Analysis of Search Algorithms

Analysis of
BFS, DFS, IDS, A*, IDA*
Using the Rush Hour Puzzle

Willie Jeng
109550173

Artificial Intelligence Capstone
Programming Assignment #1
National Yang Ming Chiao Tung University
4/6/2021

Contents

1 Introduction

This report analyzes and compares different search algorithms for problem solving. Given an initial state of a problem, the solution is a sequence of actions performed on the state to reach a "goal state" with certain given conditions.

The search algorithms analyzed are breath-first search (BFS), depth-first search (DFS), iterative deepening search (IDS), A* search, and iterative deepening A* search (IDA*). For A* and IDA*, the effectiveness of various heuristic functions will also be analyzed.

1.1 The problem

The toy problem to be used to test the algorithms is a simulation of the Rush Hour puzzle. Given the positions, lengths, and orientations of many cars on a 6×6 grid, cars are moved along their orientations to reach the goal state, which is any state where the car of index 0 reaches the position $(2, 4)$.

For this problem, an action is moving any car to any legal position. A legal position is any position along a car's orientation, as long as no other car is blocking the movement from the car's start position to end position. Each action has an equal cost, so the optimal solution has the least amount of actions.

2 Experiment

The experiment program was written in C++17, compiled using G++ 9.3.0, and run in Ubuntu on the Windows Subsystem for Linux. The entire code can be found in the Appendix.

2.1 Simulation of the problem

The game was simulated using an object-oriented approach with three classes:

- The `Car` class represents a single toy car on the board. The class contains the index, position, length, and orientation of a toy car.
- The `Action` class represents movement of a certain toy car to a certain position on the board. It contains the index of the toy car and the new position of said car. All `Action` objects can be assumed to be legal moves as this is tested for before constructing the object.
- The `State` class represents a state of the puzzle. It uses both the `Car` and the `Action` classes. The class contains a list of `Car` objects `cars` denoting all the cars on the board, a list of `Action` objects `actions` listing all actions made from the initial state to reach this state, and a 6×6 array `occ` noting the index of the car occupying a certain square, or -1 if it is unoccupied. `std::vector` is used for implementation of the lists. There are four methods in the `State` class:
  - `bool isGoalState()` performs a simple check on whether car #0 is in the correct position for a goal state. Time complexity is $O(1)$.

– `bool isLegalAction(Action a)` checks whether the action can be performed on the current state. This method is the main reason for maintaining `occ`. Without `occ`, checking for illegal actions would require looping through all cars, which would give this method a time complexity of $O(N)$ for a state with N cars. Maintaining `occ` only increases memory and time spent performing an action by a constant while reducing this method to $O(1)$ time.

– `vector<Action> expandState()` returns a list of actions that can be performed on the current state. This is done by checking if every single action is legal. Some pruning is done by not including actions that do not move any car, and not including actions that involve the last car moved in the state (if two actions are performed consecutively on the same car, they could have been performed as one action). Since the cars can only move on a grid of fixed size, the time complexity of this method is $O(1)$.

– `State performAction(Action a)` returns the state gotten after performing the action on the current state. This method is used for the expansion step of search algorithms, and also runs in $O(1)$ time.

2.2 Implementation of searches

All the search algorithms are written in similar fashion, with the data structure used to store the frontier states being the main difference.

The initial state is first added to the frontier. The first state on the frontier is checked for a goal state, and is returned as the solution if it is one. Otherwise, the state is expanded upon, the expanded states are added to the frontier, and the current state is removed from the frontier.

Graph search is used in the searches, however it can be easily changed to tree search by removing the lines of code related to the explored set (see Appendix). The explored set is implemented as an `std::map`, which is an implementation of a red-black tree. The key is a 6×6 array representing the positions of the cars on the board, with the same format as `State.occ`. The value is an `int` representing the least amount of actions performed to reach the key. If a key is reached after an amount of actions that is greater than the value, the state removed from the frontier without being expanded upon. See Observations for analysis on the impact of the explored set on time and memory.

2.2.1 Breadth-first search

An `std::queue` is used to store frontier states, which are expanded in order of being added until a goal state is found.

2.2.2 Depth-first search

An `std::stack` is used to store frontier states instead of recursion, as `std::stack` uses the heap instead of the thread stack and has a higher limit.

2.2.3 Iterative deepening search

DFS is repeatedly run on the initial state with an increasing maximum depth, starting from 1, until a solution can be found. To perform DFS with a maximum depth, a state is not expanded upon if its depth exceeds the maximum depth.

2.2.4 A* search

An `std::priority_queue`, which is an implementation of a max heap, is used to store frontier states. The heap is ordered by the costs of the states, which for this problem is defined as the heuristic added to the amount of actions. The states are stored into the priority queue using `HContainer` objects, which also stores the cost in order to avoid having to calculate it multiple times.

