Solving MAPF with Bi-Directional A*star

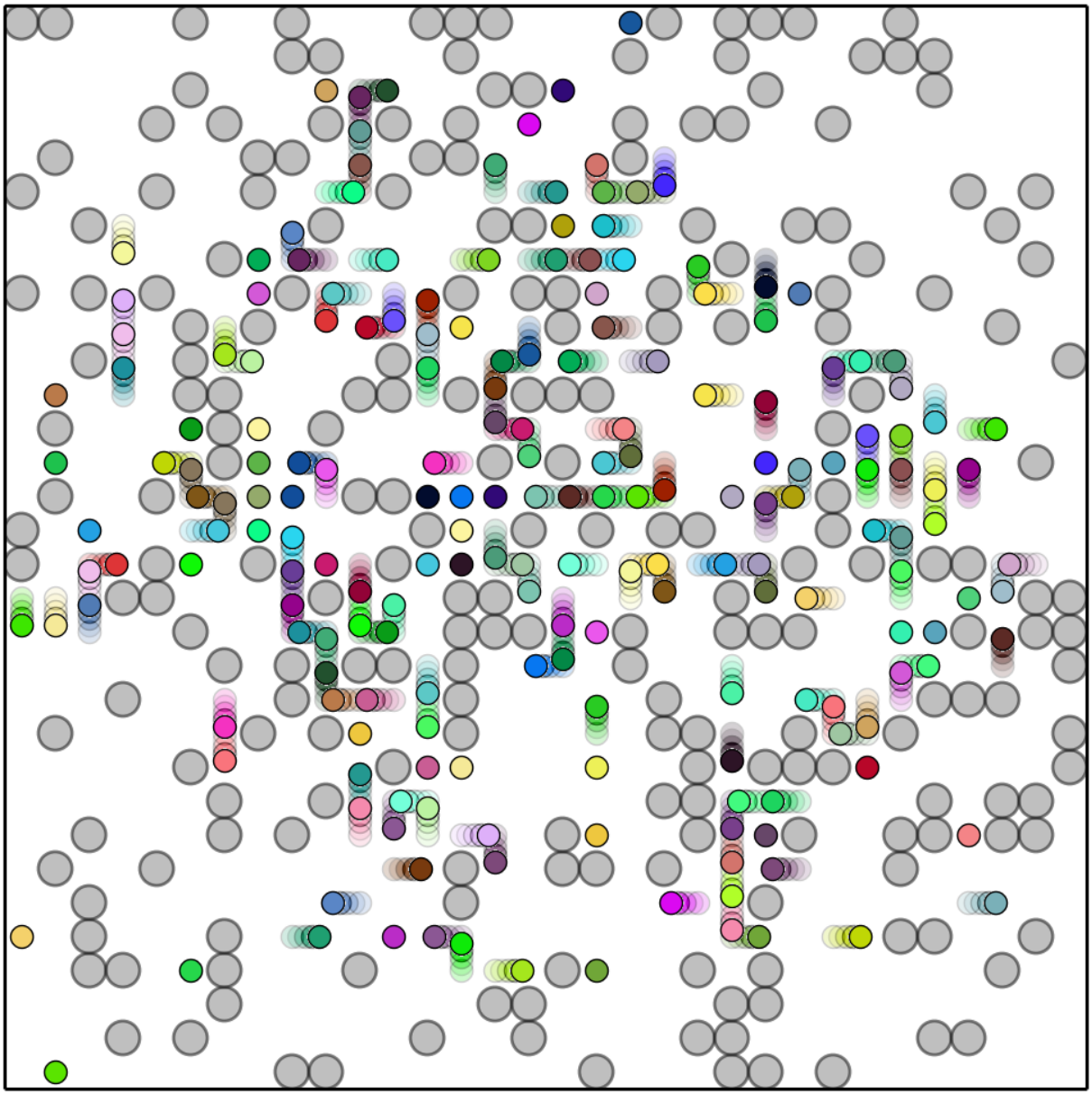
Personal project as part of “Collaborating Machines: Theories and Applications” course

Submitted by:

Asaf Hecht

201446002

Date: 17.07.2017



The photo was taken from: http://www.andrew.cmu.edu/user/gswagner/workshop/200_bots080_10_inf.png.

Contents

1. Project goals

1. Research and evaluate the approach of solving Multi Agents Path Finding (MAPF) problem with Bi-Directional Algorithm.
2. Do real cases experiment and analysis.
3. Compare the results and the performances when using the new suggested BDA*star approach.

