Iterative-Deepening Conflict Based Search
Final Report

Abstract

For Multi-Agent Path Finding (MAPF) problems in the real world, in robotics and maps, Conflict Based Search (CBS) is the leading algorithm to find solutions. However, CBS faces time and memory limiting problems when faced with larger problem instances. This can be improved by implementing an Iterative-Deepening version of Conflict Based Search (IDCBS). This will theoretically allow the algorithm to improve time-wise and memory-wise. We will test these algorithms through multiple test cases, and compare results throughout the testing.

Introduction

Conflict Based Search (CBS) is a leading algorithm used to solve Multi-Agent Path Finding problems (MAPF) in the real world, whether it be with robotics, systems, etc. The usual search is performed using some sort of A* search, and will find a solution that may or may not be optimal. The problem arises when dealing with large/numerous MAPF problems with A*, it is memory-intensive,and unable to handle larger problem instances due to exponential growth in memory usage, making it infeasible for large scale problems. For our baseline, we will add a depth limiter to the original CBS with disjoint splitting, and take data points from this.

In this project, we will implement an iterative-deepening conflict based search (IDCBS) in order to find a way to solve larger problems without the excessive memory consumption. In essence, we will utilize the A* search, but with iterative deepening implemented which combines CBS with iterative-deepening depth first search. By searching the space within a bounded memory, IDCBS will aim to reduce memory usage, while maintaining or improving the quality of solutions. To achieve this, we will also consider enhancing CBS with incremental algorithms like Lifelong Planning A* (LPA*) at the low level. Our implementation will be tested on many test cases, and run with diagnostic outputs to compare cost, memory usage, runtime and solution quality.

Implementation

For our project, we decided to test and compare 3 different algorithms. Our baseline algorithm is the CBS code with depth limiting search (DLS). Then, the 2 algorithms we will test and compare are the Iterative-deepening conflict based search (IDCBS) and the CBS with Lifelong Planning A* (LPA*).

CBS with DLS

For this simple addition to our baseline, we add a depth limiting if statement.
As CBS is known for its reliance on a best-first strategy, although it has proved to be an effective strategy, it results in high memory usage as all open nodes are stored during the search. By adding a depth limiting if statement (DLS), we are able to address this issue. The core idea of adding a DLS is to limit the depth of exploration in the conflict tree (CT) during each iteration. Effectively pruning nodes that exceed a predefined depth limit. By focusing the search within a constrained depth range, DLS significantly reduces memory consumption while ensuring that solutions within the specified limit are thoroughly explored.
To implement DLS, we introduce a depth attribute for each node in the CT. During the search, any node that exceeds the depth limit is pruned. This way we can ensure that memory usage remains constrained. This focused exploration within a bounded depth scope allows the algorithm to prioritize solutions that are reachable within the specified limit, reducing the need to maintain a large number of nodes in memory. The introduction of DLS enables CBS to remain efficient and practical in scenarios where the search space is infinite and the risk of memory exhaustion is high.
The downside to DLS is that while it helps to preserve memory usage, if the solutions are beyond the depth limit, it would not find any solutions even if one does exist.

IDCBS

The algorithm that follows is how to implement the interactive deepening CBS with either standard splitting/disjoint splitting. Essentially it is as follows:
Using a stack object, solve with your CBS A* function, then if solution is not found, then start an iterative-deepening CBS A* solution search, and continue to search ever more deeply, until either a solution is found or no solution is found.

Algorithm 2: IDCBS with disjoint splitting

Result: An Optimal Solution or Exception (No Solution)

```
Compute the heuristics;
Initialized an open_list with stack;
Get the root node;
Push root into open_list;
threshold = the cost of root node;
// Implement DFS on the conflict tree with disjoint splitting
solution = DFS(disjoint_splitting);
while solution == False do
    reset(); // Reset the open_list
    Push root into open_list;
    solution = DFS(disjoint_splitting);
    if threshold doesn't change and solution == False then
        raise Exception (No solution);
    end
end
return solution;
```

CBS with LPA*

In order to increase the efficiency of CBS, we implement CBS with the incremental algorithm Lifelong Planning A* (LPA*). This is because traditional CBS employs A* search at the low level to compute individual agent paths. The problem arises whenever new constraints are introduced as this leads A* to redundant computations as it has to recalculate paths it has already calculated through prior computations. This can become especially bothersome when a change is minor. In order to address this inefficiency, we replaced A* search with the LPA*. LPA* resolves this inefficiency by reusing prior computations prior computations and updating only the affected portions of the graph when changes occur. This is achieved by maintaining data structures such as g-values (the cost of the shortest path to a node) and rhs-values (one-step lookahead costs), which enable LPA* to efficiently identify and propagate the impacts of constraint changes.