The heuristic function used is the blocking heuristic. The blocking heuristic simply calculates the number of cars blocking the path of car #0 to the exit.

2.2.5 Iterative deepening A* search

Similar to IDS, but using A*. A* is repeatedly run on the initial state with an increasing maximum cost, starting from 0.

2.3 Testing

The metrics used are time, nodes expanded, and the greatest amount of nodes in the frontier at any time. Time is measured in microseconds using `std::chrono`, starting immediately before and ending immediately after the algorithm runs. The other two metrics are updated after every expansion.

To run the tests, a Bash script is used to cycle through all the test cases and algorithms. If a certain case runs for more than 20 seconds, it is timed out. The script outputs to stdout, but can be piped to a file using the command line.

3 Results & Observations

See Appendix for the complete experiment results.

The maximum depth of the state space m is significantly large. m is infinite if performing tree search, and still large if performing graph search. The maximum branching factor of the search tree b is manageable, as each car can only have at most five possible actions at any state. The depth of the optimal solution d is also manageable, which makes sense for a game meant to be able to be solved by hand.

3.1 Comparison of the algorithms

The experiment results mostly reflected the advantages and drawbacks of each algorithm mentioned in the lectures. However, it became apparent that some algorithms were better in every way than others for this specific problem.

BFS is a relatively effective algorithm for solving the Rush Hour puzzle. The space and time complexities for BFS are both $O(b^d)$, which is manageable. The cost of any action is equal, which gives this problem unit step costs. Because of the unit step costs, BFS gives an optimal solution.

DFS is ineffective for this problem compared to the other algorithms. Because of the large m, the $O(b^m)$ time complexity is highly ineffective. This can be seen in the results, as the time taken for DFS is significantly longer than the others, many cases even timing out before it can be finished. For graph search, DFS is complete even if it takes a long time, but if using tree search, DFS can easily fall into an infinite loop. The solution from DFS may also not be optimal, and for many cases, the solution has many times the amount of actions of an optimal solution. The advantage of DFS should be its space complexity of $O(mb)$. However, because of the large m, the number of nodes in the memory is often greater than that of BFS.

IDS is slightly more favorable than DFS, but is also inefficient for this problem. IDS has a time complexity of $O(b^d)$, which is more slightly more efficient than DFS, which has a time complexity of $O(b^m)$. However, it can take more time than DFS at times, since it performs the search multiple times with different maximum depths, and for some cases, IDS times out while DFS does not. Despite this, IDS gives an optimal solution because of the unit step costs, and is complete since d is finite.

A* has the same efficiency as BFS for tree and graph traversal, but can reach a goal state faster with its heuristic function. However, inserting a state into the heap takes $O(\lg n)$ time, and it takes almost twice as long to run than BFS for most cases. It requires slightly fewer nodes to be expanded than BFS, showing the impact of the heuristic function.

Although having the same time complexity as A*, IDA* takes more time because it runs A* multiple times at increasing maximum costs. The advantage for IDA* is the space complexity. This can be seen in the experiment results, but the maximum amount of nodes in the memory is not reduced at all for some test cases. This was because IDA* saves memory by not adding new states over a certain cost into the heap, but for some cases no states were omitted because of its cost before a solution was found.

3.2 Analysis of the heuristic function

The number of actions needed to bring a state to a goal state is always greater or equal to the number of cars blocking car #0 from the exit. This is because to it takes at least one action to move each car blocking car #0 out of the way. Therefore, the heuristic is always less than or equal to the true cost, making it admissible. However this can only guarantee optimality for tree search.

In order for a heuristic to be consistent and be optimal for graph search, it needs to satisfy the triangle equality: $h(n) \leq c(n, a, n') + h(n')$, n being the current state and n' being the state after performing an action a on n. The costs for all actions for this problem are the same, which would be assumed to be 1. The triangle equality can now be written as $h(n) - h(n') \leq 1$, meaning each action can only improve the heuristic by at most one. This applies to the blocking heuristic, as each action can move at most one car out of the exiting path.

3.3 Tree search vs. graph search

The tests were done using graph search, with the main reason being that tree search would be significantly slower and testing it for all test cases would be unrealistic. DFS is not complete for tree search, and tree search with DFS always resulted in the program being stuck at an endless loop of the same cars moving back and forth.

The minimum depth of a state for a certain board position had to be stored in the explored set in order for DFS and IDS to run correctly. If the minimum depth was not stored, states of lower depth would not be able to expand if states of high depth had already reached the same board position, which would make IDS not optimal. Therefore, an `std::map` was required for DFS and IDS. For BFS, since all states of a lower depth would be expanded before a higher depth, the value of the positions in the explored set would always be less than or equal to the current depth if it existed, and so an `std::set` could have been used, which would have been slightly faster, although having the same time complexity for lookup. However, a `std::map` was used for all algorithms for consistency in time calculations.