2. Bi-Directional A*star algorithm

For my project, we chose to research Bi-Directional version of the well-known optimal A*star algorithm.

A* search algorithm is an extension of Dijkstra's algorithm that tries to reduce the total number of states explored by incorporating a heuristic estimation of the cost to get to the goal from a given state.

In Bi-Directional Search algorithms, we search the solution in two directions - one from the initial state forward and the second from the goal state backwards. The search checks if the two directions where met and then stops. Once the search is over, the path from the initial state is then concatenated with the inverse of the path from the goal state to form the complete solution path.

A general discussion on Bi-Directional search:
Bidirectional search still guarantees optimal solutions. Assuring that the comparisons for identifying a common state between the two searches direction can be done in constant time per node. The time complexity of Bidirectional Search is O(b^d/2) since each search need only proceed to half of the solution path. Since at least one of the searches must be breadth-first in order to find a common state, the space complexity of bidirectional search is also O(b^d/2). As a result, it is space bound in practice.
Advantages of Bi-Directional Search:

- In many cases the faster speed is the an advantage of this kind of Bi-Directional algorithms.
  Sum of the time taken by two searches (forward and backward) is much less than the O(bd) complexity. In MAPF this speed consideration is important.
- The second main advantage is that it requires less memory - In our MAPF solution world it is very crucial advantage. In the experiments I did as part of this project many times the memory space that were needed is many GB (15+) even if we are dealing with only 10 agents.

Disadvantages of Bi-Directional Search:

- Implementation of bidirectional search algorithm is difficult because additional logic must be included to decide which search tree to extend at each step.

The countermeasure - I will use A*star algorithm to the discovered middle state I calculate it using my new approach that I will describe in the next section.

- We should know the goal state in advance.
  The countermeasure - in MAPF we know the goal state of the search.
- The algorithm must be very efficient in his decision if there is an intersection of the two search trees.
  My countermeasure - I will use a pre-calculated middle state (will be describe in the next section).
- It is not always possible to search backward through possible states.
  The countermeasure - in MAPF it's possible, moreover in my approach after discovering the middle state you don't really need to run backward – you can also do another forward search from the middle state to the end goal state (although you won't call this a classic Bi-Directional method).

Because of the above advantages I decided to do my personal project in the course on this Bi-Directional approach. I thought it could be very interesting to research:

- If it will be available in practice to solve MAPF problems with Bi-Directional search.
- If it will be better – from speed and memory performances.

And indeed I glad to say there were some interesting and good results that I will describe in the next following sections.

3. Finding the middle positions of the solution

Usually the approach is to run both of the directions in the same time and check if we got encounter of the two searches - if there was an encounter it means we got the final full search solution.
However, I think there could be other way to perform the search, if we got the middle positions of the search - then we can run the Bi-Directional search from the start until the middle state, and the second from the goal state (end positions) backwards to the middle state.
Sure, for some domain problems it will be tough to find this middle stage location - but as part of my project I thought on a **really nice approach to find these middle positions in a MAPF problems:**
My approach is to run each single agent from our MAPF problem alone with A*star algorithm. This stage is very fast because we calculate the path to every agent separately. Now, after having each agent best path to his goal - I am taking the middle position of every individual path of every separated agent. Therefore, I got the middle positions of each agent in his optimal solution. What's left to do is to combine those middle positions together and we get the middle state that we will run our Bi-Directional algorithm with it.
In some rarely case if there are more than one agent in the same middle position, I can choose the nearest location for one of them (because in the combine solution they could not be in the same time in the same place).

I think one of my most important discovery in my project is - in the same way I discovered the middle position (using my method mentioned above) it is possible to divide the MAPF problem to more than only 2 parts (for Bi-Directional search). It possible to divide the problem to 3,4,5 or more. Even to divide the problem even to 100 or more smaller problems (depends on how big is the source MAPF problem).
We can divide the problem to much more smaller problems – and it will give us the ability to solve the problem in better speed (because every problem need to be solved is smaller) and with much less memory consumption (because after we divide the problem and get new start and end states - we don't need to run the solver in the same time, one after the other will be good enough).
This new approach to solve MAPF could be called Divide and Conquer A*star = DCA*star.
Next in the following section I will do real experiments with my Bi-Directional A*star = **BDA*star algorithm**, and examine the results.