To implement LPA* into CBS we first had to adjust the low level pathfinding for a single agent in such a way to allow for proper utilization of the benefits of LPA*. To achieve this goal we created and maintained a priority queue of inconsistent nodes that handled edge updates caused by vertex or edge constraints. For integration with CBS, LPA* is modified to accommodate the time-expanded graph structure of MAPF, where each vertex represents a location and timestamp. Furthermore, to prioritize paths that minimizes conflicts, LPA* incorporates a tie-breaking mechanism based on the number of conflicts recorded in a global conflict avoidance table (CAT). Through this adaptation we are able to significantly accelerate low-level search by focusing only on the parts of the graph that are directly impacted by constraints, thus eliminating the need for a full recomputation and increasing efficiency.

Methodology

To evaluate the improvements/changes from CBS to IDCBS and IDCBS with LPA*, we will outline questions to be answered and evaluation metrics to measure.

Questions:
How does IDCBS perform compared to CBS in terms of runtime and memory usage across multiple test instances?
Can IDCBS solve larger instances that are infeasible for CBS due to memory constraints?
Does IDCBS maintain quality and optimality from the CBS algorithm?
Are there specific instances where IDCBS outperforms CBS, or vice versa?

Metrics for Evaluation:

- Runtime performance
- Memory usage
- Solution quality and optimality

We aim to provide a solution to the time and space limiting problems of CBS search by implementing an IDCBS approach. Through IDCBS we are able to directly address the memory inefficiencies associated with standard CBS by gradually expanding the search space through iterative-deepening techniques. In our implementation, we start by integrating a standard CBS A* search algorithm with iterative-deepening. This allows us to explore increasingly larger depth systematically while reducing the memory overhead required. Next we implement the incremental search technique LPA* instead of A* for low-level pathfinding in order to improve efficiency. To evaluate the effectiveness of IDCBS over CBS, we will test both algorithms over a variety of MAPF problem instances and measure the runtime and memory usage of both algorithms.

Experimental Setup

The experiments was conducted in the following environment:

- Programming Language: Python 3.10.
- Operating System: Ubuntu 20.04.
- Processor: Intel Core i7-11700K, 3.6 GHz, 8 cores.
- Memory: 32 GB RAM.
- Tools and Libraries:
    - network for graph representation.
    - matplotlib for visualizing paths and results.
    - Custom implementations of CBS and IDCBS algorithms.

Experimental Results

Raw data:

CBS

Instance	Cost	CPU Time	Expanded Nodes	Generated Nodes
instances\test_1.txt	41	16.09547067	13076	25873
instances\test_10.txt	19	0.045287609	4	7
instances\test_11.txt	35	0.009753704	3	5
instances\test_12.txt	36	0.00808382	6	11
instances\test_13.txt	36	0.01102519	9	17
instances\test_14.txt	24	0.000535965	2	3
instances\test_15.txt	50	0.020222664	12	23
instances\test_16.txt	51	0.78807807	759	1035
instances\test_17.txt	39	0.00151968	1	1
instances\test_18.txt	32	0.00840807	7	13
instances\test_19.txt	47	0.006367445	6	11
instances\test_2.txt	18	0	1	1
instances\test_20.txt	28	0.003118277	2	3
instances\test_21.txt	46	0.005856752	4	7
instances\test_22.txt	51	0.009907007	8	15
instances\test_23.txt	32	0.003968716	2	3
instances\test_24.txt	47	0.014125586	13	25
instances\test_25.txt	40	0.008415937	6	11
instances\test_26.txt	42	0.006459236	5	9
instances\test_27.txt	40	0.008079052	6	11
instances\test_28.txt	41	0.011085272	8	15
instances\test_29.txt	48	0.01674962	14	27
instances\test_3.txt	28	0.002005339	1	1
instances\test_30.txt	43	0.44969511	341	557
instances\test_31.txt	39	0.009999752	9	17
instances\test_32.txt	30	0.003000498	4	7
instances\test_33.txt	28	0.00987196	10	19
instances\test_34.txt	33	0.002005339	3	5
instances\test_35.txt	30	0.003005266	3	5
instances\test_36.txt	23	0.001451731	2	3
instances\test_37.txt	38	0.033166647	61	121
instances\test_38.txt	28	0.005491018	1	1
instances\test_39.txt	35	0.003177643	3	5
instances\test_4.txt	32	0.016779423	18	35
instances\test_40.txt	24	0.001000404	2	3
instances\test_41.txt	45	0.690060377	629	939
instances\test_42.txt	57	0.394371748	295	533
instances\test_43.txt	43	0.062758923	60	99
instances\test_44.txt	33	0.008051634	11	21
instances\test_45.txt	24	0.001507998	2	3
instances\test_46.txt	57	0.030081511	28	51
instances\test_47.txt				
instances\test_48.txt	36	0.004961729	5	9
instances\test_49.txt	42	0.010012388	7	13
instances\test_5.txt	26	0.005180836	6	11
instances\test_50.txt	48	1.400671482	1008	1027
instances\test_6.txt	24	0.005097389	3	5
instances\test_7.txt	34	0.005292177	2	3
instances\test_8.txt	38	0.227816105	230	347
instances\test_9.txt	24	0.005945444	1	1

IDCBS results

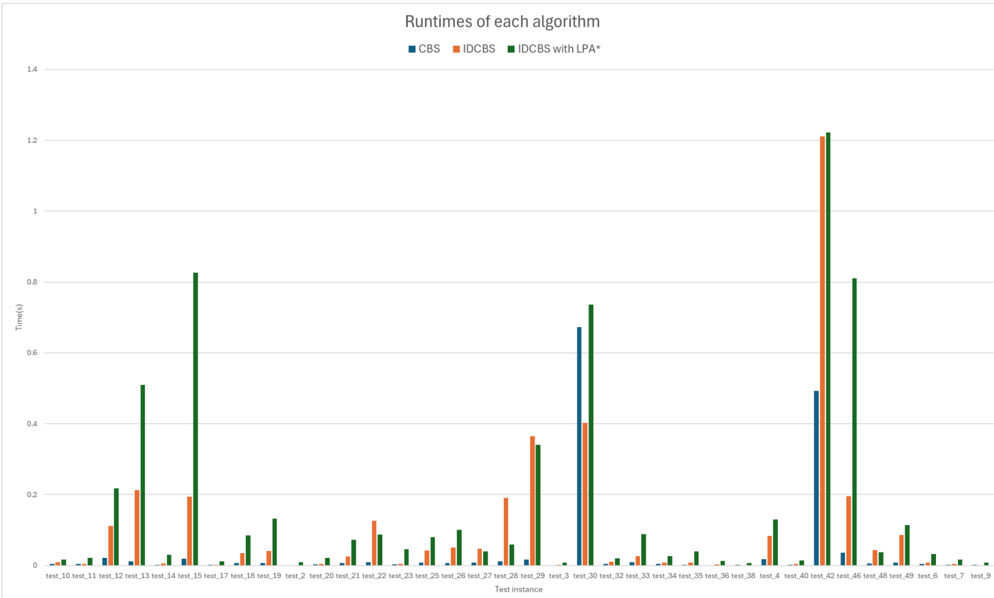
