Conflict-Based Search for Multi-Agent Pathfinding

Gabi Guillermo
Boston University
gabe441@bu.edu

Vladislav Rada
Boston University
vladrada@bu.edu

Christopher Relyea
Boston University
crelyea@bu.edu

Figure 1: The CSIRO Data61 team deploys several exploration robots as part of the 2020 DARPA Subterranean Challenge.

ABSTRACT

Conflict-Based Search (CBS) is a popular algorithm used to solve the Multi-Agent Pathfinding (MAPF) problem. While it theoretically guarantees optimality in planning paths for multiple agents, CBS is known to scale poorly when increasing the number of agents to magnitudes inline with real-world robot fleets. In this paper, we evaluate the scalability of the algorithm by measuring its runtime and memory usage across several maps commonly used for MAPF benchmarking. We analyze the performance of CBS in environments that are both city-like and similar to natural terrain. Our experiment makes use of a commonly-used C++ implementation of CBS run on a system with capabilities similar to those of a cloud robotics node. Our first experiment confirms previous claims of exponential CBS runtime growth with increasing agents, while a second analyzes memory use on a deeper level using Valgrind and Cachegrind, uncovering trends in memory behavior in different environments and agent densities. The results we have collected confirm suspicions of the real-world limitations of CBS and present possibilities for further optimizations which have important implications for the deployment of CBS-based systems in many applications.

1. INTRODUCTION

1.1 Motivations and Existing Work

The exploration of environments is a rapidly-developing topic of interest in robotics due to its wide variety of applications. Such technology can be used in search and rescue missions, foraging, mapping, and other applications. Multi-agent pathfinding, motion planning for "fleets" containing more than one robot agent working in cooperation, has the potential to significantly lower exploration times through parallelism, which could be critical in time-sensitive applications. In 2015, Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant proposed two novel algorithms to the multi-agent pathfinding problem (MAPF), where the goal was to 'find paths for all agents while avoiding collisions' [6]. The main algorithm presented by the group was Conflict-Based Search (CBS), which, unlike previous solutions, does not use a single 'joint agent' model. Instead, the algorithm develops a set of constraints, finds paths which are consistent with the constraints, then checks that the paths do not have conflicts. If the paths do intersect, a new constraint is added to the list. An extension of this algorithm, Meta-Agent CBS (MA-CBS), was also presented in the same paper. This version of the algorithm is more generalized and improves performance for the algorithms where CBS fails to perform as expected, particularly in cases with large swarms.

The optimal solution for the Multi-Agent Pathfinding problem is NP-Hard, and since the original paper's release in 2015 Conflict-Based Search has received considerable attention from researchers due to its theoretical optimality and completeness. A significant number of papers have been published using conflict-based approaches, such as "Safer-CBS" proposed by Yicong Liu et al. (2024). Safer-CBS proposes a risk-constrained algorithm for Connected and Automated Vehicles (CAVs) that "demonstrates that Safer–CBS using the proposed risk constraint optimization achieves the highest success rate, particularly with an increasing number of vehicles" [5].

In reviewing the capabilities of CBS (including MA-CBS) it becomes clear that the algorithm has limitations. We refer to two more recent papers (Gordon et. al [3], Boyarski et. al [1]) that revisit the algorithm and expose its flaws. These papers propose some more state-of-the-art implementations of the algorithm, including Boyarski's "Improved-CBS" (I-CBS). The primary limitation is that CBS is a slow and inefficient approach when deployed to a high number of agents, as admitted by the original authors of the algorithm. When the number of robots using this algorithm increases, so does the number of collisions the algorithm must consider. Since each collision event in the generated graphs necessitates a slew of mathematically-intensive calculations, CBS is not viable for real-time path-finding in large swarms.

Figure 2: Researcher Robin Murphy studies robot fleet deployment during disaster scenarios.

Sharon et. al notes in the algorithm's original proposal that while the time complexity of CBS does not grow exponentially with respect to the number of agents, it does, more specifically, scale with respect to the number of collisions. Considering, for example, the use case of subterranean exploration. It may be the case that spaces are unusually confined or unpredictable, leading to more conflicts than would occur with the same number of agents in a more predictable, open environment like the outdoors or a structured warehouse. CBS is thus a limiting choice for cave or tunnel systems. Additionally, in the case of environments that may change during the process of path-finding, this algorithm may be a poor option for its inability to meet real-time deadlines at high swarm sizes. For instance, the highly volatile settings of disaster zones, which present both a difficult environment and a safety-critical time constraint in terms of finding survivors, are an important challenge for MAPF systems. Should its limitations be addressed, CBS could become a much more viable exploration option for swarm systems in many applications.

