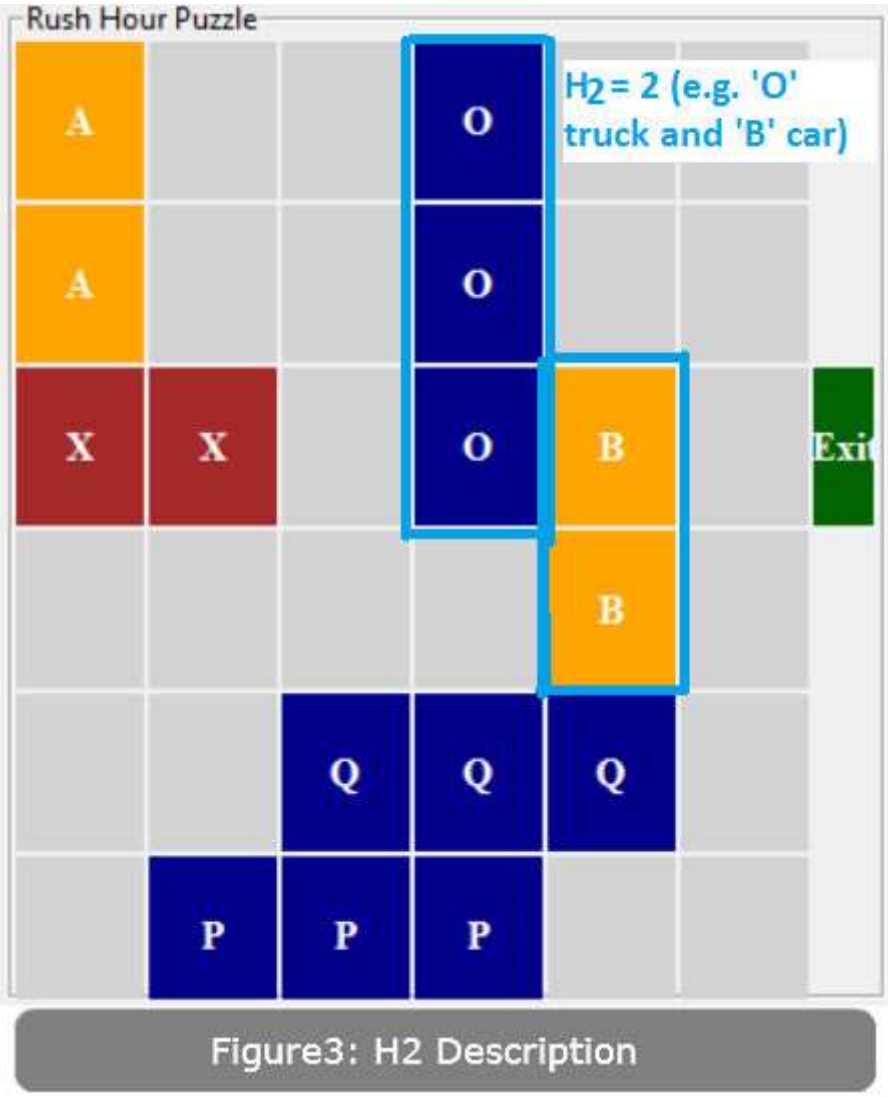

Figure3: H2 Description

H3: lower bound blocking which is H2 plus the minimum number of vehicles that block these vehicles in H2.