Instance	Cost	CPU Time	Expanded Nodes	Generated Nodes
instances\test_1.txt	42	1.153938293	573	725
instances\test_10.txt	20	0.009125948	5	7
instances\test_11.txt	38	0.005047321	5	5
instances\test_12.txt	36	0.111149073	6	11
instances\test_13.txt	36	0.213258743	18	27
instances\test_14.txt	24	0.005141258	2	3
instances\test_15.txt	50	0.194758654	12	23
instances\test_16.txt	51	0.686965704	337	413
instances\test_17.txt	39	0.002001524	1	1
instances\test_18.txt	32	0.035111189	7	13
instances\test_19.txt	47	0.041259527	6	11
instances\test_2.txt	18	0.000998735	1	1
instances\test_20.txt	28	0.003999472	2	3
instances\test_21.txt	46	0.025699854	4	7
instances\test_22.txt	52	0.12602067	62	89
instances\test_23.txt	32	0.005082846	2	3
instances\test_24.txt	47	0.085065126	13	25
instances\test_25.txt	40	0.041769743	6	11
instances\test_26.txt	42	0.051362991	5	9
instances\test_27.txt	40	0.046585083	6	11
instances\test_28.txt	41	0.191113234	8	15
instances\test_29.txt	48	0.365374565	14	27
instances\test_3.txt	28	0.002005816	1	1
instances\test_30.txt	43	0.402051687	159	191
instances\test_31.txt	39	0.084125042	9	17
instances\test_32.txt	31	0.010186911	6	9
instances\test_33.txt	31	0.026712418	25	27
instances\test_34.txt	33	0.008024931	3	5
instances\test_35.txt	30	0.008466482	3	5
instances\test_36.txt	23	0.003508806	2	3
instances\test_37.txt	38	0.022686243	22	27
instances\test_38.txt	28	0.00100112	1	1
instances\test_39.txt	35	0.009005785	3	5
instances\test_4.txt	32	0.083828449	18	35
instances\test_40.txt	24	0.003998995	2	3
instances\test_41.txt	45	1.323705912	557	749
instances\test_42.txt	57	1.211457729	314	479
instances\test_43.txt	46	0.128262281	69	73
instances\test_44.txt	34	0.046919107	28	45
instances\test_45.txt	24	0.002999783	2	3
instances\test_46.txt	57	0.19532156	23	35
instances\test_47.txt	69	372.8208911	123882	127845
instances\test_48.txt	36	0.043842316	5	9
instances\test_49.txt	42	0.085618258	7	13
instances\test_5.txt	26	0.032210588	6	11
instances\test_50.txt	48	0.0832932	36	39
instances\test_6.txt	24	0.008508682	3	5
instances\test_7.txt	34	0.003999949	2	3
instances\test_8.txt	41	0.034782171	28	29
instances\test_9.txt	24	0.001001835	1	1