1.2 Real-World Deployment

Warehouse fleets are a common use case for robot swarms, often at a large scale. A good example is found in Amazon's fulfillment centers, where state-of-the-art robotics and computing technology are applied to reduce delivery times as much as possible. Robotics have apparently been implemented throughout every stage of the company's process (though they maintain that human workers continue to have a role in shipping operations).

Increasing the efficiency of high-scale fleet deployment is a significantly lucrative endeavor for such companies. Amazon opened a warehouse in Shreveport, Louisiana last year which has 10 times as many robotic piece of equipment in its pipeline as previous locations. According to financial reports, this investment in a highly-automated fulfillment process has led to a 25 percent reduction in the cost to fulfill orders. Analysts at Morgan Stanley estimate that further investments in similar warehouse operation practices will generate Amazon around $10 billion in annual savings by 2030 [2].

This trend underscores the necessity of efficient path-planning computing and careful deployment of multi-agent fleets. As the size of these robotic teams scales further with time, it becomes crucial for algorithms like Conflict-Based Search to be as efficient as possible. This makes our evaluations aligned with the growing demands of real-world robotics, with the commerce sector being only one example.

2. SIMULATION SETUP

2.1 Existing Implementations

To test CBS on our own system, we made use of a public Github repository that implements the algorithm in C++. The code is published in accompaniment with a paper written by Jiaoyang Li et al, which also takes a look at CBS in the context of multi-agent pathfinding [4]. As a part of the work put forward in this paper, the repository includes a number of modifications to the original CBS algorithm, which the authors intended as improvements to lower runtime and computational cost, including methods referred to as "disjoint splitting" and "mutex propagation." The C++ code is written with arguments to enable or disable these modifications. By specifying in the command line to disable every modification to the original CBS algorithm, we can use the work of Li et al. to run CBS as it was originally designed.

Due to the performance-critical nature of CBS applications, we decided to use a C++ implementation of Conflict-Based Search rather than Python alternatives. C++ offers advantages in the areas of computation speed and memory efficiency, making it a more accurate lens through which to view the limitations of CBS. Python, by comparison, comes with significant computational overhead which would have made conclusive results for CBS performance difficult to attain.

2.2 Simulations Dataset

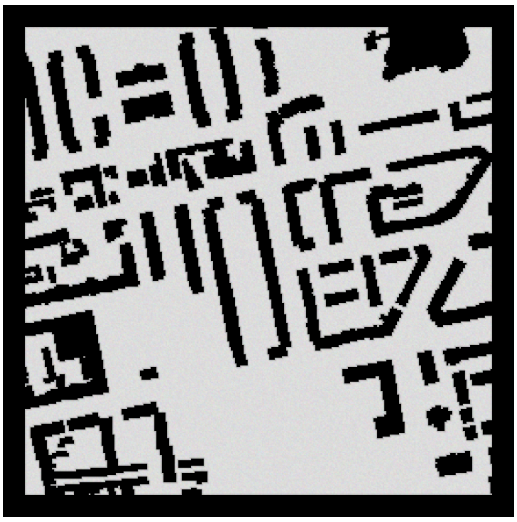

Figure 3: Berlin_1_256.map

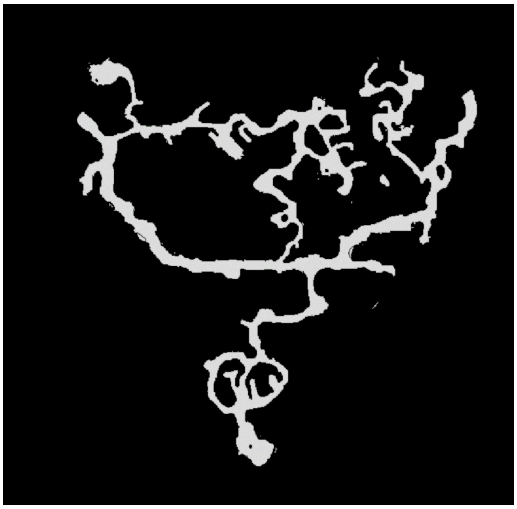

Figure 4: w_woundedcoast.map

17	Berlin_1_256.map	256	256	24	153	58	201	69.11269836
60	Berlin_1_256.map	256	256	51	230	11	23	241.12489166
46	Berlin_1_256.map	256	256	82	170	253	170	184.25483398
33	Berlin_1_256.map	256	256	93	77	69	195	134.32590179
67	Berlin_1_256.map	256	256	56	9	243	103	269.06601715
24	Berlin_1_256.map	256	256	141	38	226	56	97.76955261
26	Berlin_1_256.map	256	256	250	121	148	125	106.97056274
41	Berlin_1_256.map	256	256	178	15	37	71	164.19595947
0	Berlin_1_256.map	256	256	157	140	160	138	3.82842712
27	Berlin_1_256.map	256	256	80	142	84	36	110.97056274
30	Berlin_1_256.map	256	256	125	82	171	180	122.91168823
64	Berlin_1_256.map	256	256	13	43	232	27	256.53910522
39	Berlin_1_256.map	256	256	115	248	81	104	158.08326111
10	Berlin_1_256.map	256	256	17	151	51	167	40.62741699
24	Berlin_1_256.map	256	256	79	126	122	205	96.81118317
14	Berlin_1_256.map	256	256	126	184	79	206	56.11269836