4 Remaining Questions

Randomness could have been an interesting factor in searching. For example, for DFS, would adding new states to the frontier in a random order decrease the chance of running into an infinite loop before finding a solution?

The blocking heuristic reduced the amount of nodes expanded, but the additional time of the heap made A* take more time than BFS in most cases. If a heuristic function could be devised that can better estimate the cost of a state, A* and IDA* could be made more effective.

5 Conclusion

By measuring the time, the amount of nodes expanded, and the maximum amount of nodes in the frontier for many test cases using different algorithms, it could be seen that BFS performed the best out of these algorithms in both time and space. However, A* and IDA* could potentially perform better with a different heuristic function. DFS and IDS were not effective algorithms for this problem because of the sizes of the problem's branching factor and depth.

A Program Structure

```
src/
    car.hpp
    action.hpp
    state.hpp
    heuristic.hpp
    searches.hpp
    car.cpp
    action.cpp
    state.cpp
    heuristic.cpp
    bfs.cpp
    dfs.cpp
    ids.cpp
    aStar.cpp
    idaStar.cpp
    main.cpp
tests/
    L01.txt
    L02.txt
    ...
    L40.txt
test.sh
```

B Program Code

B.1 Compiling and running the program

For Linux:

- To compile the code, run `g++ src/*.cpp` in the root directory.
- To run the program, run `./a.out` in the root directory. The program takes two arguments.
  - The first argument is a number corresponding to the algorithm.
    1. BFS
    2. DFS
    3. IDS
    4. A*
    5. IDA*
  - The second argument is the filename of the test case inside `tests/`.
- To test for all algorithms for all cases in `tests/`, run `chmod +x ./test.sh` then `./test.sh` in the root directory. The timeout time can be changed by changing `TOTIME`.

B.2 car.hpp

```
#ifndef CAR_HPP
#define CAR_HPP

class Car {
public:

    int ind;
    int row;
    int col;
    int len;
    int ori;

    Car(Car const& a);
    Car(int a, int b, int c, int d, int e);

};

#endif
```

B.3 action.hpp

```
#ifndef ACTION_HPP
#define ACTION_HPP

#include <ostream>
using namespace std;

class Action {
public:
    int ind;
    int row;
    int col;
    Action(int a, int b, int c);
};

ostream& operator<<(ostream& os, Action const& ac);

#endif
```

B.4 state.hpp

```
#ifndef STATE_HPP
#define STATE_HPP

#include <vector>
#include <array>
using namespace std;

#include "car.hpp"
#include "action.hpp"

class State {
```

```
public:

    vector<Car> cars;
    vector<Action> actions;
    array<array<int,6>,6> occ; // occupied squares, -1 if empty,
    otherwise the car index
    bool failed = false;

    State(vector<Car> const& v1, vector<Action> const& v2);
    State(State const& S);
    State(bool f);

    bool isGoalState();
    bool isLegalAction(Action const& a);
    vector<Action> expandState();
    State performAction(Action const& a);

    int carsBlockingExit();

};

ostream& operator<<(ostream& os, State const& st);

const State FAILED_STATE(true);

#endif
```

B.5 heuristic.hpp

```
#ifndef HEURISTIC_HPP
#define HEURISTIC_HPP

#include "state.hpp"

class HContainer {
public:
    State s;
    int h;
    HContainer(State const& a);
};

class HComp {
public:
    bool operator()(HContainer const& lhs, HContainer const& rhs);
};

int getCost(State const& s);

int blockingHeuristic(State const& s);

#endif
```

B.6 searches.hpp

```
#ifndef SEARCHES_HPP
#define SEARCHES_HPP

#include "state.hpp"

State bfs(State initState, int &expandedNodes, int &maxNodes);

State dfs(State initState, int &expandedNodes, int &maxNodes);
State dfs(State initState, int maxDepth, int &expandedNodes, int &
    maxNodes);

State ids(State initState, int &expandedNodes, int &maxNodes);

State aStar(State initState, int &expandedNodes, int &maxNodes);
State aStar(State initState, int maxCost, int &expandedNodes, int &
    maxNodes);

State idaStar(State initState, int &expandedNodes, int &maxNodes);

#endif
```

B.7 car.cpp

```
#include "car.hpp"

Car::Car(Car const& a)
    : ind(a.ind), row(a.row), col(a.col), len(a.len), ori(a.ori) { }

Car::Car(int a, int b, int c, int d, int e)
    : ind(a), row(b), col(c), len(d), ori(e) { }
```

