Algorithm Comparison for Planning Problem

Mahta Ramezanian
mahta.ramezanian@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

The goal is to compare the Critical Path Method (CPM) with non-deterministic search algorithms to find a measure that indicates which algorithm is the best for a given scenario and to help future projects decide which method would work best for their case. So far the results of a heuristic analysis and some uninformed search algorithms has been acquired to be compared to the CPM on different levels in the next phase.

Introduction

Automated planning and scheduling, as the name suggests, is about organizing the sequences of tasks to reach a specific goal. The steps to to achieve that goal are called *actions* and each action leads the project to a specific *state*, e.g. the final state is the goal or destination (Ghallab, Nau, and Traverso 2004).

One expects the planning algorithms to find three parameters for each action: the earliest incident time, the latest incident time, and the slack (float) time of each action (LaValle 2006).

The efficiency of planning has been widely studied. The goal of this project is not only to examine the execution time of different algorithms, but also the quality of the results of those algorithms. For this purpose, A fully deterministic algorithm (CPM) with the perfect result is compared with uninformed algorithms (Breadth-First Search (BFS), Depth-First Search (DFS), and uniform cost search) and A* search with various heuristics. The realm of the planning problems is quite vast and each problem demand an specific approach; however, many of them can be categorized as follows.

- Based on the **actions**
  - If they are deterministic or non-deterministic.
    - For the non-deterministic cases, finding the associated probabilities might require a sophisticated method.
  - Duration of each action
    - If multitasking possible
- Based on the **states**
  - Continuity of states
    - For the discrete case, if the possibilities finite or infinite
  - Observability of states
  - Number of the initial states (if it is one, finite or infinite
- Based on the **objective**
  - The definition of cost and reward
  - If the goal is to reach (a) specific state(s) or to maximize the reward
- Based on the **agents**
  - Number of agents and availabilities
  - If the agents are cooperative or competitive
  - If the agents decide individually or the plans are constructed centrally

The algorithms are implemented on the air cargo transportation planning problem addressed in chapter 10 of (Russell and Norvig 2016) involving delivering and receiving cargo and flying it from airport to airport. The performance measure for this problem is both the execution time, plan length, and the elapsed time for running the algorithm. The length of the path each parcel should take to reach the destination is widely variable to examine each algorithm for efficiency.

The assumptions for this project are discussed in the Methodology and the problem is defined in Planning Domain Definition Language (PDDL) (McDermott et al. 1998).

Although the techniques implemented so far are established methods, this project is dedicated to find a measure to assign the best method for a given case. That being said, the results can be used for many diverse examples in traveling salesman (Lin and Kernighan 1973), motion problems (Latombe 2012), robotics, cell-to-cell communications (Chen, Zhu, and Hu 2010), production plan (Nahmias and Cheng 2005), and etc.

By this time, preliminary results for non-deterministic algorithms have suggest advantage of uniform cost search among the uninformed algorithms; however, both the efficiency and the quality of heuristic searches far exceeds those results.

Planning problems date back to the salesman problem and nowadays have a vast domain of application. Hence, it is hard to point out state of the art in this area since studies are on various types of problem. To name a few, there are applications in communication, robotics, management, and service delivery. The cargo problem studied in this project can be extended to many of resource-allocation problems; however, further studies are needed for more complicated graphs i.g. temporal graphs.

This document closely follows the milestone for this project which was submitted as a course component in July 2019. However, since then the implementation of CPM algorithm has been completed, the results of all algorithms are now carefully investigated.

Related Work

The heuristics for the informed search can be either domain-independent (heuristics based on the logical structure of the problem) or domain-specific. In order to find out which approach is more efficient, A* search with both the ignore-precondition and level-sum heuristics is implemented. (McDermott et al. 1998) gives a formulation for PDDL problems by introducing the following components:

- Objects: Cargo, airports, and flights
- Predicates: True/false arguments about the objects, e.g. if the cargo c_1 is in the airport a_1 or not.
- Initial state: Given set of predicates for all the objects.
- Goal specification: The desired state
- Actions/Operators: Flies/loads/unloads

Planning tasks specified should separated into two files:

1. A domain file containing predicates and actions.
2. A problem file containing objects, initial state and goal specification.

