COMPARISONS BETWEEN

STANDARD CONFLICT BASED SEARCH

AND

ITERATIVE DEEPENING CONFLICT BASED SEARCH

Hengyu Cui

Justin Yiu Ming Mok

Don Van

1. INTRODUCTION

With robotics and automation such as automated vehicles and robots being utilized in a variety of businesses such as warehouses and restaurants, the advancement of multi-agent path finding (MAPF) systems has been rapidly developing as well.

> MAPF is an important type of multi-agent planning problem in which the task is to plan paths for multiple agents, where the key constraint is that the agents will be able to follow these paths concurrently without them colliding. [1]
>
> Conflict Based-Search (CBS) is a common and popular two-level search-based MAPF algorithm which resolves collisions by adding constraints at a high level and computing paths consistent with those constraints at a low level. [2]

The top level of CBS searches on a conflict tree. In contrast, the lower level of CBS perform A* search on each agent based on its constraints. Unfortunately, the size of its conflict tree can increase drastically when poor choices are made in the CBS algorithm which would increase the runtime and the amount of memory being used. To counteract these faults, an Iterative Deepening Conflict Based Search (IDCBS) was introduced to address the issues.

The IDCBS was created to reduce the amount of memory used while theoretically not increasing its runtime asymptotically compared to CBS. To implement this iterative deepening search algorithm, we utilized a stack, depth-first node generation, and node ordering.

The aim of this project was to study the trade-offs between time and space complexities of CBS and IDCBS. These search algorithms were tested with 65 test cases of various complexities.

2. IMPLEMENTATION

CBS (Algorithm 1) and IDCBS (Algorithm 2) are both two level algorithms that utilize constraint tables to resolve MAPF instances. The difference between CBS and IDCBS is that IDCBS limits the depth of its search on every iteration allowing it to find optimal solutions without expanding branches too deeply which would have used up more resources than necessary.

Algorithm 1: CBS

```
CBS(MAPF instance):
    R ← new node
    R.constraints ← {}
    R.solution ← {a shortest path for each agent}
    R.cost ← totalCost(R.solution)
    R.conflicts ← findConflicts(R.solution)
    Insert R into Open
    While Open is not empty do
        N ← node with lowest cost in Open
        Delete N from Open
        If N has no conflict then
            Return N.solution
        Conflict ← Choose random Conflict(N)
        C ← GenerateConstraints(Conflict)
        Foreach Constraint c in C do
            N' ← Generate Child(N, c)
            Insert N' to Open
    Return "No solution"
```

Algorithm 2: IDCBS

```
IDCBS(MAPF instance):
    R ← new node
    R.constraints ← {}
    R.solution ← {a shortest path for each agent}
    R.cost ← totalCost(R.solution)
    R.conflicts ← findConflicts(R.solution)
    CostLimit = R.cost
    While CostLimit < 1000:
        Insert R into Open
        While Open is not empty do
            N ← node with lowest cost in Open
            Delete N from Open
            If N.cost > CostLimit:
                Continue to next iteration
            If N has no conflict then
                Return N.solution
            Conflict ← Choose random Conflict(N)
            C ← GenerateConstraints(Conflict)
            Foreach Constraint c in C do
                N' ← Generate Child(N, c)
                Insert N' to Open
        CostLimit ← CostLimit + 1
    Return "No solution"
```

3. METHODOLOGY

To analyze the space and time complexities of CBS and IDCBS in detail, three comparisons were used to create the necessary graphs used for analysis. For each comparison, the test-cases were executed 100 times for each algorithm with the average and median being inputted on a spreadsheet. (Figure 1)