4. Experiments of solving MAPF using BDA*star + results

I used the course MAPF code infrastructure to experiment and examine my new approach of Bi-Directional A*star.
I executed the tests on 3 main problems maps. In this section I will describe the results and the performance I had observed.
General explanations:

- "Synthetic Bi-Directional Astar": is when I took the middle positions from the middle of the full multiagent A*star solution. It represents the best middle positions (and optimal) of the problem but to get it we need to really solve the full problem before. Meaning the algorithm solved the problem entirely by A*star and then the middle state was token by me to be the middle state for running my Bi-Directional A*star.
- "Automated Bi-Directional Astar": is when I automated the process of calculation of the middle state. As describe in the above section, I firstly calculated each agent separately then took the middle of each optimal solution path and combine them together to run my Bi-Directional algorithm on this middle state of all the agents.
- "Total cost" - the total cost of the problem – we can see that in my experiments the solutions with BDA*star is still optimal and don't disturb the cost of the MAPF solution.
- "Total time" - the total time in milliseconds it took to solve the problem using the specific method (accordingly to the method row in the results table).
- "Expanded nodes" - the total number of the expanded nodes the algorithm was needed to calculate to solve the problem.
- "Generated nodes" - the total number of the generated nodes the algorithm was needed to calculate to solve the problem.
- "Improvement in percentage" - how much does the new approach improve the results. The improvement rate was calculated from the regular full A*star to the automated BDA*star method.
- "Memory consumption improvement" - in all of my test the RAM memory that was needed to the algorithm to solve the problem was reduce by almost 50%!

My experiment environment - I used my own personal laptop:

Visual Studio Enterprise 2015 - version 14.0.25431.01.
CPU - intel Core i7-6700HQ CPU 2.60GHZ
Total available RAM memory - 32GB.
While running the experiments my computer did other tasks.

In general, there where some cases that the **original** A*star was dramatically perform slowly. I checked those cases and it's weird and not so clear, for example if I would change only one agent location by moving it 1 spot aside it will run fast and normally again. I think it might be a code implementation error, that occurred in some rarely cases.

4.1 Dragon map test

4.1.1 The map of the problem:

The original name of that dragon maps is “den502d.map”.
In my experiments and code, I called it “Instance-252-66-X-0” where X is the number of agents in that specific MAPF problem file.
This is a big map - its size is 251 X 211.
Those actual experiment files are attached in the “problem instances” folder.

4.1.2 The map results and evaluation:

- Solving the map with **2 agents:**

	Total cost	Total time	Total open nodes	Generated nodes
Full regular Astar:	508	14	10351	5050
Synthetic Bi-Directional Astar:	508	9	7777	4935
Automated Bi-Directional Astar:	508	9	7777	4937
First half forward search:	254	5	4011	4011
Second half reverse search:	254	4	3766	3766
Improvement in percentage:		**35.71%**	24.87%	2.24

Time improvement = 35.71%
Memory improvement of almost 50%

- Solving the map with **3 agents:**

	Total cost	Total time	Total open nodes	Generated nodes
Full regular Astar:	701	323	43840	27362
Synthetic Bi-Directional Astar:	701	279	39071	26774
Automated Bi-Directional Astar:	701	263	38927	26023
First half forward search:	351	115	20661	14092
Second half reverse search:	350	148	18266	11931
Improvement in percentage:		**10.46%**	11.21%	4.89

Time improvement = 10.46%
Memory improvement of almost 50%

- Solving the map with **4 agents:**

	Total cost	Total time	Total open nodes	Generated nodes
Full regular Astar:	924	1427	205210	140969
Synthetic Bi-Directional Astar:	924	1251	189796	133874
Automated Bi-Directional Astar:	924	1193	192747	134757
First half forward search:	463	522	103911	73716
Second half reverse search:	461	671	88836	61041
Improvement in percentage:		**16.4%**	6.07%	4.41

Time improvement = 16.4%
Memory improvement of almost 50%

- Solving the map with **5 agents:**

	Total cost	Total time (ms)	Time in Mins	Total open nodes	Generated nodes
Full regular Astar:	1088	115331	1.922183333	998765	704169
Synthetic Bi-Directional Astar:	1088	83647	1.394116667	931646	668622
Automated Bi-Directional Astar:	1088	81216	1.3536	954222	674192
First half forward search:	545	54681	0.91135	514536	366917
Second half reverse search:	543	26355	0.43925	439686	307275
Improvement in percentage:		**29.58%**		4.46%	4.26%

Time improvement = 29.58%
Memory improvement of almost 50%

- Solving the map with **6 agents:**

	Total cost	Total time (ms)	Time in Mins	Total open nodes	Generated nodes
Full regular Astar:	1221	990444	16.5	4959430	3529906
Bi-Directional Astar:	1221	631849	10.5	4637146	3352171
Automated Bi-Directional Astar:	1221	553288	9.22	4755622	3378056
First half forward search:	612	238913	3.98	2563286	1836817
Second half reverse search:	609	314375	5.24	2192336	1541239
Improvement in percentage:		**44.13%**		4.11%	4.3%

Time improvement = 44.13%
Memory improvement of almost 50%

As an example - I attached the full output of the solving process of this problem - it's in the "Problem Instances" folder, called: "Example for full output - on problem instance-252-66-6-0.txt". It provides the full statistics results includes the full path solutions for all the agents.