Figure 5: Structure of a scenario file which places many possible agents on a map by specifying origin and destination.

In the paper authored by Sharon et al. which proposes the CBS algorithm for the first time, the authors include a data set with many sample maps, including a plot of obstacles and valid spaces for agent movement, and “scenario” files which specify the origin and destination coordinates for around one thousand agents, different for each map. The CBS code we used for this experiment is compatible with these maps and scenario files. When running the CBS code, the user can specify the file name of the map they’d like to use, the scenario file they’d like to pull the agent data from, and the number of agents to pull from the scenario. For a number of agents n specified in the command line, the first n agents in the appropriate scenario (starting from the top of the .scen file) will be placed on the designated map and simulated using CBS.

In the maps/scenarios dataset, each map comes with scenario files of type "even" or "random." According to the creators of the benchmark data, "even" scenario files entail agents laid out throughout a map in a manner that covers the environment with roughly uniform spacing between all agents, while "random" files do not take any particular care in placing agents on the map, assorting them randomly throughout the environment. After some preliminary testing to measure the difference in CBS behavior between the two types of scenario files, we determined that it would be best to use "even" scenario files for our experiments, since random files present the possibility of unrealistic agent placement. For example, there is nothing preventing a random scenario file from placing most of its agents on one corner of a map, or making many agents have absolute distances (straight line from origin to destination) that are very short, expecting the agent to move only a few units in one direction, while others may need to traverse from one end of the map to the other.

The map files provided in the sample dataset are useful for simulating real-world environments. For example, three maps are provided based on cities: Berlin (Figure 3a), Boston, and Paris. These environments model the streets of a dense city, and could be useful to analyze the effectiveness of deployment of a swarm of robots in such a setting. This could be an interesting and appropriate benchmark for testing a search and rescue swarm system deployable in such areas after a disaster scenario. Another map, w_woundedcoast (Figure 3b) which is used in our second experiment, appears to resemble some natural, winding formations one might associate with some connected bodies of water or the tunnels of a subterranean environment. This kind of map is a good indicator of CBS’ application to robotic swarm exploration, especially in environments such as caves which may be too dangerous for human navigation.

2.3 Code Contributions

To manage the prewritten C++ code and run it at scale, we wrote a customizable and comprehensive Python script to run many maps and scenario files in succession as determined by the user. Our code is used to iterate through every map designated for testing, and then execute each map with an increasing number of agents to measure changes in the performance of the CBS algorithm. This process will be described in more detail in the next section.

2.4 System

For our controlled experiment, we ran the CBS code on a desktop PC with an Intel i7-11700KF and 32 GB of RAM running Fedora Linux 41. This configuration is not identical to the embedded systems usually found in physical robotic systems, but the computational capabilities of our system are representative of those which might be used in cloud servers for edge robotics. Such centralized planning systems are often used for real-world mutli-agent systems, which tend to offload complex computations (like CBS) to external nodes with processing power comparable to ours. Thus, our configuration provides a reasonable imitation with which we may accurately evaluate the computational performance and scalability of Conflict-Based Search in realistic scenarios.

Figure 6: Package transportation robots in an Amazon warehouse.

3. EXPERIMENT

3.1 First Attempt

For our first experiment, we opted to use a single map, Berlin_1_256, and measure the effects on the performance of CBS that occur as a result of scaling the number of agents. Our goal was to verify the statements made in the original CBS paper regarding the exponential complexity of the algorithm with hopes of making more concrete statements on the runtime and memory usage of Conflict-Based Search.

The Berlin map defines a 256 by 256 unit environment of obstacle spaces or valid movement spaces. 25 usable "even" scenario files are provided for this map, each dictating the origin and target for 1010 unique agents.

We configured our Python script to try each of the 25 scenario files for Berlin in succession. For each one of those scenario files, we repeatedly applied CBS to an increasing number of agents, starting with one agent (pulled from the start of the scenario file as described in the previous section) and increasing on each run until a 60 second timeout is reached for that number of agents (deemed to be a realistic stopping point after which multi-agent path planning takes too much time).