	A	B	C	D	E	F	G	H
1	File	cost	time	expanded	generated	maximum nodes	Initial collision count	sampleSize
2	instances\test_01.txt	41	0.014333	19.66	35.34	292.7	6	100
3	instances\test_02.txt	18	0.001047	1	1	11	0	100
4	instances\test_03.txt	28	0.000957	1	1	19	0	100
5	instances\test_04.txt	32	0.004728	9.23	17.05	106.26	2	100
6	instances\test_05.txt	26	0.003447	7.14	11.92	97.93	2	100
7	instances\test_06.txt	24	0.001666	3	5	56.05	2	100
8	instances\test_07.txt	34	0.001691	2	3	66	1	100
9	instances\test_08.txt	38	0.033985	58.15	103.8	176.81	4	100
10	instances\test_09.txt	24	0.001099	1	1	49	0	100
11	instances\test_10.txt	19	0.002234	3.79	6.58	74.25	2	100
12	instances\test_11.txt	35	0.004461	3.51	5.51	184.8	1	100
13	instances\test_12.txt	36	0.003542	6.34	11.34	68.89	4	100
14	instances\test_13.txt	36	0.005722	8.97	15.29	129.52	5	100
15	instances\test_14.txt	24	0.001702	2	3	60	1	100
16	instances\test_15.txt	50	0.017569	23.04	41	202.95	3	100
17	instances\test_16.txt	51	0.082466	64.66	96.22	333.4	4	100
18	instances\test_17.txt	39	0.001988	1	1	50	0	100
19	instances\test_18.txt	32	0.005698	8.56	12.48	78.66	2	100
20	instances\test_19.txt	47	0.005614	4.91	8.3	160.66	3	100
21	instances\test_20.txt	28	0.002177	2	3	68.58	1	100
22	instances\test_21.txt	46	0.00547	5.83	10.43	115.77	5	100
23	instances\test_22.txt	51	0.009337	13.54	23.92	84.9	3	100
24	instances\test_23.txt	32	0.002305	2.44	3.88	53.17	2	100
25	instances\test_24.txt	47	0.008345	12.92	24.81	87.23	4	100

Figure 1

Initially, the program ran on all of the test cases using both search algorithms. For this comparison, the runtimes and cost of each test case were the factors used from the spreadsheet. The average and median of each test case's runtime and cost would be used to plot the graph. (Figure 2)