- Solving the map with **7 agents and more:**
  When solving with more than 7 agents it took lots of time and memory to solve it with the regular full A*star algorithm, therefore I stopped with 6.

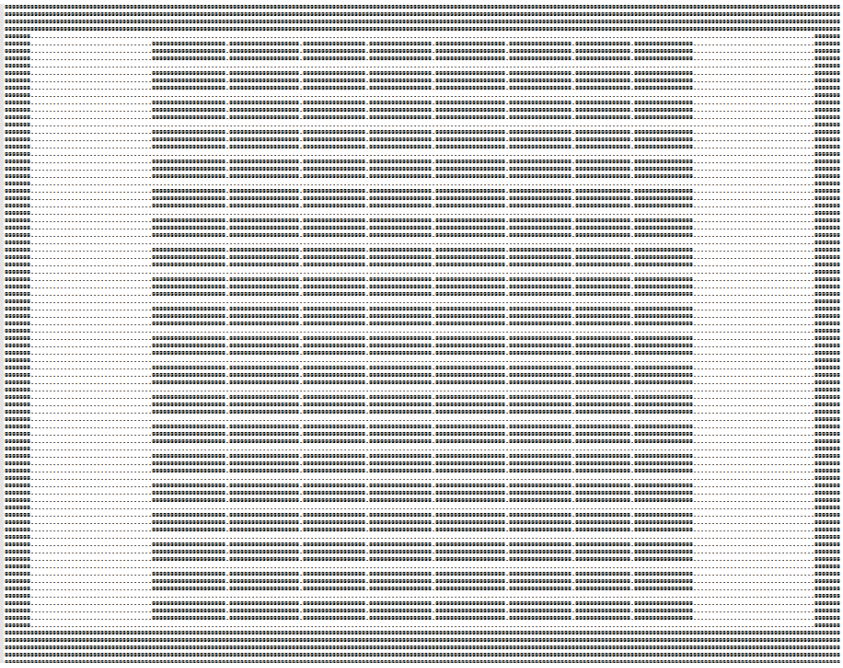
4.1.3 The current observations:

We can see that in all the above instances the new method of automated BDA*star succeeded to solve the problem in:

- An optimal cost.
- Less time - an average of 28% quicker.
- Less total number of open nodes – at least 4% less.
- It needed mush less RAM memory – almost 50% less in all the instances.

4.2 KIVA like map test

4.2.1 The map of the problem:



This map was created by me and designed to be like the “KIVA” problem we learn in class.
The experiment problem files are also in the “problem instances folder” and are named: “Instance-252-77-X-0” where X is the corresponding number of agents in that specific problem file.
It’s a big map, its size is 90 X 227.

4.2.2 The map results and evaluation:

- Solving the map with **5 agents:**

	Total cost	Total time (ms)	Total open nodes	Generated nodes
Full regular Astar:	851	4841	307482	204730
Automated Bi-Directional Astar:	851	4387	293439	200393
First half forward search	427	2984	159284	113208
Second half reverse search	424	1403	134155	87185
Improvement in percentage:		9.38%	4.56%	2.11%

Time improvement = 9.38%
Memory improvement of almost 50%

- Solving the map with **6 agents:**

	Total cost	Total time (ms)	Total open nodes	Generated nodes
Full regular Astar:	1066	48310	1341870	954477
Automated Bi-Directional Astar:	1066	44041	1241616	903434
First half forward search	535	33199	702270	520332
Second half reverse search	531	9082	539346	383102
Improvement in percentage:		8.37%	7.47%	5.35%