That being said, one can conclude that PDDL restricts the problem.he planning problems are very diverse. (Ghallab, Nau, and Traverso 2004) suggests assumptions that significantly simplify the problem while keeping it applicable to real-world systems. Those assumptions are as follows.

1. Finite system (Finite number of objects in the system)
2. Fully observable
3. Deterministic actions with single outcome (i.e. in our case, the flights and shippings never fail)
4. Static (the states change only with the defined actions)
5. Attainment goals (the objective is to reach a designated state)
6. Sequential plans (linearly ordered sequence of actions $(a_1, a_2, , a_n)$
7. Implicit time (no time duration; the states are instantaneous)
8. Off-line planning

Figure 1 is the graph of the automated planning with the aforementioned assumptions

The efficiency of the CPM algorithm have been studied for years throughout different problem sizes, composite methods, various activity times, and different comparison methods. In (Chang et al. 1995), *Fuzzy Delphi method*

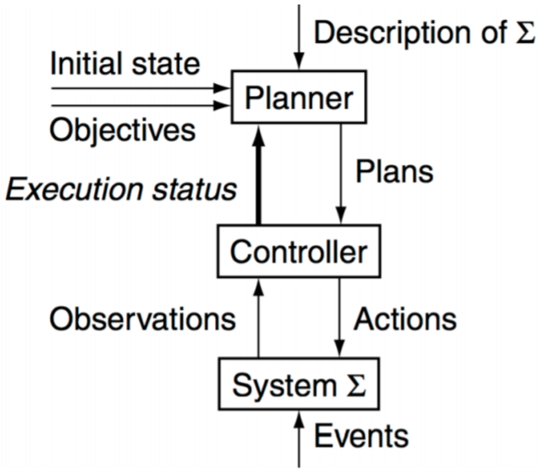


Figure 1: Classical planning with the restrictive assumptions

(Kaufmann and Gupta 1991) is used to forecast reliable time interval of each action. Then CPM (Critical Path Method) and PERT (Project Evaluation and Review Technique) are implemented on these deterministic time estimate to calculate the fuzzy project completion time and the degree of criticality for each task in the project. This method is specially interesting as it expands the application of this project to more relaxed assumptions i.e. when the time of each action is not implicit. which is out of the scope of this project. A more useful paper to direct this research could be (Vanhoucke and Debels 2007). This paper suggest a simple formulation and computational results for four different types of discrete time/cost trade-off (DTCT) problems. (Vanhoucke and Debels 2007) Also introduces a meta-heuristic method which provides near-optimal heuristic solutions for DTCT problems. This method consists of three simple steps: 1)initialization, 2)neighbourhood search, and 3) diversification. The seudo-code of this method is shown below.

```
Procedure Heuristic_search
    Initialization
    While (!stop criterion)
        Neighborhood search
        Diversification
    End while
    Return
```

Although this paper focuses on the implementation of ignore-preconditions and levelsum heuristics as the instances of informed search, the formulation and the comparison method of the latter paper is used and it is described in the methodology section.

So far, our results are consistent with the results in (Vanhoucke and Debels 2007): The heuristic search (although heuristics are different) has the best performance. As one clearly expects though, meta-heuristic does not outperform CPM. It is interesting to see if that happens with the heuristics of this study or not. Also, as in my code, the infeasibility rate is always $\%0$.

Methodology

Dataset

The dataset from this project is defined in (Russell and Norvig 2016). PDDL enables us to express all the objects (the **initial state**, the **actions** that are available in a state, the **result** of applying an action, and the **goal test**) with one schema as follows.

- State: conjunction/*set* of fluents that are true simultaneously. E.g. $At(Truck1, Melbourne)\&At(Truck2, Sydney)$ corresponds to state in a package delivery problem.
- Action: set of action schemas that implicitly define the $ACTIONS(s)$ and $RESULT(s, a)$ functions needed to do a problem-solving search. The example below describes an action schema for flying plane p from one place to another.