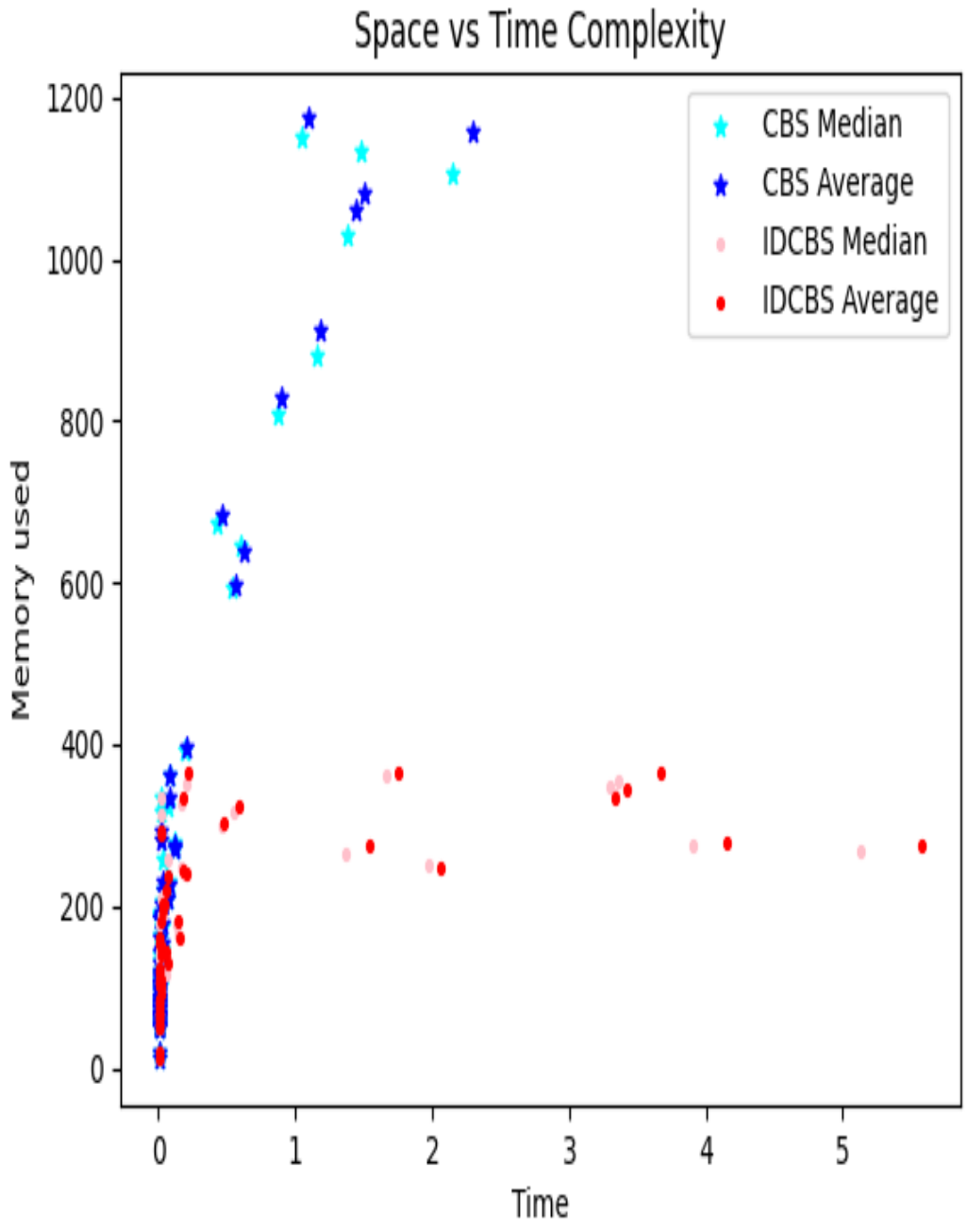

Figure 2

The second comparison was between the time complexity of each algorithm and the size of each test case. This graph showed the relationship between the number of initial conflicts to the time it took to finish each test case after running each case 100 times on each algorithm. (Figure 3)