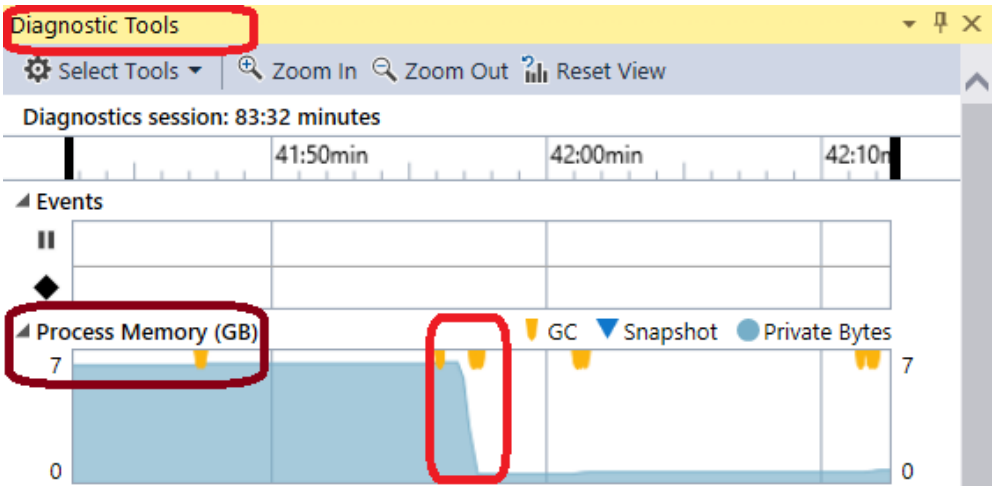
Time improvement = 8.37%
Memory improvement of almost 50%

- Solving the map with **7 agents:**

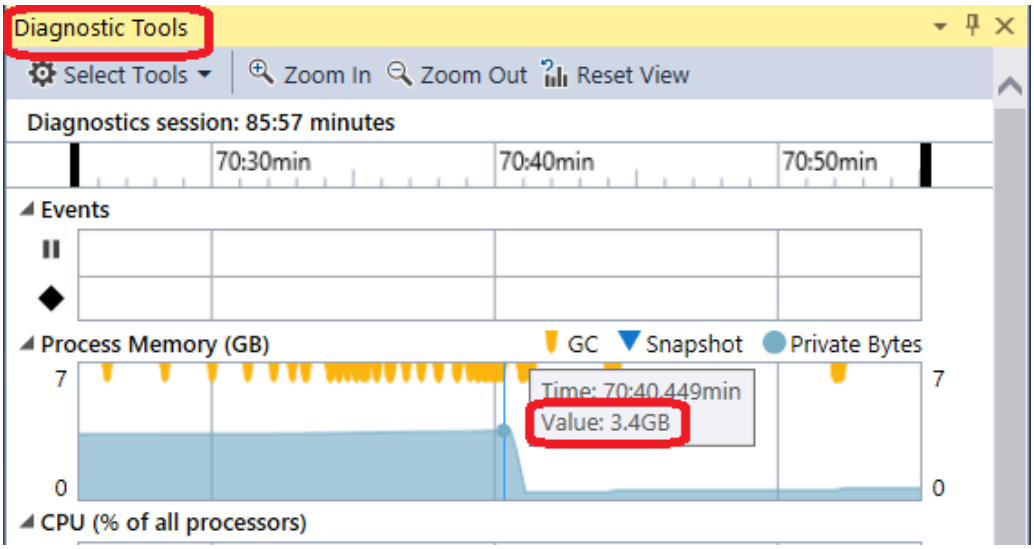
	Total cost	Total time (ms)	Time in Mins	Total open nodes	Generated nodes
Full regular Astar:	1263	2209624	36.82706667	5885060	4326514
Automated Bi-Directional Astar:	1263	2196411	36.60685	5299901	3965921
First half forward search	634	1652926	27.54876667	3119930	2370663
Second half reverse search	629	543485	9.058083333	2179971	1595258
Improvement in percentage:		0.56%		9.94%	8.33%

Time improvement = 0.56%
Total open nodes improvement = 9.94%
Memory improvement of almost 50%

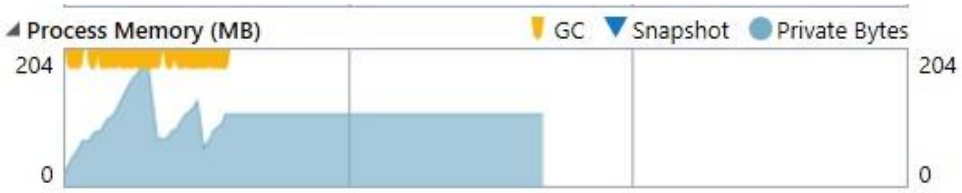
Here is a drill down to the improvement of the memory consumption:
I took it from the Visual Studio's Diagnostic Tool while the problem was running and solving this MAPF with 7 agents.
This is the first full A*star algorithm when it finished. We can see it needed almost 7GB of RAM memory to succeed to solve this MAPF problem.