B.8 action.cpp

```
#include "action.hpp"

#include <ostream>
using namespace std;

Action::Action(int a, int b, int c)
    : ind(a), row(b), col(c) { }

ostream& operator<<(ostream& os, Action const& ac){
    os << "Car " << ac.ind << " to (" << ac.row << "," << ac.col << ")"
   ;
    return os;
}
```

B.9 state.cpp

```
#include "state.hpp"
```

```cpp
#include <vector>
#include <array>
#include <ostream>
using namespace std;

#include "car.hpp"
#include "action.hpp"


State::State(vector<Car> const& v1, vector<Action> const& v2)
    : cars(v1), actions(v2) {
    for(int i=0; i<6; ++i)
        for(int j=0; j<6; ++j)
            occ[i][j] = -1;
    for(auto i : cars){
        for(int j=0; j<i.len; ++j){
            if(i.ori == 1) occ[i.row][i.col+j] = i.ind;
            else occ[i.row+j][i.col] = i.ind;
        }
    }
}

State::State(State const& S)
    : cars(S.cars), actions(S.actions), failed(S.failed) {
    for(int i=0; i<6; ++i)
        for(int j=0; j<6; ++j)
            occ[i][j] = S.occ[i][j];
}

State::State(bool f)
    : failed(f) { }

bool State::isGoalState() {
    return (cars[0].row == 2 && cars[0].col == 4);
}

bool State::isLegalAction(Action const& a) {

    const Car *cc = &cars[a.ind]; // current car
    int p1,p2;
    if(cc->ori == 1)
        p1 = cc->col, p2 = a.col;
    else
        p1 = cc->row, p2 = a.row;
    if(p1 > p2) swap(p1,p2);
    p2 += cc->len - 1;


    for(int i=p1; i<=p2; ++i){
        if(cc->ori == 1 && (occ[a.row][i]!=-1
            && occ[a.row][i]!=a.ind))
            return false;
        if(cc->ori == 2 && (occ[i][a.col]!=-1
            && occ[i][a.col]!=a.ind))
            return false;
```

```
    }

    return true;

}

vector<Action> State::expandState() {

    vector<Action> ret;

    for(auto i : cars) {
        if(!actions.empty() && actions.back().ind == i.ind) continue;
        for(int j=0; j<7-i.len; ++j){
            int tr, tc;
            if(i.ori == 1){
                if(j == i.col) continue;
                tr = i.row, tc = j;
            } else {
                if(j == i.row) continue;
                tr = j, tc = i.col;
            }
            Action tmpAct(i.ind,tr,tc);
            if(isLegalAction(tmpAct))
                ret.push_back(tmpAct);
        }
    }

    return ret;

}

State State::performAction(Action const& a) {

    State ret(*this);

    Car* cc = &ret.cars[a.ind]; // current car
    for(int i=0; i<cc->len; ++i){
        if(cc->ori == 1) ret.occ[cc->row][cc->col+i] = -1;
        else ret.occ[cc->row+i][cc->col] = -1;
    }
    cc->row = a.row;
    cc->col = a.col;
    for(int i=0; i<cc->len; ++i){
        if(cc->ori == 1) ret.occ[cc->row][cc->col+i] = a.ind;
        else ret.occ[cc->row+i][cc->col] = a.ind;
    }
    ret.actions.push_back(a);

    return ret;
}

int State::carsBlockingExit() {

    int ret = 0;
    for(int i=cars[0].col+cars[0].len; i<6; ++i)
```

```
        ret += occ[2][i];
    return ret;

}

ostream& operator<<(ostream& os, State const& st){

    for(int i=0; i<6; ++i, os << "\n")
        for(int j=0; j<6; ++j){
            if(st.occ[i][j] == -1) os << " . ";
            else os << (st.occ[i][j]<10&&st.occ[i][j]>=0?" ":"") << st.
   occ[i][j] << " ";
        }
    for(auto i:st.actions)
        os << i << "\n";

    return os;

}
```

B.10 heuristic.cpp

```
#include "heuristic.hpp"

HContainer::HContainer(State const& a)
    : s(a) {
    h = getCost(s);
}

bool HComp::operator()(HContainer const& lhs, HContainer const& rhs) {
    return lhs.h > rhs.h;
}

int getCost(State const& s){
    return s.actions.size() + blockingHeuristic(s);
}

int blockingHeuristic(State const& s){
    int ret = 0;
    for(int i=s.cars[0].col+s.cars[0].len; i<6; ++i)
        ret += (s.occ[2][i]!=-1);
    return ret;
}
```