Sharon et. al originally found that the runtime of CBS scales exponentially with the number of agents (or more specifically, the number of agent collisions) in the problem. This claim is confirmed in the results of our preliminary experiment, where CBS runtime shows explosive growth beginning after about 80 agents (Figure 6).

Understandably with a map size of 256 by 256 units, we begin to see timeouts roughly around the 127 agents mark. This is the point where the map, due to its size and/or density of obstacles, does not have enough available space to plot all of the agent paths without collisions.

Overall, runtime costs were lower than we anticipated. Some iterations of CBS in this initial experiment did not take longer than 1 second until there were over 100 agents drawn from that scenario file. After the one-second runtime was passed, in general, the tests soon began to fail, surpassing the 60 second timeout limit (as previously stated, this is most likely due to a lack of valid solutions).

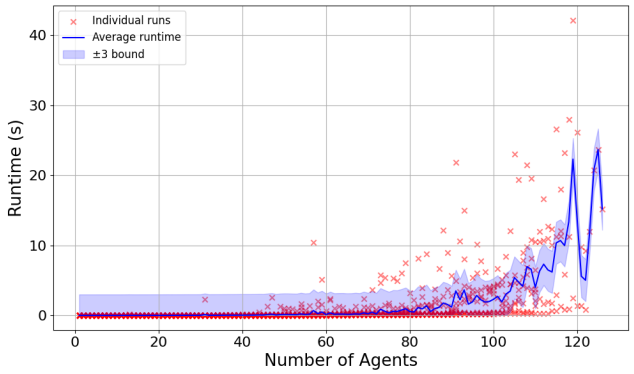

Figure 7: CBS runtime in seconds vs. number of agents

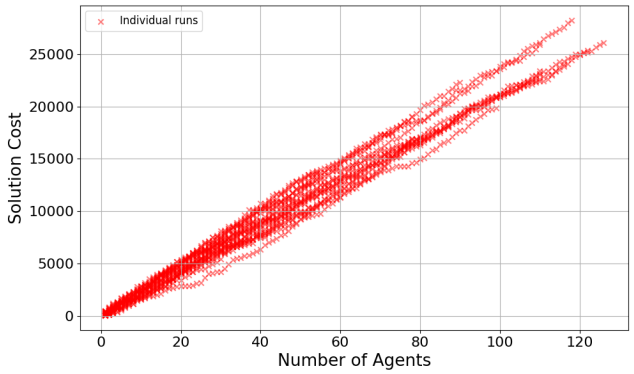

Figure 8: Total solution cost vs. number of agents

Solution cost, a distance metric associated with path planning problems, describes the total length of all paths of every agent. We can analyze this metric as our best estimate of how energy consumption may be affected by the performance of CBS, since with the more distance all of the agents will need to cover, the more actuation energy required to move robots through their environment to fulfill the planned paths. Solution cost, therefore, provides a useful metric for CBS' application to search-and-rescue missions where it is imperative for swarms of robots to cover as much ground as possible before depleting their actuation energy supplies, helping as many survivors as possible. The CBS library we have chosen to use for our experiments reports solution cost at the conclusion of every run. For our preliminary run, a graph of solution costs is presented in Figure 7.

Finally, we have collected basic data on the memory usage trends of agent-scaling CBS by recording measurements of peak RAM consumption during the execution of the CBS code. As seen in Figure 8, RAM usage starts very low at less than 1MB up until around 20 agents, then suddenly increases. Using the memory tool Valgrind and its heap profiler massif, we investigated this memory pattern further (Figures 9 and 10). We found that the average heap size per snapshot remains relatively consistent, about 0.83 MB, while the peak heap per snapshot is about 2.12 MB on average.