Now, in the next print screen we can see the situation when the first half of the BDA*star was finished.
It was needed only 3.4 GB. After this time spot, the algorithm continues to the second reverse half that also took 3.4GB - but not in the same time!
It's a big advantage when we are lack of memory or wants to run bigger problem maps with more agents.



Here is another real case schema of my new Bi-Directional A*star memory saving in action. We can observe when the first full A*star solver ended, then the first half forward solver ended, and finally the second half backwards solver ended (when the program is totally finished the memory consumption is constant):

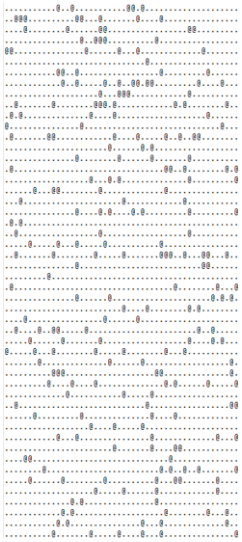


4.2.3 The current observations:

We can see also in this KIVA like problem map the new BDA*star was performing better. It solves the problem in the optimal cost in less time and used less RAM memory, with less open nodes in the solving process.
In compare to the Dragon map the time improvement is smaller. I think it could be because the Kiva domain map has narrow corridors in contrast to the Dragon map that is much more open and therefore in the dragon the A*star has more options it needs to consider in its solving process.

4.3 Random map test

4.3.1 The map of the problem:



This is a random map that was created automatically by the MAPF code randomly.
It's a medium size map - size 50 X 50.
This instance is also in the "problem instances" folder named "Instance-50-10-6-0".

4.3.2 The map results and evaluation:

- Solving the map with **6 agents:**

	Total cost	Total time (ms)	Total open nodes	Generated nodes
Full regular Astar:	174	13140	544547	407809
Automated Bi-Directional Astar:	174	9797	512330	387589
First half forward search	89	4572	257962	194152
Second half reverse search	85	5225	254368	193437
Improvement in percentage:		25.44%	5.91%	4.95%

Time improvement = 25.44%
Memory improvement of almost 50%

4.3.3 The current observations:

We can see a better performance with the automated Bi-Directional A*star.
I will add that later on I did more tests on other random maps and the performance improvements were less than in this specific experiment (but there was still an improvement).

5. Great possible future work

In this project, I research the behavior of an automated Bi-Directional A*star solution.
I observed in the experiments that this new method is indeed improving the results and the performance of the solution.
As I described in the beginning it will be very interesting to research the same division method - of running each agent separately and then combined their middle state together. In my project I used it to split the problem to two halves.
The thing is that it possible to divide the same problem to three, four or even more. Then solve each part in its own. Each problem part could be solved in a forward A*star (it is not must to ran in the reverse order backwards).
Therefore, the full solution of a complicated MAPF problem can be calculated in less time and less memory consumption. After the solving of each smaller problem the full solution will be driven from their combination.

6. Summary and Conclusions

My project submission includes:
- This final report document.
- The “problem instances” folder with all the problem maps that were mention above.
- The full project code infrastructure - based on the course MAPF infrastructure, I wrote modification notes and explanations where I added my own code.

In this course project I researched a new algorithm to solve MAPF problems - an automated Bi-Directional A*star algorithm (BDA*star).
I thought on a new method to find the middle state of a given MAPF problem.
This method includes solving each agent separately with A*star and then choose the middle of each individual path. After this, a combination of all of those middle positions that were calculated very fast (because they run on a single agent solver) will give us the full middle state of our MAPF problem.
After discovering the middle state of the given problem - I executed a Bi-Directional search from the start forward to that middle and from the end goal state backward.
I described the tests I performed. In those tests my new solver performed better than the regular A*star - less time and used less memory. Moreover, it still gave the same optimal path cost.
My method of finding the middle state in my project can be used in more ways. I can take a given problem and split it to more than just 2 parts. It possible to divide the given MAPF problem to much more smaller problems and then solve them separately. A suggestion of mine for such an algorithm is Divide and Conquer A*star (DCA*star).

In a personal perspective - I enjoyed doing this project and in general the course was very interesting.
I hope my project includes some new thoughts on how to deal with MAPF problems.

Thanks,
Asaf Hecht