B.11 bfs.cpp

```
#include "searches.hpp"

#include <iostream>
#include <vector>
#include <queue>
#include <map>
#include <array>
```

```cpp
using namespace std;

#include "state.hpp"

State bfs(State initState, int &expandedNodes, int &maxNodes){

    queue<State> qu;
    qu.push(initState);
    map<array<array<int,6>,6>,int> ex; // explored set

    while(!qu.empty()){

        maxNodes = max(maxNodes, (int)qu.size());

        State curState = qu.front();
        qu.pop();

        if(curState.isGoalState())
            return curState;

        // check if in explored set
        auto *i = &ex[curState.occ];
        if(*i && *i <= (int)curState.actions.size() + 1) continue;
        *i = (int)curState.actions.size() + 1;

        // expand frontier
        vector<Action> acts = curState.expandState();
        for(auto i : acts)
            qu.push(curState.performAction(i));
        ++expandedNodes;

    }

    return FAILED_STATE;

}
```

B.12 dfs.cpp

```cpp
#include "searches.hpp"

#include <iostream>
#include <vector>
#include <stack>
#include <map>
#include <array>
using namespace std;

#include "state.hpp"

State dfs(State initState, int &expandedNodes, int &maxNodes) {
    return dfs(initState, -1, expandedNodes, maxNodes);
}
```

```cpp
State dfs(State initState, int maxDepth, int &expandedNodes, int &
    maxNodes){ // infinite depth if maxDepth = -1

    stack<State> st;
    st.push(initState);
    map<array<array<int,6>,6>,int> ex; // explored set, value is the
    depth

    while(!st.empty()){

        maxNodes = max(maxNodes, (int)st.size());

        State curState = st.top();
        st.pop();

        if(maxDepth != -1 && (int)curState.actions.size() + 1 >
    maxDepth) continue;

        if(curState.isGoalState())
            return curState;

        // check if in explored set
        auto *i = &ex[curState.occ];
        if(*i && *i <= (int)curState.actions.size() + 1) continue;
        *i = (int)curState.actions.size() + 1;

        // expand frontier
        vector<Action> acts = curState.expandState();
        for(auto i : acts)
            st.push(curState.performAction(i));
        ++expandedNodes;

    }

    return FAILED_STATE;

}
```

B.13 ids.cpp

```cpp
#include "searches.hpp"

#include <iostream>
using namespace std;

#include "state.hpp"

State ids(State initState, int &expandedNodes, int &maxNodes){

    int depth = 1;
    State retState = dfs(initState, depth, expandedNodes, maxNodes);

    while(retState.failed){
        ++depth;
```

```
        retState = dfs(initState, depth, expandedNodes, maxNodes);
    }

    return retState;

}
```

B.14 aStar.cpp

```
#include "searches.hpp"

#include <iostream>
#include <algorithm>
#include <queue>
#include <map>
using namespace std;

#include "state.hpp"
#include "heuristic.hpp"

State aStar(State initState, int &expandedNodes, int &maxNodes){
    return aStar(initState, -1, expandedNodes, maxNodes);
}

State aStar(State initState, int maxCost, int &expandedNodes, int &
   maxNodes){  // infinite cost if -1

    priority_queue<HContainer, vector<HContainer>, HComp> pq;
    pq.push(HContainer(initState));
    map<array<array<int,6>,6>,int> ex; // explored set, value is the
   depth

    while(!pq.empty()){

        maxNodes = max(maxNodes, (int)pq.size());

        HContainer curCont = pq.top();
        pq.pop();

        if(maxCost != -1 && curCont.h > maxCost) continue;

        if(curCont.s.isGoalState())
            return curCont.s;

        // check if in explored set
        auto *i = &ex[curCont.s.occ];
        if(*i && *i <= (int)curCont.s.actions.size() + 1) continue;
        *i = (int)curCont.s.actions.size() + 1;

        // expand frontier
        vector<Action> acts = curCont.s.expandState();
        for(auto i : acts){
            HContainer tmpCont(curCont.s.performAction(i));
            if(maxCost == -1 || tmpCont.h <= maxCost)
```

```
                pq.push(tmpCont);
        }
        ++expandedNodes;

    }

    return FAILED_STATE;

}
```

B.15 idaStar.cpp

```
#include "searches.hpp"

#include <iostream>
using namespace std;

#include "state.hpp"

State idaStar(State initState, int &expandedNodes, int &maxNodes){

    int cost = 0;
    State retState = aStar(initState, cost, expandedNodes, maxNodes);

    while(retState.failed) {
        ++cost;
        retState = aStar(initState, cost, expandedNodes, maxNodes);
    }

    return retState;

}
```

B.16 main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <chrono>
using namespace std;
using namespace std::chrono;