  $Action(Fly(p, from, to)$,
  PRECOND:$At(p, from)\&Plane(p)\&$
  $Airport(from)\&Airport(to)$
  EFFECT: $At(p, from)\&At(p, to))$
- Initial state: conjunction of ground objects. Goal: conjunction of literals (positive or negative) that may contain variables. E.g. $At(p, SFO)Plane(p)$

In this project, the problem has been restricted to the 8 assumptions introduced in (Ghallab, Nau, and Traverso 2004) which where counted in the literature review.

With this language defined, the dataset can be synthesised similar to this one from AIND planing problem (Sheahen 2017):

```
def air_cargo_problem():
    cargos = ['C1', 'C2']
    planes = ['P1', 'P2']
    airports = ['JFK', 'SFO']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, JFK)'),
           expr('At(P1, SFO)'),
           expr('At(P2, JFK)')]
    neg = [expr('At(C2, SFO)'),
           expr('In(C2, P1)'),
           expr('In(C2, P2)'),
           expr('At(C1, JFK)'),
           expr('In(C1, P1)'),
           expr('In(C1, P2)'),
           expr('At(P1, JFK)'),
           expr('At(P2, SFO)')]
    init = FluentState(pos, neg)
    goal = [expr('At(C1, JFK)'),
            expr('At(C2, SFO)')]
```

The size of the dataset could be made arbitrarily long. It would be plausible to find out how the time/cost tradeoff changes versus the size of the date. As a sample example, consider having P_1 and C_1 initially at SFO and P_2 and C_2 initially at JFK. Imagine the goal is to deliver $(unload)C_1$ to JFK and C_2 to SFO. The schematic tree for such procedure is shown in Figure 2 (note that several inner branches are not shown). Figure 2 is a visualization of how progression planning problems can be solved with search and scheduling algorithms. The graph for this particular example, consists of 52 nodes (which CPM expands and evaluate them all). However, the problem can be solved without expanding all the nodes.

Problems

For more accuracy, this project investigates three different problems to compare the search algorithms with: Problem 0, 1, and 2. Problem 0 is a simpler problem with a smaller graph has been designed to compare CPM with the search algorithm. The search spaces for all the problems are provided in the appendix.

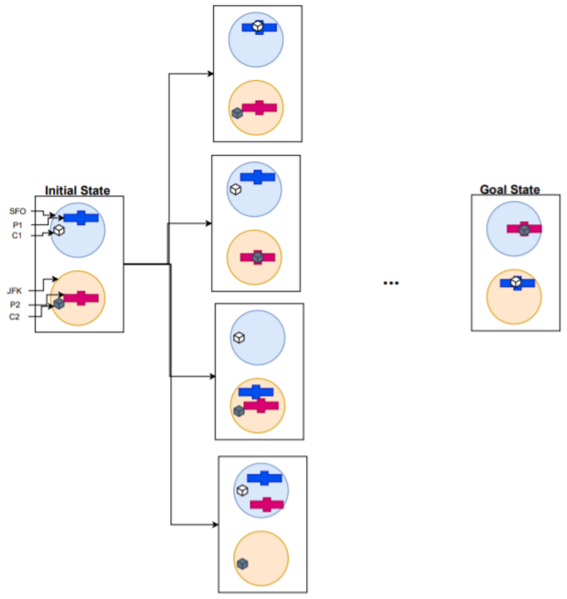


Figure 2: Schematic tree of a cargo problem with a given initial state and goal.

So far there are 6 candidates to be implemented on the data set: BFS, DFS, uniform cost search, A*(ignore-precondition and sum level heuristic), and finally CPM discussed in details in the subsections bellow.

Algorithms

Breadth First Search (BFS) In BFS, one explores the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes. As the name BFS suggests, one traverses the graph breadthwise:

1. Move horizontally and visit all the nodes of the current layer.
2. Move to the next layer (Moore 1959).

   The seudo-code is as follows.

```
BFS (G, s) //G: the graph
            //s: the source node
      let Q be queue.
```

```
    Q.enqueue( s ) //Insert in queue
                   //until all neighbour
                   //vertices are marked.

    mark s as visited.
    while ( Q not empty)
         v  =  Q.dequeue( )

        for all neighbours w of
                  v in Graph G
             if w is not visited
                     Q.enqueue( w )
                     //Stores w in Q
                     //to further visit
                     //its neighbour
                     mark w as visited.
```

In our problem, BFS guaranties the minimal path length.