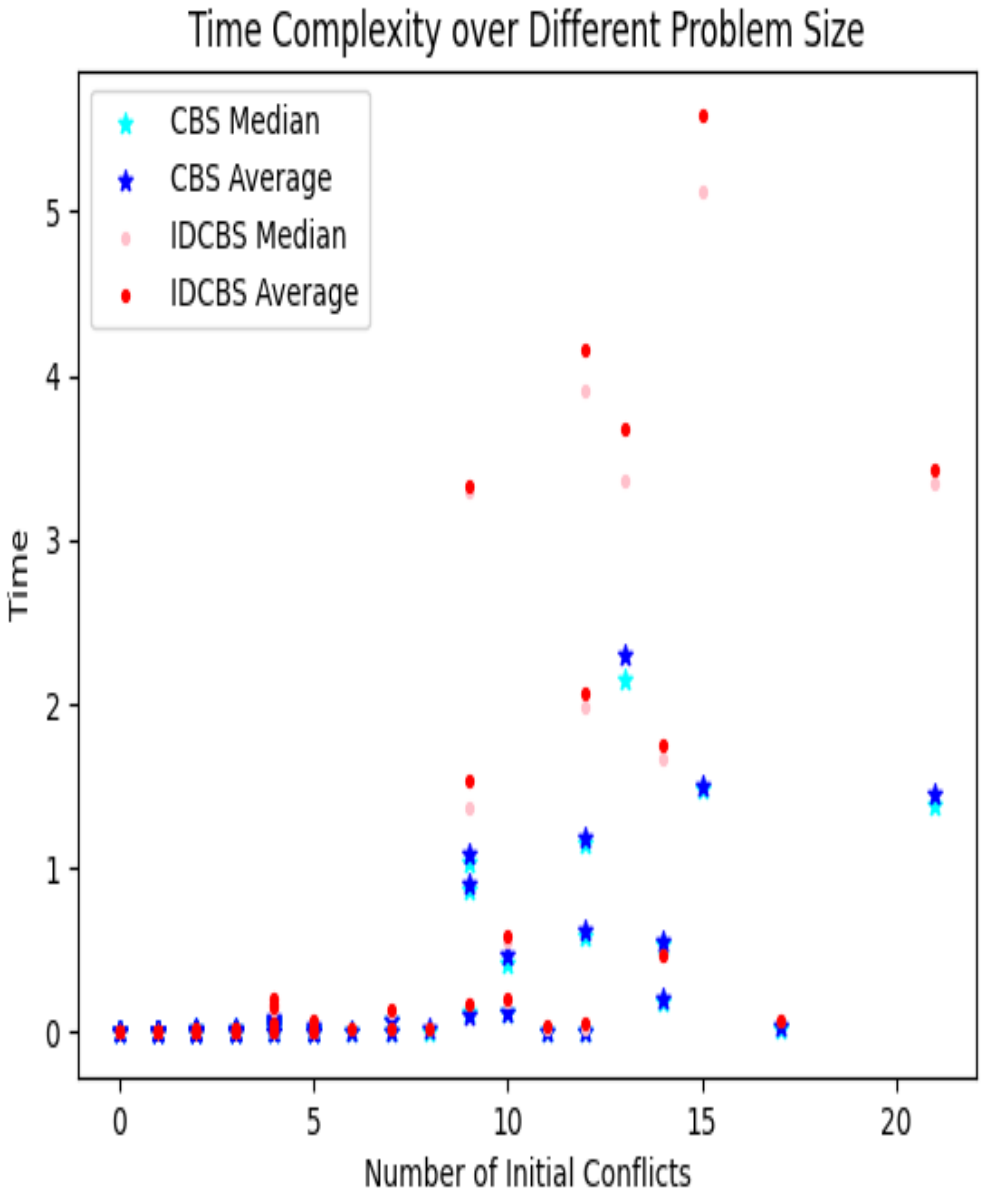

Figure 3

The third comparison was to test the space complexity of each algorithm based on the size of the test case. For this, we used the number of initial conflicts with the average and median of the memory used for each test case. (Figure 4)

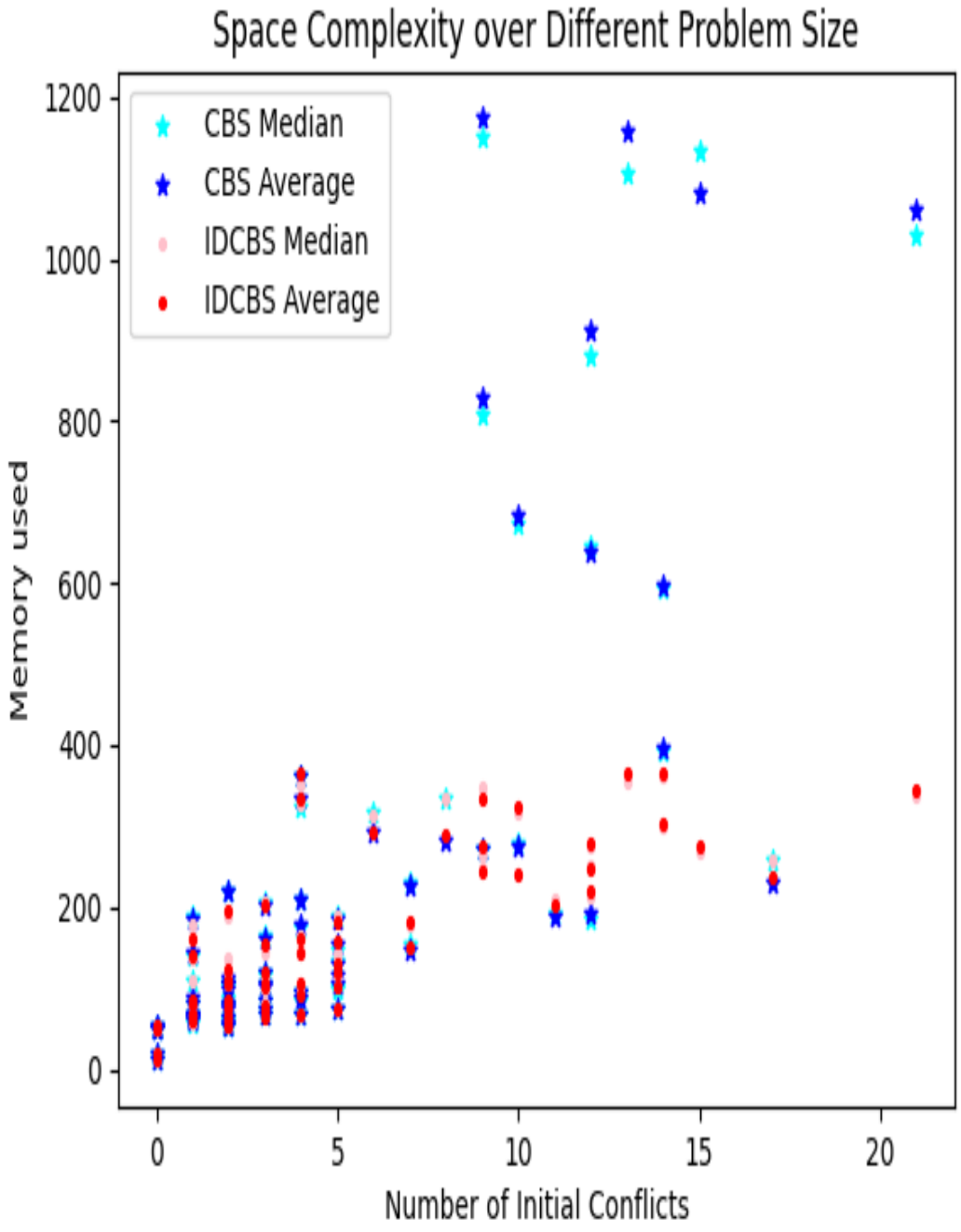

Figure 4

4. EXPERIMENTAL SETUP

The experiments were tested on a laptop with Windows 10 on an Intel i7-8550U as its processor. This processor has a base clock speed of 1.80 GHz and a boost clock speed of 4.00 GHz. The language being used to run these experiments was Python 3.8.2.