#include "searches.hpp"
#include "state.hpp"
#include "action.hpp"
#include "car.hpp"

string algName(int algType);

void solve(int algType, string fileName);

int main(int argc, char **argv) {
```

```
    if(argc == 1) solve(1, "L01.txt");
    else solve(stoi(argv[1]), argv[2]);
}

string algName(int algType) {
    if(algType == 1) return "BFS";
    if(algType == 2) return "DFS";
    if(algType == 3) return "IDS";
    if(algType == 4) return "A*";
    if(algType == 5) return "IDA*";
    return "";
}

void solve(int algType, string fileName) {

    cout << fileName << ", " << algName(algType) << endl;

    vector<Car> initCars;

    int a, b, c, d, e;
    ifstream fin("tests/" + fileName);
    while(fin >> a >> b >> c >> d >> e)
        initCars.push_back(Car(a,b,c,d,e));
    fin.close();

    State initState(initCars, vector<Action>());

    int expandedNodes = 0;
    int maxNodes = 0;
    State ansState = FAILED_STATE;

    auto startTime = high_resolution_clock::now();

    if(algType == 1){
        ansState = bfs(initState, expandedNodes, maxNodes);
    } else if (algType == 2) {
        ansState = dfs(initState, expandedNodes, maxNodes);
    } else if (algType == 3) {
        ansState = ids(initState, expandedNodes, maxNodes);
    } else if (algType == 4) {
        ansState = aStar(initState, expandedNodes, maxNodes);
    } else if (algType == 5) {
        ansState = idaStar(initState, expandedNodes, maxNodes);
    }

    auto stopTime = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stopTime - startTime);

    if(ansState.failed) cout << "No solution found\n";
    else printf("%12d  actions in solution\n", (int)ansState.actions.
   size());
    printf("%12d  microseconds elapsed\n", (int)duration.count());
    printf("%12d  nodes expanded\n", expandedNodes);
    printf("%12d  nodes in memory\n", maxNodes);
```

```
}
```

B.17 test.sh

```bash
#!/bin/bash

TOTIME=20

for f in tests/L*.txt ; do
    for i in {1..5} ; do
        if ! timeout $TOTIME ./a.out $i $(basename $f) ; then
            echo Timed out after $TOTIME seconds
        fi
        echo
    done
done
```

C Complete Results

```
L01.txt, BFS
           8  actions in solution
      104546  microseconds elapsed
        1057  nodes expanded
        2477  nodes in memory

L01.txt, DFS
         995  actions in solution
      397060  microseconds elapsed
        1203  nodes expanded
        7197  nodes in memory

L01.txt, IDS
           8  actions in solution
      908179  microseconds elapsed
       11920  nodes expanded
          68  nodes in memory

L01.txt, A*
           8  actions in solution
      166871  microseconds elapsed
         952  nodes expanded
        2924  nodes in memory

L01.txt, IDA*
           8  actions in solution
      342034  microseconds elapsed
        2444  nodes expanded
        2837  nodes in memory

L02.txt, BFS
           8  actions in solution
      197713  microseconds elapsed
```

```
      2525  nodes expanded
      6900  nodes in memory

L02.txt, DFS
      1696  actions in solution
    797288  microseconds elapsed
      1705  nodes expanded
     12767  nodes in memory

L02.txt, IDS
         8  actions in solution
   1087492  microseconds elapsed
     15534  nodes expanded
        66  nodes in memory

L02.txt, A*
         8  actions in solution
    156282  microseconds elapsed
      1108  nodes expanded
      4840  nodes in memory

L02.txt, IDA*
         8  actions in solution
    175308  microseconds elapsed
      1790  nodes expanded
      2737  nodes in memory

L03.txt, BFS
        14  actions in solution
     56103  microseconds elapsed
       774  nodes expanded
      1548  nodes in memory

L03.txt, DFS
       647  actions in solution
   1383368  microseconds elapsed
      7902  nodes expanded
      3521  nodes in memory

L03.txt, IDS
        14  actions in solution
   1515193  microseconds elapsed
     21846  nodes expanded
        87  nodes in memory

L03.txt, A*
        14  actions in solution
     92726  microseconds elapsed
       627  nodes expanded
      1458  nodes in memory

L03.txt, IDA*
        14  actions in solution
    431998  microseconds elapsed
      3603  nodes expanded
```

```
         877  nodes in memory

L04.txt, BFS
           9  actions in solution
       19281  microseconds elapsed
         340  nodes expanded
         840  nodes in memory

L04.txt, DFS
         130  actions in solution
        8581  microseconds elapsed
         154  nodes expanded
         638  nodes in memory

L04.txt, IDS
           9  actions in solution
      109800  microseconds elapsed
        2097  nodes expanded
          33  nodes in memory

L04.txt, A*
           9  actions in solution
       23531  microseconds elapsed
         234  nodes expanded
         801  nodes in memory

L04.txt, IDA*
           9  actions in solution
       41527  microseconds elapsed
         543  nodes expanded
         460  nodes in memory

L10.txt, BFS
          17  actions in solution
      149403  microseconds elapsed
        1977  nodes expanded
        1870  nodes in memory