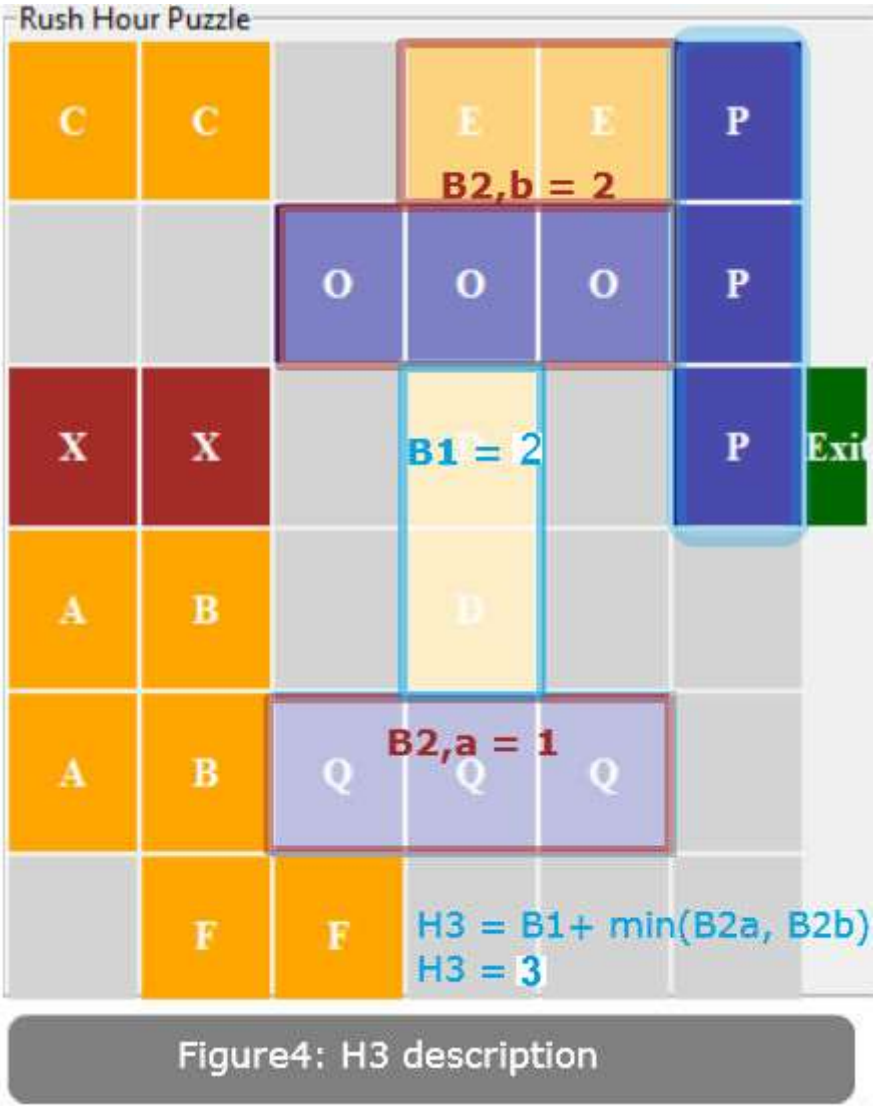

Figure4: H3 description

Admissibility and Consistency of the heuristics

H1: This heuristics compute the distance from target to the exit. In rush hour puzzle the cost to the goal is always greater than or equal to the steps (e.g. distance to the exit) that are required to move the target vehicle from its current place to the exit. Therefore, it is admissible and it never overestimates the cost to the goal. Also it is consistent because when the target vehicle moves one step toward the exit it always return a value that is reduced by one comparing with the state before moving.

H2: When there is vehicles block the way to the exit, these vehicles have to move for sure. So, it is admissible and it never overestimates the cost to the goal. As well it is consistent because it counts the vehicles that are blocking the exit and when one vehicle move out then the heuristic value is reduced by one compared to when the vehicle blocking the way.

H3: For this heuristic there is two types of blocking vehicles. First type is the vehicles that block target vehicle to the exit way and second type is these vehicles that block the first vehicle. It always returns the number of vehicles in first type plus the minimum number of vehicles in second type because these vehicles have to move as well in according to move vehicle in first type. Therefore, it is admissible. It is consistent because when one vehicle in first type or second type move out, it will return less value compared to the state before moving.