Depth First Search (DFS) DFS starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. (Cormen et al. 2001). The seudo-code is as follows.

```
DFS(G,v):
    label v as discovered
    for all directed edges from v to w
    that are in G.adjacentEdges(v) do
        if vertex w is not labeled as
        discovered then recursively
        call DFS(G,w)
```

Uniform cost Search Unlike Depth First Search where the maximum depth had the maximum priority, Uniform Cost Search gives the minimum cumulative cost the maximum priority (Gayathri 2019). The seudo-code is as follows.

```
Insert the root into the queue
While the queue is not empty
      Dequeue the maximum priority node
      if path ends in the goal state
        return path
      else
            Insert all the children of
            the dequeued element, with
            the cumulative costs as
            the priority
```

A* Search A* find a path to the given goal from a designated source node having the smallest cost (least distance travelled, shortest time, etc.). The costs are determined with the heuristics.

The choice of the heuristics depend on the problem. Here two types of heuristics are tested on the A* Algorithm: Ignore-preconditions and sum-level Heuristic (Zeng and Church 2009).

Ignore-preconditions Heuristics Ignore preconditions Heuristic drops all preconditions from operations. Any goal condition can be achieves in one step (Hoffmann and Brafman 2005).

Level-sum Heuristics For multiple goal literals (g_i), level-sum heuristics is defined. as $\sum level - cost(g_i)$. One should remember that this heuristic maybe inadmissible.

Critical Path Method (CPM) In CPM approach, the critical path of executing the project is found. The critical path reveals the jobs that tolerate no *slack* and any delays in executing them result in at least one unit of lateness in total duration. It is proven that the critical path gives the shortest duration possible for finishing the project (Fondahl 1962)

Based on (Morovatdar et al. 2013), calculating the forward recursion of CPM is enough to find the critical path. One can achive the forward recursion of CPM simply by finding the shortest path from Dijkstras Algorithm (Dijkstra 1959) for finding the shortest path. The procedure is shown in the following psuedocode.

```
Require: Project network G = (V, E)
and Activity durations D.
Ensure: Set of all critical paths (Pn).
1: T1 = 0;
2: for i : 2 to n
3:      T_i = max{T_h+d_hi|h is in Pred_i}
4:      Pred_i* = {h||h is in Pred_i}}
5: next i
6: V* = {n}
7: for i : n to 2
8:      if i is in V* then V* = V* & Pred_i
9: next i
10: P1 = {P1_1} = {{1}};
11: for i : 2 to n
12:     if i is not in V* then next i
13:     m = 0 ;
14:     for h in Pred_i
15:         for l : 1 to |P_h|
16:             m = m + 1;
17:             P_i{m} = P_i{l}&{i}
18:         next l
19:     next h
20: next i
```

Where $Pred_i^*$ is the set of active predecessors realizing the i^{th} event, and V* is the set of critical events.

Because of the nature of the cargo problem, in this project *modified* CPM is used instead. The rationale for changing CPM algorithm comes from the objective of the cargo problem. CPM is specially useful when the path is important. In the described cargo problem, the cargo should be delivered to a specific airport as soon as possible and no one cares about the means. For this project, the nodes which take shorter to reach to are given priorities.

Results

Measure of Comparison (Vanhoucke and Debels 2007) introduces six indicators for comparing the results:

1. The instances for which an exact solution has been found.
2. Average percentage of deviation from the optimal solution.

3. The number of problem instances with a feasible though not necessarily an optimal solution
4. The problem instances with a feasible though not necessarily an optimal solution.
5. The number of created nodes is equal to the number of visited solutions (the values for these columns serve only to describe the hardness of the problem instances, but cannot be compared with each other)
6. Computational effort: average CPU usage for each approach.

With these in mind, as in this project all the problems are feasible and have optimal solutions by all the given algorithms, measure 3 and 4 are not taken into account. Instead, **Time elapsed, Number of expanded node, and the length of the proposed path** serve as the indicator of usefulness of each algorithm.