L10.txt, DFS
        1320  actions in solution
     5024195  microseconds elapsed
       26928  nodes expanded
        7725  nodes in memory

L10.txt, IDS
          17  actions in solution
     4052032  microseconds elapsed
       57051  nodes expanded
          87  nodes in memory

L10.txt, A*
          17  actions in solution
      308105  microseconds elapsed
        1662  nodes expanded
        2280  nodes in memory
```

```
L10.txt, IDA*
          17  actions in solution
     1669074  microseconds elapsed
       10803  nodes expanded
        2280  nodes in memory

L11.txt, BFS
          25  actions in solution
       52944  microseconds elapsed
         829  nodes expanded
         572  nodes in memory

L11.txt, DFS
         466  actions in solution
     2613890  microseconds elapsed
       18259  nodes expanded
        1923  nodes in memory

L11.txt, IDS
          25  actions in solution
     5176309  microseconds elapsed
       75058  nodes expanded
          68  nodes in memory

L11.txt, A*
          25  actions in solution
      102318  microseconds elapsed
         756  nodes expanded
         668  nodes in memory

L11.txt, IDA*
          25  actions in solution
     1007575  microseconds elapsed
        8420  nodes expanded
         668  nodes in memory

L20.txt, BFS
          10  actions in solution
      117642  microseconds elapsed
        1557  nodes expanded
        4261  nodes in memory

L20.txt, DFS
Timed out after 20 seconds

L20.txt, IDS
          10  actions in solution
      517978  microseconds elapsed
       10019  nodes expanded
          49  nodes in memory

L20.txt, A*
          10  actions in solution
       73042  microseconds elapsed
```

```
        664  nodes expanded
       2364  nodes in memory

L20.txt, IDA*
         10  actions in solution
     131643  microseconds elapsed
       1666  nodes expanded
       1890  nodes in memory

L21.txt, BFS
         21  actions in solution
      11758  microseconds elapsed
        257  nodes expanded
        144  nodes in memory

L21.txt, DFS
        163  actions in solution
      36307  microseconds elapsed
        485  nodes expanded
        514  nodes in memory

L21.txt, IDS
         21  actions in solution
     697712  microseconds elapsed
      17073  nodes expanded
         77  nodes in memory

L21.txt, A*
         21  actions in solution
      22177  microseconds elapsed
        248  nodes expanded
        144  nodes in memory

L21.txt, IDA*
         21  actions in solution
     195526  microseconds elapsed
       2577  nodes expanded
        144  nodes in memory

L22.txt, BFS
         26  actions in solution
     308361  microseconds elapsed
       3459  nodes expanded
       4225  nodes in memory

L22.txt, DFS
       3856  actions in solution
    9304190  microseconds elapsed
      12493  nodes expanded
      25416  nodes in memory

L22.txt, IDS
         26  actions in solution
    7527686  microseconds elapsed
     102774  nodes expanded
```

```
        107  nodes in memory

L22.txt, A*
         26  actions in solution
     549960  microseconds elapsed
       3078  nodes expanded
       4549  nodes in memory

L22.txt, IDA*
         26  actions in solution
    2096610  microseconds elapsed
      15741  nodes expanded
       3919  nodes in memory

L23.txt, BFS
         29  actions in solution
     183812  microseconds elapsed
       2379  nodes expanded
       2645  nodes in memory

L23.txt, DFS
       2898  actions in solution
    5243704  microseconds elapsed
       6719  nodes expanded
      14511  nodes in memory

L23.txt, IDS
         29  actions in solution
    4426338  microseconds elapsed
      65771  nodes expanded
        102  nodes in memory

L23.txt, A*
         29  actions in solution
     274461  microseconds elapsed
       1647  nodes expanded
       1836  nodes in memory

L23.txt, IDA*
         29  actions in solution
    1209520  microseconds elapsed
       9251  nodes expanded
       1604  nodes in memory

L24.txt, BFS
         25  actions in solution
     504968  microseconds elapsed
       4341  nodes expanded
       6151  nodes in memory

L24.txt, DFS
Timed out after 20 seconds

L24.txt, IDS
Timed out after 20 seconds
```

```
L24.txt, A*
          25  actions in solution
      962409  microseconds elapsed
        4211  nodes expanded
        7320  nodes in memory

L24.txt, IDA*
          25  actions in solution
    13984248  microseconds elapsed
       62349  nodes expanded
        7320  nodes in memory

L25.txt, BFS
          27  actions in solution
      909536  microseconds elapsed
        8474  nodes expanded
        7411  nodes in memory

L25.txt, DFS
        5298  actions in solution
     7911914  microseconds elapsed
        5479  nodes expanded
       33724  nodes in memory