Results

Here is a figure shows the hardness (e.g. minimum number of steps required to solve a puzzle) of puzzles before(e.g. random puzzles) and after applying hill climbing:

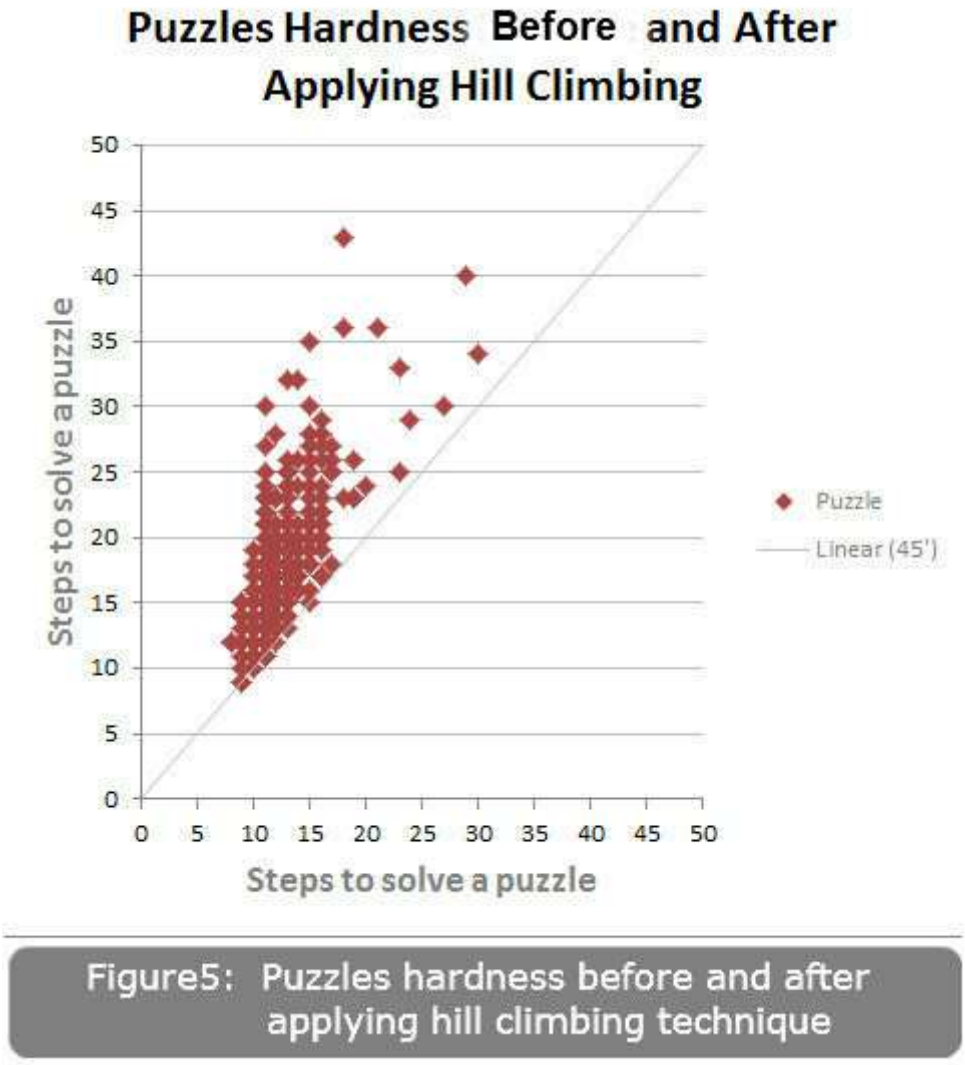

Figure5: Puzzles hardness before and after applying hill climbing technique

Here is figure shows the heuristics comparison based on number of expanded nodes:

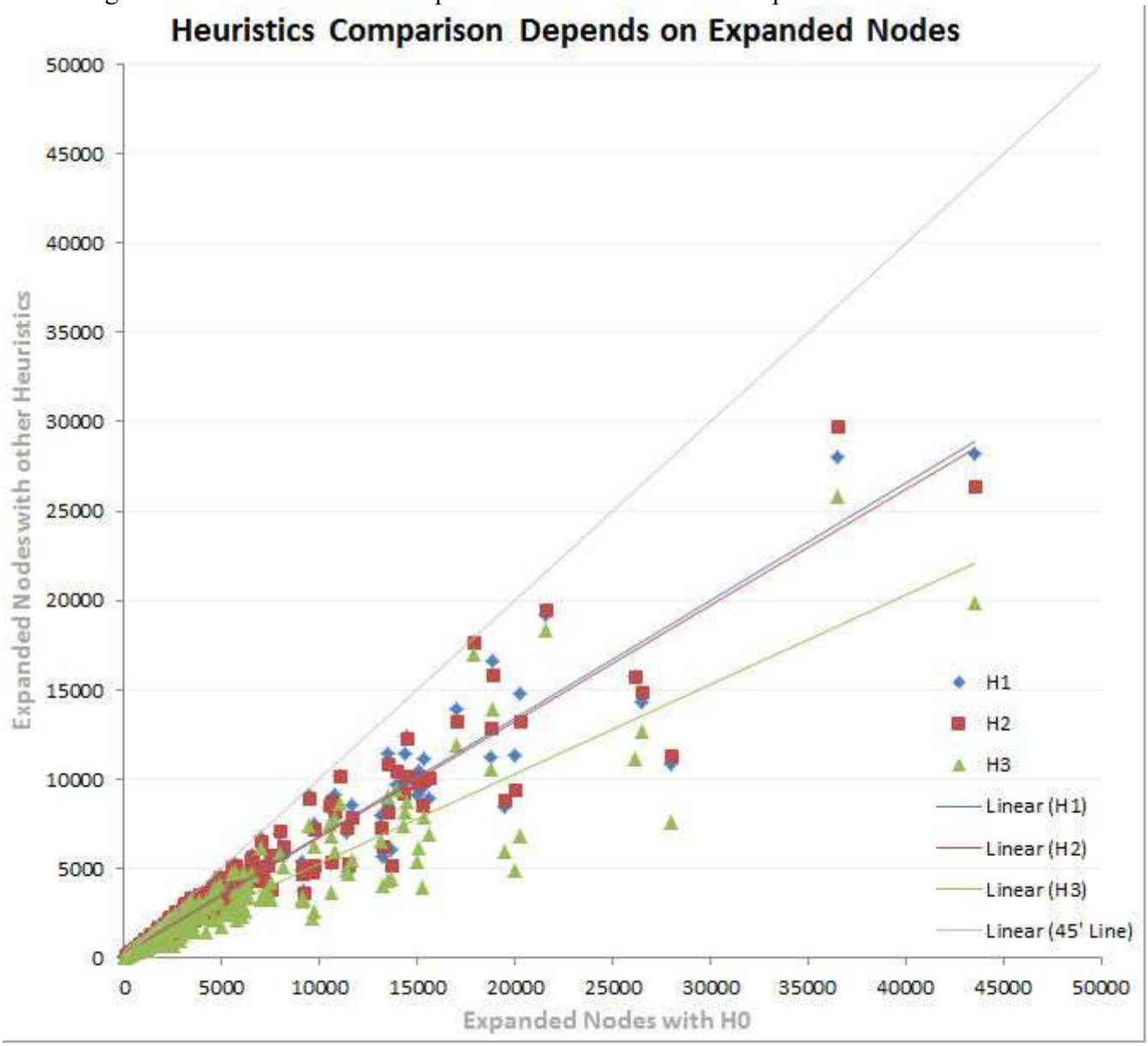

Figure6: Heuristics comparison based on number of expanded nodes

Here is figure show the heuristics comparison based on time in milliseconds to solve rush hour puzzles:

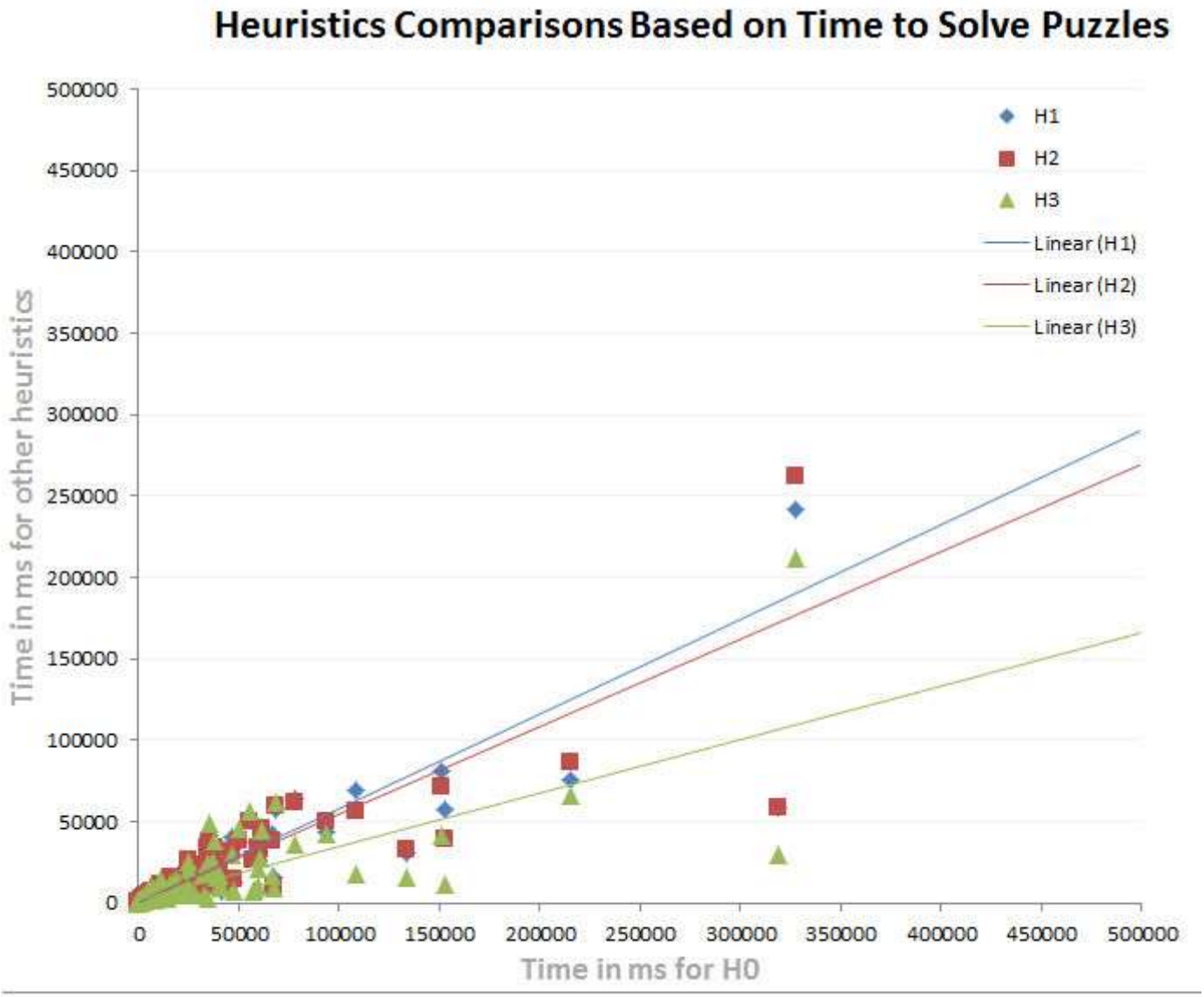

Figure7: Heuristics comparison based on time to solve puzzles

Conclusion

A* algorithm is one possible way to solve such puzzles. All heuristics ,that I used, solve puzzles in minimum number of steps. So, they are admissible and consistent. H1, H2, and H3 all work better than H0 in term of CPU time and number of expanded nodes. H2 is slightly better than H1. Finally, H3 is work very well to solve puzzles and it is better than other heuristics in terms of time and number of expanded nodes.

Future work

One possible thing to do in the future is to develop more advance heuristics and compare it with those that already have. The idea of coming heuristic is to consider all the vehicle in the whole puzzle that have to move in such way to clear the way to the exit. Other interesting thing come to my mind is to develop more scientific method that can produce harder puzzles. Finally, I really interested to know what are the parameters that make this puzzle game interested for humans.

Reference

https://en.wikipedia.org/wiki/Rush_Hour_%28board_game%29

AI-for-rush-hour-game maintained by AbbasHommadi

Published with GitHub Pages