The Results of all the algorithms are shown in the tables below. Figure 3 shows the results for Problem 0 where CPM is included. One can deduce that although CPM takes a lot of time producing the whole graph and finding the path, it is still working better than depth limited graph search which is surprising. Also, BFS and greedy search give the optimal answer of CPM but with much less expansions, as the time elapsed for CPM is significantly higher for CPM than any other algorithm, one should prefer BFS or greedy search for this kind of problems.

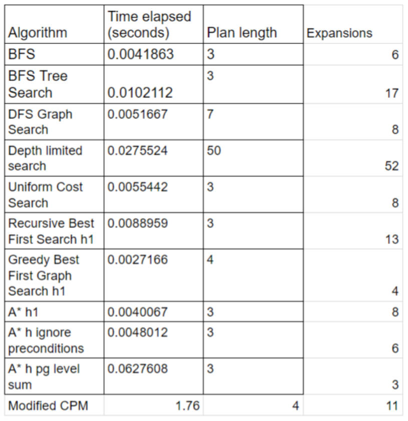
Few data in problem 0 may result in bad statistics. Hence, More complicated problems should be investigated. Figure 4 shows results for the case shown in Figure 2 (Problem 1) where P_1 and C_1 are at SFO and P_2 and C_2 are at JFK and the objective is to deliver C_1 to JFK and C_2 to SFO. The graph made by CPM results is too large and the author was unable to process CPM on this problem. Figure 4 shows the advantage of BFS to all the other algorithms with respect to all the measures introduced; however, using tree search while does not improve the optimality of answer, is significantly more process-intensive.

In all the given graphs, the processing time looks proportionate to the time elapsed which is expected, but this shows us how long it takes for CPM to solve problem 1 or 2.

The results for problem 2 are beyond our processing resources. Therefore, only three of the algorithms are tested. The results are as shown in 5. The best performance for DFS, in contrast to the last two cases.

Discussion and conclusion

The results clearly show that each problem requires a specific algorithm to tackle. During the research I found out that CPM is absolutely inapplicable for large datasets. BFS results in the optimal solution for all of our cases, and it is the fastest algorithm when the search space is small. However, in relatively larger datasets, DFS performs far better. *Discussion* section (∼1 pages) describes the implications of your results, the impact of your approach (i.e., to what extent does your approach help to improve the problem), and the limitations.



Algorithm	Time elapsed (seconds)	Plan length	Expansions
BFS	0.0041863	3	6
BFS Tree Search	0.0102112	3	17
DFS Graph Search	0.0051667	7	8
Depth limited search	0.0275524	50	52
Uniform Cost Search	0.0055442	3	8
Recursive Best First Search h1	0.0088959	3	13
Greedy Best First Graph Search h1	0.0027166	4	4
A* h1	0.0040067	3	8
A* h ignore preconditions	0.0048012	3	6
A* h pg level sum	0.0627608	3	3
Modified CPM	1.76	4	11

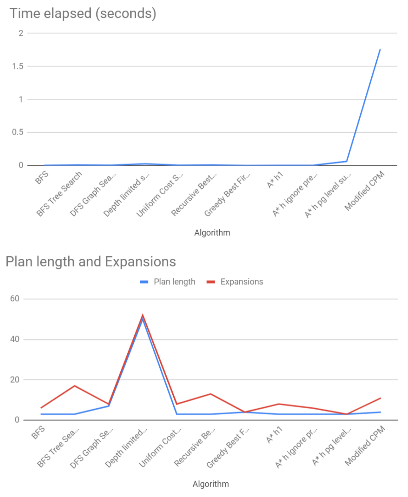


Figure 3: Results for Problem 0

The primary motive for this research was to investigate the algorithms of path choosing for planning problems. However, this can be extended to a far wider area of pathfinding which happens to be demanding in the era of reinforcement learning and robotics.

The computer used for this research has a 7th Gen Intel Core i7 processor with 7.88 GB of usable RAM. Although the Tech specs are below the requirement for this project, an understanding of deterministic and undeterministic algorithms is acquired by the experiments.

Further research can be done on more complicated paths: paths with too many nodes where the complexity is exponentially higher (i.e. in IP routing) or paths with no optimal solutions. Another interesting expansion would be real time analysis of path (applications in traffic, fluids, etc.).