3.2 Second Attempt

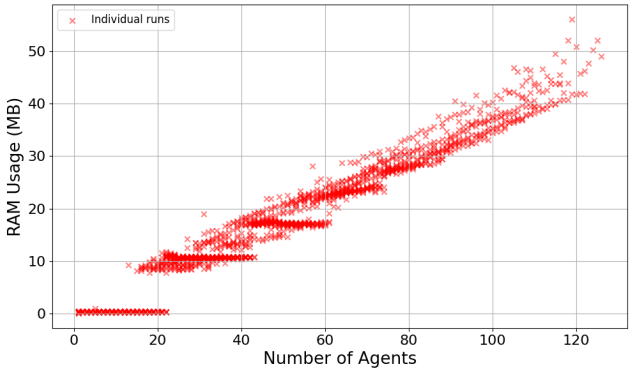

Figure 9: RAM usage vs. number of agents

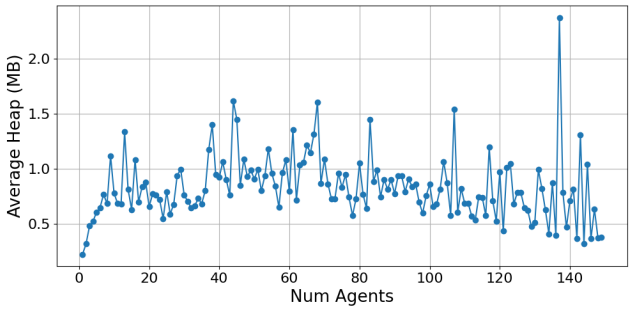

Figure 10: Average heap usage vs. number of agents

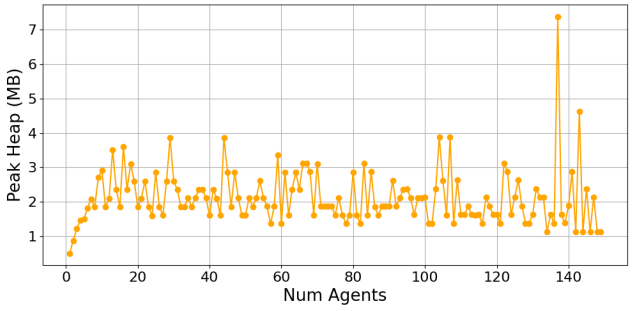

Figure 11: Peak heap usage vs. number of agents

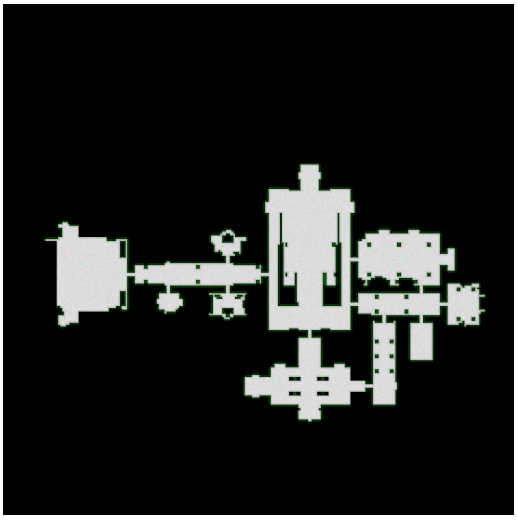

Figure 12: lt_gallowstemplar_n.map

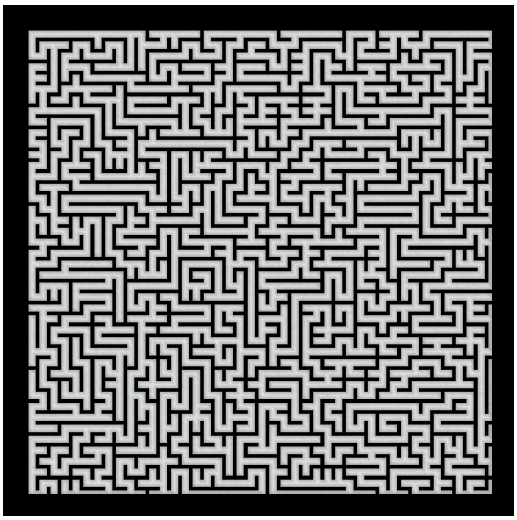

Figure 13: maze-128-128-2.map

For our second experiment, we used a workstation with an Intel i9-14900K and 64 GB of RAM running Fedora Linux 42. As this computer was much faster than the one from our previous experiment, the CBS runtime was overall lower, but our data still reflects the exponential growth displayed by the algorithm.

The same CBS C++ library was used as the one used in the previous experiment. However, more in-depth experiments were run, and additional tools from Valgrind were used to gather information about cache misses that occurred during the program's execution. Valgrind's Cachegrind and Massif tools were used to gather data here. For the cache simulation done via Cachegrind, the parameters used were the following: instruction level 1 cache with a size of 32 KB, 8-way associativity, and a line size of 64 bytes; the instruction level 1 data cache with a size of 48 KB, 12-way associativity, and a line size of 64 bytes; and a last-level cache with a size of 2 MB, 16-way associativity, and a line size of 64 bytes.

3 different maps were used here. Berlin_1_256 was once

again analyzed, in addition to It_gallowstemplar_n (Figure 11a) and w_woundedcoast (Figure 3b). These maps were chosen as they are all very different from each other. Berlin is not very dense, structured so that it has very large open areas between the buildings. Gallows Templar is structured to be more video game-like, having the structure of a few rooms connected to each other with thin hallways. This may closely mirror our application of underground search-and-rescue missions. The Wounded Coast map, as mentioned earlier, is structured with a few long, wide tunnels, which may also reflect our intended application. Each map came with pre-made scenario files which randomly placed the agents throughout the map, and 5 scenario files were selected to be used as examples for the experiments.