IDCBS with LPA*

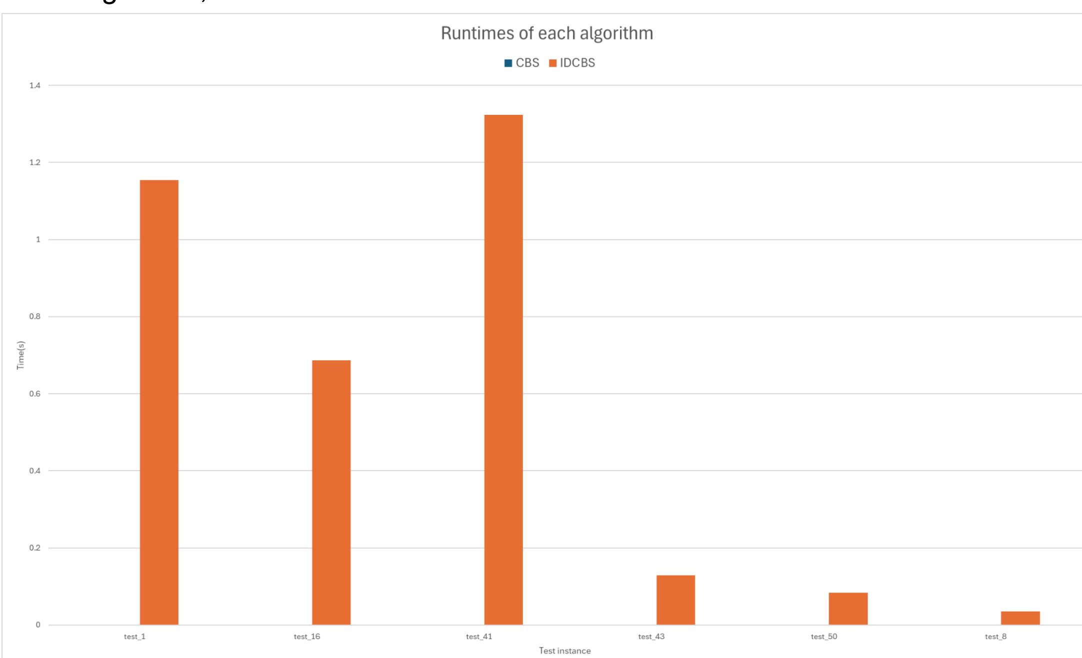
Instance	Cost	CPU Time	Expanded Nodes	Generated Nodes
instances\test_1.txt				
instances\test_10.txt	20	0.017019033	4	5
instances\test_11.txt	38	0.022027731	5	5
instances\test_12.txt	36	0.217746973	7	13
instances\test_13.txt	36	0.510110378	19	29
instances\test_14.txt	24	0.02969718	3	5
instances\test_15.txt	50	0.826478481	12	23
instances\test_16.txt				
instances\test_17.txt	39	0.011509657	1	1
instances\test_18.txt	32	0.084963083	7	13
instances\test_19.txt	47	0.132409334	23	31
instances\test_2.txt	18	0.009764194	1	1
instances\test_20.txt	28	0.02176857	2	3
instances\test_21.txt	46	0.07260251	4	7
instances\test_22.txt	52	0.087383032	11	15
instances\test_23.txt	35	0.046041012	9	11
instances\test_24.txt				
instances\test_25.txt	40	0.079577923	6	11
instances\test_26.txt	42	0.100782633	5	9
instances\test_27.txt	43	0.040364027	3	5
instances\test_28.txt	41	0.059547424	5	9
instances\test_29.txt	52	0.340853453	126	151
instances\test_3.txt	28	0.007999897	1	1
instances\test_30.txt	43	0.73623848	188	277
instances\test_31.txt				
instances\test_32.txt	31	0.020200491	7	11
instances\test_33.txt	31	0.088281155	54	59
instances\test_34.txt	33	0.026903152	4	7
instances\test_35.txt	30	0.040200233	4	7
instances\test_36.txt	23	0.012722015	2	3
instances\test_37.txt				
instances\test_38.txt	28	0.007038593	1	1
instances\test_39.txt				
instances\test_4.txt	32	0.129355431	18	35
instances\test_40.txt	24	0.014511347	2	3
instances\test_41.txt	45	2.228090763	558	751
instances\test_42.txt	57	1.221785069	271	409
instances\test_43.txt	46	0.151870966	69	73
instances\test_44.txt				
instances\test_45.txt				
instances\test_46.txt	57	0.810809135	11	21
instances\test_47.txt				
instances\test_48.txt	36	0.03724575	4	7
instances\test_49.txt	42	0.113820553	7	13
instances\test_5.txt				
instances\test_50.txt				
instances\test_6.txt	24	0.031957626	3	5
instances\test_7.txt	34	0.017089844	2	3
instances\test_8.txt	41	0.033881187	13	13
instances\test_9.txt	24	0.008224249	1	1

Graphs:

Comparison of CBS, IDCBS, IDCBS+LPA* runtime



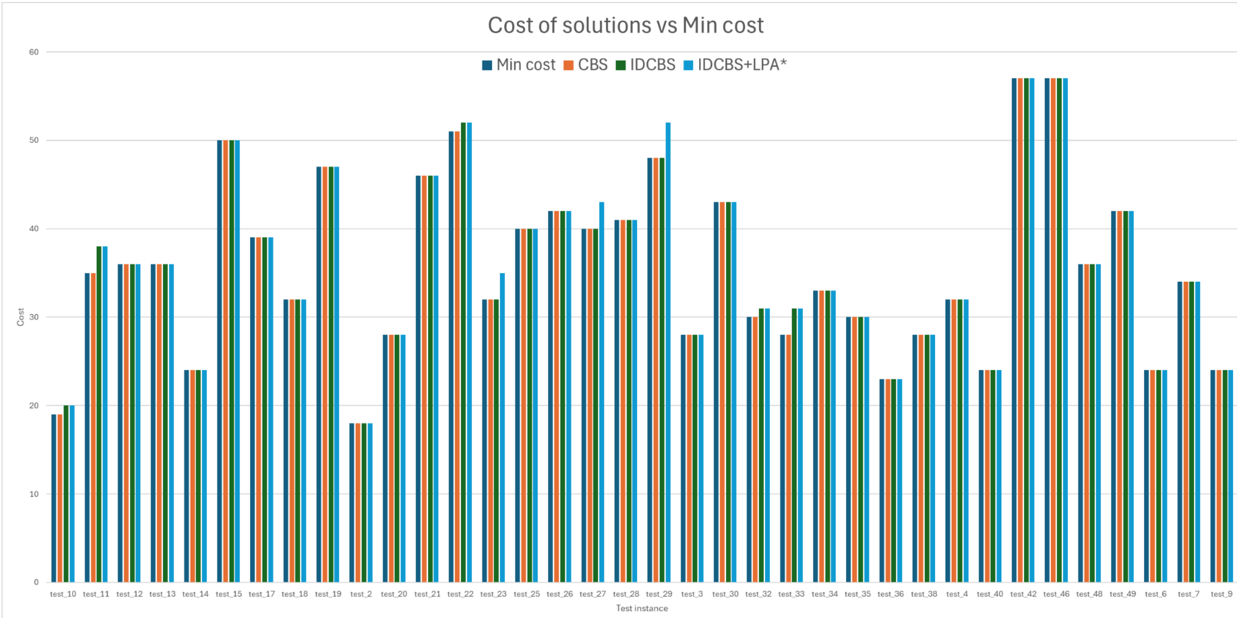
The graph above shows the runtime for each test instance that CBS, IDCBS and IDCBS with LPA* were able to complete, and we can compare the runtimes for each. Notably CBS is the faster algorithm, and IDCBS with LPA* is the slower of the three.



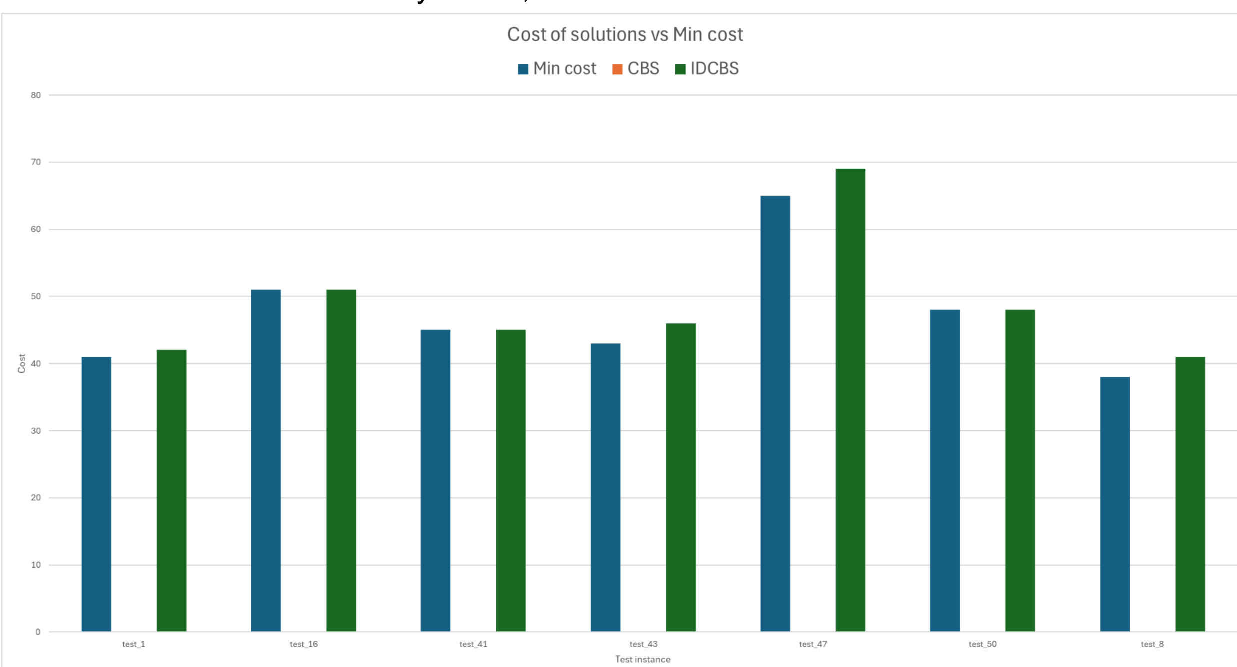
The graph above shows the runtimes of the IDCBS algorithm for all the test instances where CBS was unable to solve. All the solutions are computed within 1.5 seconds, and are fairly quick.