References

[Chang et al. 1995] Chang, I. S.; Tsujimura, Y.; Gen, M.; and Tozawa, T. 1995. An efficient approach for large scale project planning based on fuzzy delphi method. *Fuzzy sets and systems* 76(3):277–288.

[Chen, Zhu, and Hu 2010] Chen, H.; Zhu, Y.; and Hu, K. 2010. Multi-colony bacteria foraging optimization with cell-to-cell communication for rfid network planning. *Applied Soft Computing* 10(2):539–547.

[Cormen et al. 2001] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm, year*.

[Dijkstra 1959] Dijkstra, E. W. 1959. *Communication with an automatic computer*. Ph.D. Dissertation, Excelsior.

[Fondahl 1962] Fondahl, J. W. 1962. A non-computer approach to the critical path method for the construction industry.

[Gayathri 2019] Gayathri, R. 2019. Comparative analysis of various uninformed searching algorithms in ai.

[Ghallab, Nau, and Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

[Hoffmann and Brafman 2005] Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005.

[Kaufmann and Gupta 1991] Kaufmann, A., and Gupta, M. M. 1991. Introduction to fuzzy arithmetic: Theory and applications. 1991. *VanNostrand Reinhold, New York*.

[Latombe 2012] Latombe, J.-C. 2012. *Robot motion planning*, volume 124. Springer Science & Business Media.

[LaValle 2006] LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

[Lin and Kernighan 1973] Lin, S., and Kernighan, B. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research* 21(2):498–516.

[McDermott et al. 1998] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

[Moore 1959] Moore, E. F. 1959. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, 285–292.

[Morovatdar et al. 2013] Morovatdar, R.; Aghaie, A.; Roghanian, E.; and ASL, H. A. 2013. An algorithm to obtain possibly critical paths in imprecise project networks.

[Nahmias and Cheng 2005] Nahmias, S., and Cheng, Y. 2005. *Production and operations analysis*, volume 6. McGraw-hill New York.

[Russell and Norvig 2016] Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[Sheahen 2017] Sheahen, D. 2017. Aind-planning. `https://github.com/keithblaha/AIND-Planning/tree/1296d9c7f7c17b11bda652bd7a3c01d3873eb3ac`.

[Vanhoucke and Debels 2007] Vanhoucke, M., and Debels, D. 2007. The discrete time/cost trade-off problem: extensions and heuristic procedures. *Journal of Scheduling* 10(4-5):311–326.

[Zeng and Church 2009] Zeng, W., and Church, R. L. 2009. Finding shortest paths on real road networks: the case for a. *International journal of geographical information science* 23(4):531–543.

Appendix: Problems

Problem 0

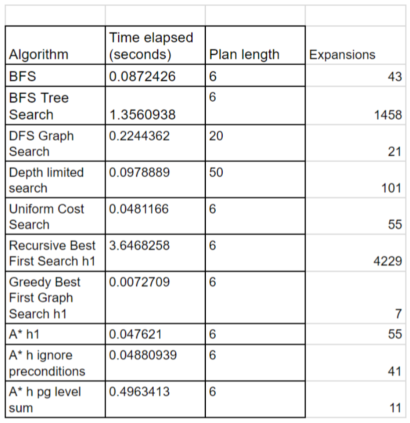
```
    cargos = ['C1', 'C2']
    planes = ['P1']
    airports = ['JFK', 'SFO']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, SFO)'),
           expr('At(P1, SFO)')]
    neg = [expr('At(C2, JFK)'),
           expr('In(C2, P1)'),
           expr('In(C2, P2)'),
           expr('At(C1, JFK)'),
           expr('In(C1, P1)'),
           expr('In(C1, P2)'),
           expr('At(P1, JFK)')]
    init = FluentState(pos, neg)
    goal = [expr('At(C1, JFK)')]
```

Problem 1

```
def air_cargo_problem():
    cargos = ['C1', 'C2']
    planes = ['P1', 'P2']
    airports = ['JFK', 'SFO']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, JFK)'),
           expr('At(P1, SFO)'),
           expr('At(P2, JFK)')]
    neg = [expr('At(C2, SFO)'),
           expr('In(C2, P1)'),
           expr('In(C2, P2)'),
           expr('At(C1, JFK)'),
           expr('In(C1, P1)'),
           expr('In(C1, P2)'),
           expr('At(P1, JFK)'),
           expr('At(P2, SFO)')]
    init = FluentState(pos, neg)
    goal = [expr('At(C1, JFK)'),
            expr('At(C2, SFO)')]
```