We were initially going to use a fourth map, a maze, but the corridors of the maze were too thin (2 tiles wide) to allow us to increase the number of agents (maze-128-128-2.map, Figure 11b). Beginning at 5 agents, the map would begin to time out. As such, this map was removed from the dataset, as it did not provide useful information about the exponential growth of the program. However, it serves as a reminder that the amount of agents generally cannot be increased past what the map can reasonably handle.

To run these experiments, we modified our Python script with an outer loop cycling through the maps, an inner loop cycling through the 5 different scenarios, and a final innermost loop which ran the experiment with 1 to 150 agents. If at any point during the experiment the algorithm took longer than 120 seconds or could not find a solution, the scenario would be skipped and the loop would continue to the next one, or would skip to the next map if no scenarios remained. Each iteration ran the algorithm three times: once to collect the runtime, once to collect the Cachegrind output, and once to collect the Massif output.

Ultimately, we found that with the larger dataset the runtime continued to increase with the number of agents, just as it did in the prior experiments (Figure 12). This was measured in terms of the number of instructions read during the experiments. This appears to follow an exponential curve, just as the original paper argued it would. It is also important to note that with the increased instructions read, we start to see the number of cache misses also exponentially increase (Figure 13). The number of memory reads also increased exponentially (Figure 14).

We also collected the heap usage data. Because of varying results across maps, we have separated these graphs into 3 maps (Figure 15, Figure 16, and Figure 17). Curiously, on some maps the average peak heap usage increased linearly and others did not.

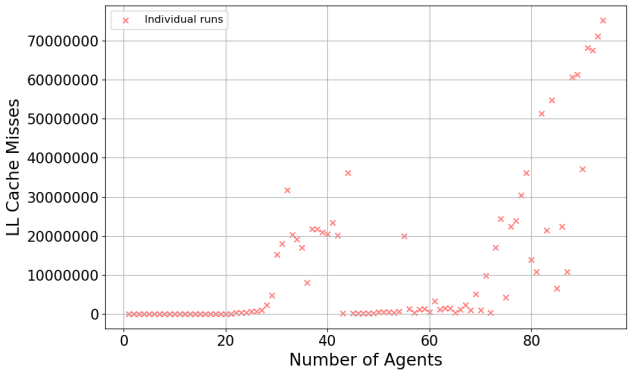

Figure 14: Number of **data read** Lower Level cache misses vs. number of agents

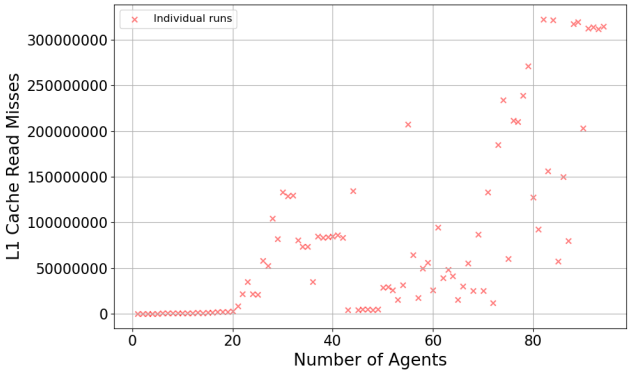

Figure 15: Number of **data read** cache misses at Level 1 Cache vs number of agents

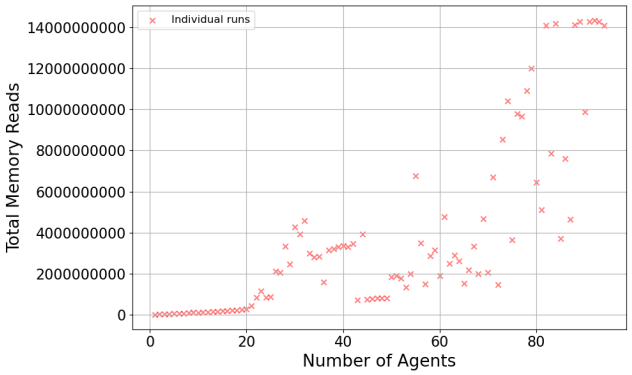

Figure 16: Total **memory reads** vs. number of agents

4. ANALYSIS

4.1 First Attempt

Our initial investigation into the CBS algorithm did not show if there is a determinate amount of time for which one should expect path-finding algorithm to run before it times out and attempts to restart the algorithm in hopes of finding a valid path on the second try. The C++ implementation of CBS which we are using does include a timeout length argument. In practical uses, timing out a motion planning process can