Not shown is the test instance 47, where IDCBS was able to solve, in 372.82 seconds.(Not shown because graph would be skewed terribly)

Comparison of CBS, IDCBS, IDCBS+LPA* optimality (sum cost vs ideal)

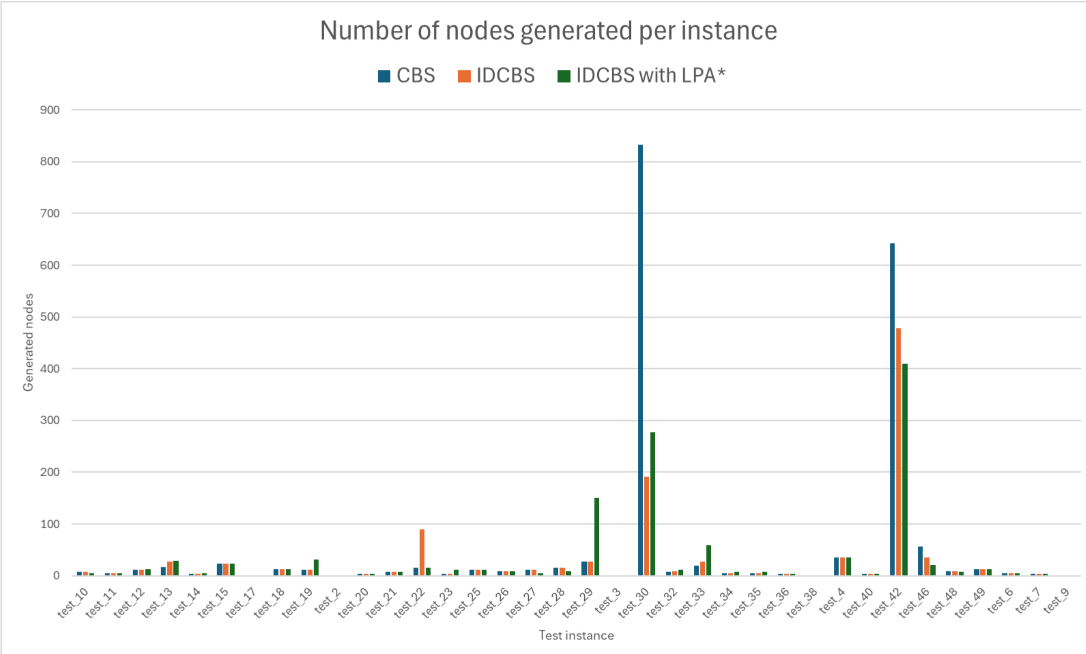


The graph above illustrates the solutions and optimality of solutions generated by our three algorithms CBS, IDCBS, and IDCBS with LPA*. Not included are instances where CBS was unable to find a solution. In the following graph, we will illustrate the solutions that CBS could not find a solution due to memory issues, and IDCBS was able to:

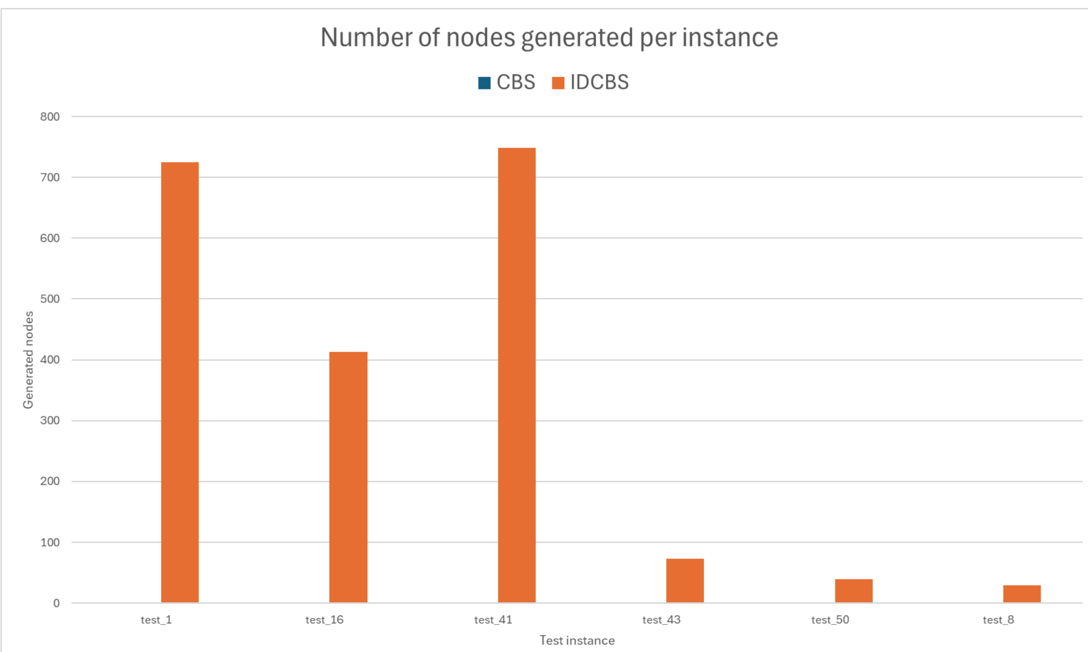


This shows that IDCBS was able to solve the instances where CBS was limited due to memory usage. The quality of solutions derived by IDCBS are also fairly optimal, sometimes a little more than min cost, but not too much.

Comparison of CBS, IDCBS, IDCBS+LPA* memory usage (generated nodes)



Shown in the above graph, is the comparison of generated nodes per each algorithm. Notably most solutions require very similar memory usage across the 3 algorithms. However on occasion the amount of memory usage seems to be unreasonably large(test_30).



Again, shown here is the amount of nodes generated by the IDCBS algorithm, where the CBS algorithm was unable to find a solution. Interestingly, the amount of generated nodes needed are quite large for most of these. Also, we have not included test_47 again, as it had generated a whopping 127845 nodes. (It would skew the graph beyond understanding)

Conclusions

Original questions from Methodology section:

How does IDCBS perform compared to CBS in terms of runtime and memory usage across multiple test instances?

- While IDCBS performed slower than CBS in terms of runtime, it showed a similar amount of memory usage to CBS.

Can IDCBS solve larger instances that are infeasible for CBS due to memory constraints?

- Yes

Does IDCBS maintain quality and optimality from the CBS algorithm?

- Yes

Are there specific instances where IDCBS outperforms CBS, or vice versa?

- CBS outperforms IDCBS when the state space of the search is minimal whereas if the state space of the search is sufficiently large, IDCBS outperforms CBS.

In conclusion, our data shows a key trade-off between runtime efficiency and problem-solving capacity in CBS and IDCBS. While CBS consistently achieves a faster runtime for solvable instances, its reliance on best-first search limits its ability to handle larger state spaces due to memory constraints. On the other hand, IDCBS is able to overcome this limitation by incorporating iterative depth-limiting, allowing it to incrementally explore the CT without exhausting memory resources at the cost of an acceptable amount of increased runtime. This enables IDCBS to solve problems with significantly larger state spaces that are beyond CBS's capabilities while preserving the quality and optimality of the solutions. These findings demonstrate that IDCBS offers a robust alternative to CBS by balancing scalability with computational efficiency.

Bibliography

Andreychuk, A., Yakovlev, K., Boyarski, E., & Stern, R. (2021). Improving Continuous-Time Conflict Based Search. Proceedings of the International Symposium on Combinatorial Search, 12(1), 145–146. https://doi.org/10.1609/socs.v12i1.18564

Jiang, H., Huang, R. IDCBS and DFBnB (sample report paper provided in coursys page)

Boyarski, E., Felner, A., Harabor, D., Stuckey, P., Cohen, L., Li, J. & Koenig, S. Iterative-Deepening Conflict-Based Search (paper provided in coursys)

Andreychuk, A., Yakovlev, K., Boyarski, E., & Stern, R. (2021). Improving Continuous-Time Conflict Based Search. Proceedings of the International Symposium on Combinatorial Search, 12(1), 145–146. https://doi.org/10.1609/socs.v12i1.18564