Problem 2

```
        cargos = ['C1', 'C2', 'C3', 'C4']
    planes = ['P1', 'P2']
    airports = ['JFK', 'SFO', 'ATL', 'ORD']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, JFK)'),
           expr('At(C3, ATL)'),
           expr('At(C4, ORD)'),
           expr('At(P1, SFO)'),
              expr('At(P2, JFK)')
              ]
neg = [expr('At(C1, JFK)'),
              expr('At(C1, ATL)'),
              expr('At(C1, ORD)'),
              expr('In(C1, P1)'),
              expr('In(C1, P2)'),
              expr('At(C2, SFO)'),
              expr('At(C2, ATL)'),
              expr('At(C2, ORD)'),
              expr('In(C2, P1)'),
              expr('In(C2, P2)'),
              expr('At(C3, SFO)'),
              expr('At(C3, JFK)'),
              expr('At(C3, ORD)'),
              expr('In(C3, P1)'),
              expr('In(C3, P2)'),
              expr('At(C4, SFO)'),
              expr('At(C4, JFK)'),
              expr('At(C4, ATL)'),
              expr('In(C4, P1)'),
              expr('In(C4, P2)'),
              expr('At(P1, JFK)'),
              expr('At(P1, ATL)'),
              expr('At(P1, ORD)'),
              expr('At(P2, SFO)'),
              expr('At(P2, ATL)'),
              expr('At(P2, ORD)')
              ]
goal = [expr('At(C1, JFK)'),
               expr('At(C3, JFK)'),
               expr('At(C2, SFO)'),
               expr('At(C4, SFO)')
               ]
init = FluentState(pos, neg)
```



Algorithm	Time elapsed (seconds)	Plan length	Expansions
BFS	0.0872426	6	43
BFS Tree Search	1.3560938	6	1458
DFS Graph Search	0.2244362	20	21
Depth limited search	0.0978889	50	101
Uniform Cost Search	0.0481166	6	55
Recursive Best First Search h1	3.6468258	6	4229
Greedy Best First Graph Search h1	0.0072709	6	7
A* h1	0.047621	6	55
A* h ignore preconditions	0.04880939	6	41
A* h pg level sum	0.4963413	6	11

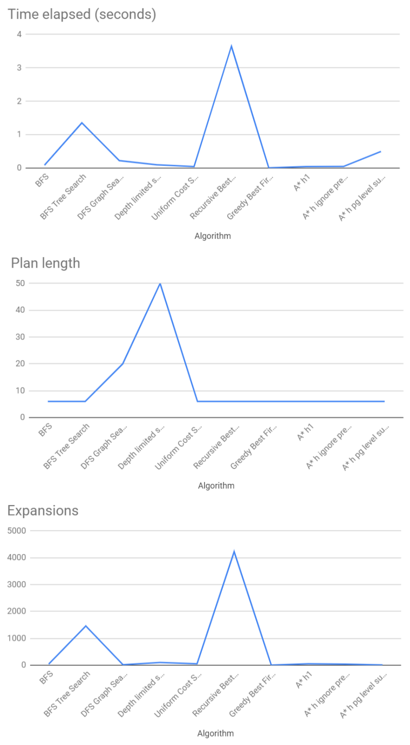
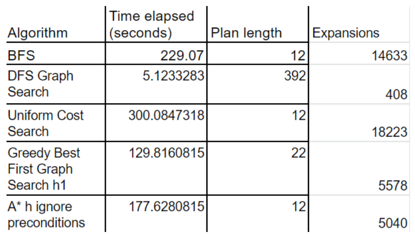


Figure 4: Results for Problem 1



Algorithm	Time elapsed (seconds)	Plan length	Expansions
BFS	229.07	12	14633
DFS Graph Search	5.1233283	392	408
Uniform Cost Search	300.0847318	12	18223
Greedy Best First Graph Search h1	129.8160815	22	5578
A* h ignore preconditions	177.6280815	12	5040

Figure 5: Results for Problem 2