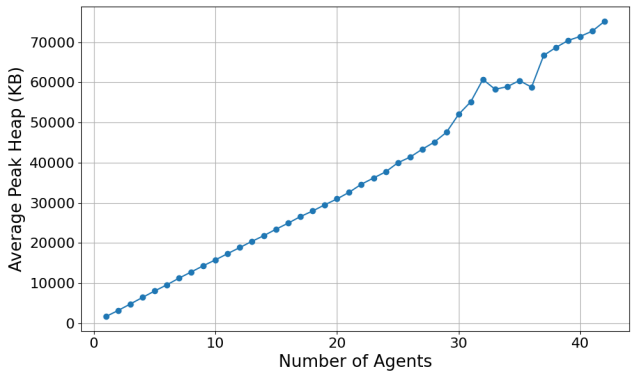

Figure 17: **w_woundedcoast.map**: Average peak heap vs. number of agents

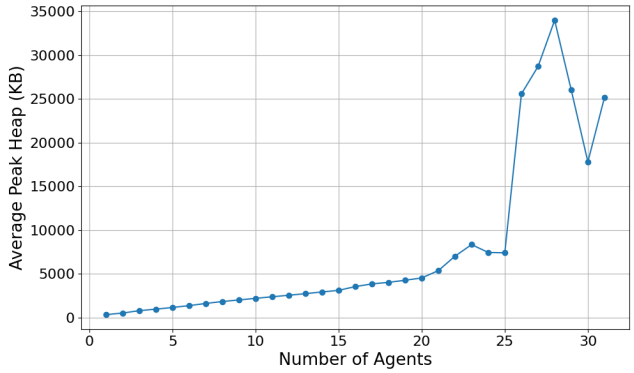

Figure 18: **It_gallowstemplar_n.map**: average peak heap vs. number of agents

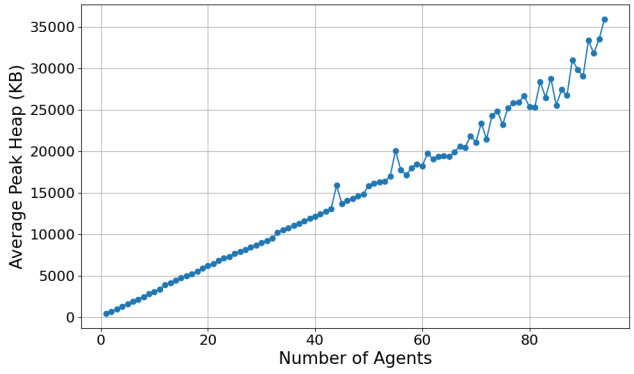

Figure 19: **Berlin_1_256.map**: average peak heap vs. number of agents

have drastic consequences for safety-critical operations such as search and rescue.

This memory usage trend suggests some interesting possibilities. The Linux kernel, used on our machine for this experiment, appears to statically allocate chunks of memory for the shell to use. After 20 MB, it would seem that the kernel switches to dynamic memory allocation, allocating the minimum amount of memory possible that CBS will need to complete its pathfinding task.

A critical consideration that we must point out is that all of this computation is being taken place on a desktop machine with essentially no limit on the amount of energy (Watts) that the CBS algorithm can use to find solutions. Autonomous robots do not have the luxury of limitless energy for computation and sacrifices need to be made for energy conservation that affects the performance of the CBS algorithm, especially considering the fact that computation time scales exponentially when using a higher number of agents. This constraint is based on the method of computation deployed to a robot swarm. While our system is more inline with a cloud-based solution (a host central computer would be closer to our implementation than would an embedded system on a robot, in terms of power capacity and compute capacity).

4.2 Second Attempt

Utilizing a faster workstation in combination with the tools provided by Valgrind yielded some interesting results that merit discussion. Trends appear when investigating cache misses between the Level 1 and Level 2 caches (lower level). It appears that in both graphs there seem to be two exponential trends happening when the number of agents reach about 40. There are a few possible theories for this. In an i9-14900K processor, there are two different types of processor core utilized when computing: performance cores and efficiency cores. The performance cores utilize a higher clock speed which results in faster computation times, but the drawback of utilizing performance cores is a smaller cache size, at half the size (2MB) of the efficiency cores. For these tests, it was out of our control which cores the Linux Kernel decided to use, and our current observations suggest that for the first half of the run the Linux Kernel decided to use a performance core for computation. Due to the extended computation time of the tests combined with the limited cache size of the performance core, the Linux kernel switched the computation to utilize an efficiency core, providing it with much higher headroom for memory storage when using an increasing number of agents. These results are in line with our expectations that CBS is inefficient to use at scale, as the number of memory read misses increases exponentially with the number of agents that are used within the algorithm.