The code utilized for this project was provided by our professor that was modified to fit the goals of this project (Figure 5 - 6).

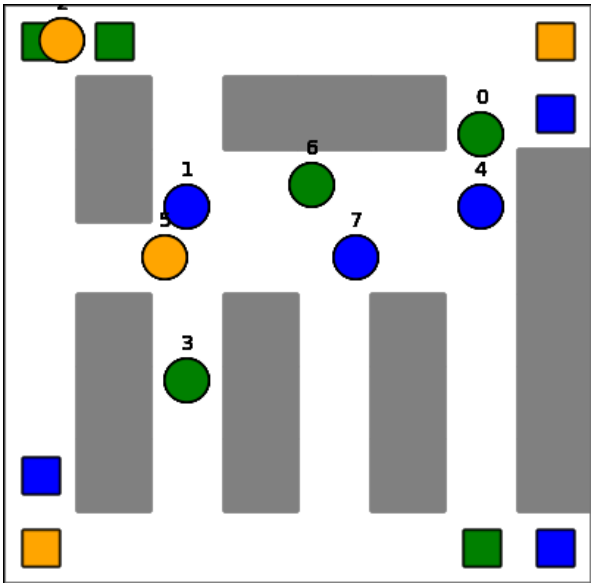

Figure 5

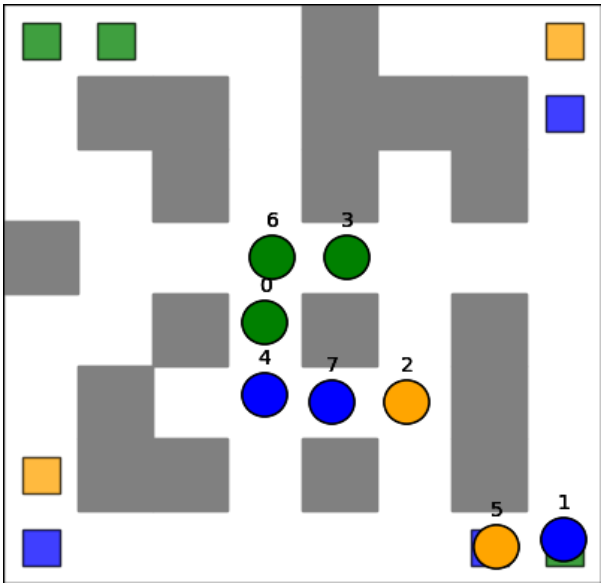

Figure 6

5. RESULTS

CBS would tend to be faster, but looked like it quickly expanded to use an excessive amount of memory as the memory requirement grew exponentially with the runtime. On the other hand, IDCBS limited its memory usage but its time to complete also increased rapidly, sometimes doubling or even tripling the amount of time used by CBS on larger test cases. This meant that while we would have preferred to use CBS for faster computations on MAPF instances, we would have been forced to use IDCBS due to the physical limitations of modern computing as the required amount of memory increases exponentially. As a result, the number of initial conflicts primarily affected the time complexity of both search algorithms. However, the amount of initial conflicts severely affected the space-complexity for CBS but only minorly affected IDCBS.

6. CONCLUSION

This project primarily focused on the trade-offs between time and space complexities of CBS and IDCBS by running various test cases. We gathered the required information needed such as the number of initial conflicts, runtime, and memory usage to create the required graphs to analyze our output.

Based on the graphs that were created, CBS ran faster but used a greater amount of memory as time increased. Also, the number of initial conflicts affected the space complexity for CBS but only minutely affected IDCBS.

If we were to continue this experiment we would like to test more randomly generated test cases regardless of how long it would take to resolve the instances. To accomplish this we would need hardware that could be completely dedicated to generating and solving MAPF instances to reduce the amount of external variables affecting the runtime.

Reference List

[1] R. Stern *et al.*, “Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks”, *CoRR*, vol abs/1906.08291, 2019.

[2] J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig, “Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search,” *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 442–449, 2019.