L25.txt, IDS
Timed out after 20 seconds

L25.txt, A*
          27  actions in solution
     2038566  microseconds elapsed
        7333  nodes expanded
        9972  nodes in memory

L25.txt, IDA*
          27  actions in solution
    12499027  microseconds elapsed
       54530  nodes expanded
        9972  nodes in memory

L26.txt, BFS
          28  actions in solution
      623361  microseconds elapsed
        4699  nodes expanded
        3557  nodes in memory

L26.txt, DFS
        2044  actions in solution
     4937732  microseconds elapsed
       12380  nodes expanded
       11597  nodes in memory

L26.txt, IDS
Timed out after 20 seconds
```

```
L26.txt, A*
          28  actions in solution
      922575  microseconds elapsed
        4071  nodes expanded
        4012  nodes in memory

L26.txt, IDA*
          28  actions in solution
     7079796  microseconds elapsed
       41392  nodes expanded
        4012  nodes in memory

L27.txt, BFS
          28  actions in solution
      183905  microseconds elapsed
        2660  nodes expanded
        1506  nodes in memory

L27.txt, DFS
        1287  actions in solution
      363480  microseconds elapsed
        1372  nodes expanded
        6033  nodes in memory

L27.txt, IDS
          28  actions in solution
     8818802  microseconds elapsed
      139577  nodes expanded
         102  nodes in memory

L27.txt, A*
          28  actions in solution
      347690  microseconds elapsed
        2428  nodes expanded
        1598  nodes in memory

L27.txt, IDA*
          28  actions in solution
     2286392  microseconds elapsed
       17859  nodes expanded
        1598  nodes in memory

L28.txt, BFS
          30  actions in solution
      161958  microseconds elapsed
        1923  nodes expanded
        1726  nodes in memory

L28.txt, DFS
        1628  actions in solution
    16838736  microseconds elapsed
       25662  nodes expanded
        9676  nodes in memory

L28.txt, IDS
```

```
          30  actions in solution
     3978030  microseconds elapsed
       69116  nodes expanded
         109  nodes in memory

L28.txt, A*
          30  actions in solution
      240870  microseconds elapsed
        1538  nodes expanded
        2324  nodes in memory

L28.txt, IDA*
          30  actions in solution
     1030018  microseconds elapsed
        8931  nodes expanded
        2270  nodes in memory

L29.txt, BFS
          31  actions in solution
      458803  microseconds elapsed
        4327  nodes expanded
        2892  nodes in memory

L29.txt, DFS
Timed out after 20 seconds

L29.txt, IDS
Timed out after 20 seconds

L29.txt, A*
          31  actions in solution
      901143  microseconds elapsed
        4289  nodes expanded
        3295  nodes in memory

L29.txt, IDA*
          31  actions in solution
    12596793  microseconds elapsed
       56728  nodes expanded
        3295  nodes in memory

L30.txt, BFS
          32  actions in solution
      106944  microseconds elapsed
        1163  nodes expanded
         718  nodes in memory

L30.txt, DFS
         871  actions in solution
      517948  microseconds elapsed
        1604  nodes expanded
        3785  nodes in memory

L30.txt, IDS
          32  actions in solution
```

```
     5871510  microseconds elapsed
       87934  nodes expanded
         102  nodes in memory

L30.txt, A*
          32  actions in solution
      179646  microseconds elapsed
        1077  nodes expanded
         720  nodes in memory

L30.txt, IDA*
          32  actions in solution
     1281643  microseconds elapsed
       10906  nodes expanded
         720  nodes in memory

L31.txt, BFS
          37  actions in solution
      412665  microseconds elapsed
        3975  nodes expanded
        2287  nodes in memory

L31.txt, DFS
Timed out after 20 seconds

L31.txt, IDS
Timed out after 20 seconds

L31.txt, A*
          37  actions in solution
      782316  microseconds elapsed
        3865  nodes expanded
        2632  nodes in memory

L31.txt, IDA*
          37  actions in solution
     9864524  microseconds elapsed
       52860  nodes expanded
        2632  nodes in memory

L40.txt, BFS
          51  actions in solution
      421974  microseconds elapsed
        3024  nodes expanded
        2136  nodes in memory

L40.txt, DFS
        1584  actions in solution
      670278  microseconds elapsed
        1684  nodes expanded
        8213  nodes in memory

L40.txt, IDS
Timed out after 20 seconds
```

```
L40.txt, A*
          51  actions in solution
      520607  microseconds elapsed
        2806  nodes expanded
        2475  nodes in memory

L40.txt, IDA*
          51  actions in solution
     6940203  microseconds elapsed
       43861  nodes expanded
        2423  nodes in memory
```

D Link to Complete Project

The complete project files can be found at:

`https://github.com/wj3ng/rush-hour-ai`