Another result that reinforced our expectations of memory inefficiency occurred at the beginning of our computation. Initially, we programmed our run script to run concurrently to speed up computation. We quickly learned that this would not be possible due to the inefficiency of the CBS algorithm. Our workstation contained 64GB of ram, and that figure very quickly filled, and by the time our testing hit 50 agents, the Linux kernel needed to force crash our IDE software because it exceeded the amount of ram available in the system. Due to this fact, it is evident that the CBS algorithm needs to run

in a datacenter with much higher memory headroom if it is to be used with a large number of agents.

In general, the heap usage, as reported by Cachegrind, was linear. The Gallows Templar map experienced higher peak heap usage levels, but this may be due to the differing layout of the map causing additional difficulties due to its narrow hallways. Because the heap increases linearly instead of exponentially with the number of agents, it is not to be considered as one of the major bottlenecks of this algorithm. The algorithm primarily stresses the CPU.

There are a few factors that could be changed to provide more accurate results that show the inefficiencies of the CBS algorithm. Not having full control of the processor cores for the second round of testing proved to be disadvantageous due to the different cache sizes between the two types of cores. Research into a tool that controls where the Linux scheduler assigns tasks could be beneficial to the accuracy of data gathered. Another element of data collection that would have been useful is measuring the power required for CBS computation, as it would have been interesting to see any trends with power requirements vs. the scaling number of agents. Unfortunately a tool is not yet known to us that can measure the power usage of a specific program without taking into account the additional overhead of the operating system that the simulations were running on.

5. CONCLUSION

This paper evaluated the scalability and performance of Conflict-Based Search for Multi-Agent Pathfinding with a focus on the context of real-world robotic systems such as warehouse fleets, search and rescue swarms, and robot-power exploration. CBS is widely regarded for being mathematically sound, but its success in scaled deployment is limited by poor performance in real-time, high-agent conditions. Through a series of experiments using a C++ implementation of the algorithm, we confirmed that the algorithm's runtime grows exponentially with the number of agents (and by extension, the number of collisions), thus deepening theoretical expectations of the algorithm's performance as suggested by the original authors. Our computations empircally reinforce the theoretical time complexity described in the original proposal for CBS. The runtime behavior measured here poses a difficult challenge for scalability.

Our second experiment conducted a deeper analysis of CBS' memory behavior using Valgrind and Cachegrind, revealing that memory access becomes increasingly inefficient when CBS is deployed at scale. Our low-level analysis revealed some expected inefficiencies in memory access patterns correlated with, again, deployment to a higher number of agents. Cache misses grew rapidly when the number of agents was increased, further suggesting that CBS is not scalable. This development highlights the algorithm's limitations for deployment on hardware where memory constraints are important to consider, a common characteristic of decentralized robotics.

Our work opens the door to a restructuring of the CBS algorithm tailored to cases where constrained resources must be considered, namely for embedded systems that do not have the same level of computational power available as our testing system. There is a clear need to develop more informed variants of CBS that leverage the mathematical completeness of the algorithm while considering computational feasibility. By putting the low-level performance of Conflict-Based Search to the test and carefully measuring responses to scaling agents through a number of metrics, we hope to inform those who may work to improve CBS in the future.

6. APPENDIX

6.1 Individual Contributions

Gabi Guillermo

- Sacrificed workstation for computation
- Contributed to Python script for map iteration
- Contributed to data compilation and graph generation
- Researched state-of-the-art applications of the CBS algorithm

Vladislav Rada

- Created the framework for the Python script to run the experiments
- Added support to the Python script so that we could collect data via Valgrind tools
- Wrote portions of the report
- Contributed to data compilation and graph generation

Christopher Relyea

- Initial research for limitations of CBS/real-world relevancy
- Explored possible modifications to Python code, proposed (now unused) script ideas for generating custom environments and scenarios
- Contributed to Python script for iterating through maps/scenarios library
- Collected and organized files from MAPF benchmark dataset
- Paper formatting/LaTeX

REFERENCES

[1] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony, "Icbs: Improved conflict-based search algorithm for multi-agent pathfinding," in *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 2015, pp. 740–746.

[2] Financial Times, "Amazon steps up use of robotics in warehouses," ://www.ft.com/content/31ec6a78-97cf-47a2-b229-d63c44b81073, 2025.

[3] O. Gordon, Y. Filmus, and O. Salzman, "Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, no. 1, 2021, pp. 64–72.

[4] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[5] Y. Liu, H. Huang, Q. Xu, S. Xu, and J. Wang, "Safer conflict-based search: Risk-constrained optimal pathfinding for multiple connected and automated vehicles," *IEEE Transactions on Automation Science and Engineering*, pp. 1–15, 2024.

[6] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.