**Implementation, Testing, and Results**

To demonstrate the efficacy of the implementation strategy presented in Chapter ?, the design pattern was used in the creation of a simulation and test program named FleetBench. FleetBench was developed to provide a guided user interface (GUI) overtop a sufficiently performant and extensible simulation of multi-agent problems and solutions while remaining intuitive and simple, all with the goal of increasing the accessibility of implementing and testing solutions to MAPF and MAPD problems. FleetBench natively allows a user to define the number, order, and positions of agents and tasks. Before simulation, the end user is able to define a number of options which determine the behavior of the simulation at runtime, as discussed in Chapter ?. During simulation, a state machine designed exactly as presented in Chapter ? is used to drive the execution of all implemented algorithms. Data is collected and displayed continuously to the user during runtime, providing instant feedback about the performance of an algorithm. To aid in analysis, the state of the simulation is reconstructable from saved data at any particular timestep, providing an intuitive way to seek explanations for algorithmic failures. For convenience, Table ? lists all algorithms which are included in the application at time of writing. Developed in Python and using the native GUI library TkInter, the application exposes a customized rendering engine accessible in user-defined scripts which produces visualizations of found paths, agent motions, key point highlighting, and labeling on a per-state basis at the user’s request.

|  |  |  |
| --- | --- | --- |
| **Table 1** Listing FleetBench algorithms and appendix locations. | | |
| **Problem Type** | **Algorithm** | **Appendix** |
| Single-Agent Pathfinding | Single-Agent A\* | ? |
| MAPF | Multi-Agent A\* (LRA\*) | ? |
| Cooperative A\* (CA\*) | ? |
| Hierarchical CA\* (HCA\*) | ? |
| Windowed HCA\* (WHCA\*) | ? |
| MAPD | Token Passing (TP) | ? |
| TP with Task Swaps (TPTS) | ? |

FleetBench is designed to be extensible. By placing script files in the appropriate application path, a user is able to add additional algorithms to the program or modify the behavior of existing work. The state machine structure provides a regulated way of calling general functions which are listed in Appendix ?. These functions must be present in the algorithm scripts and some data must be returned in a certain format, but no further requirements are imposed.

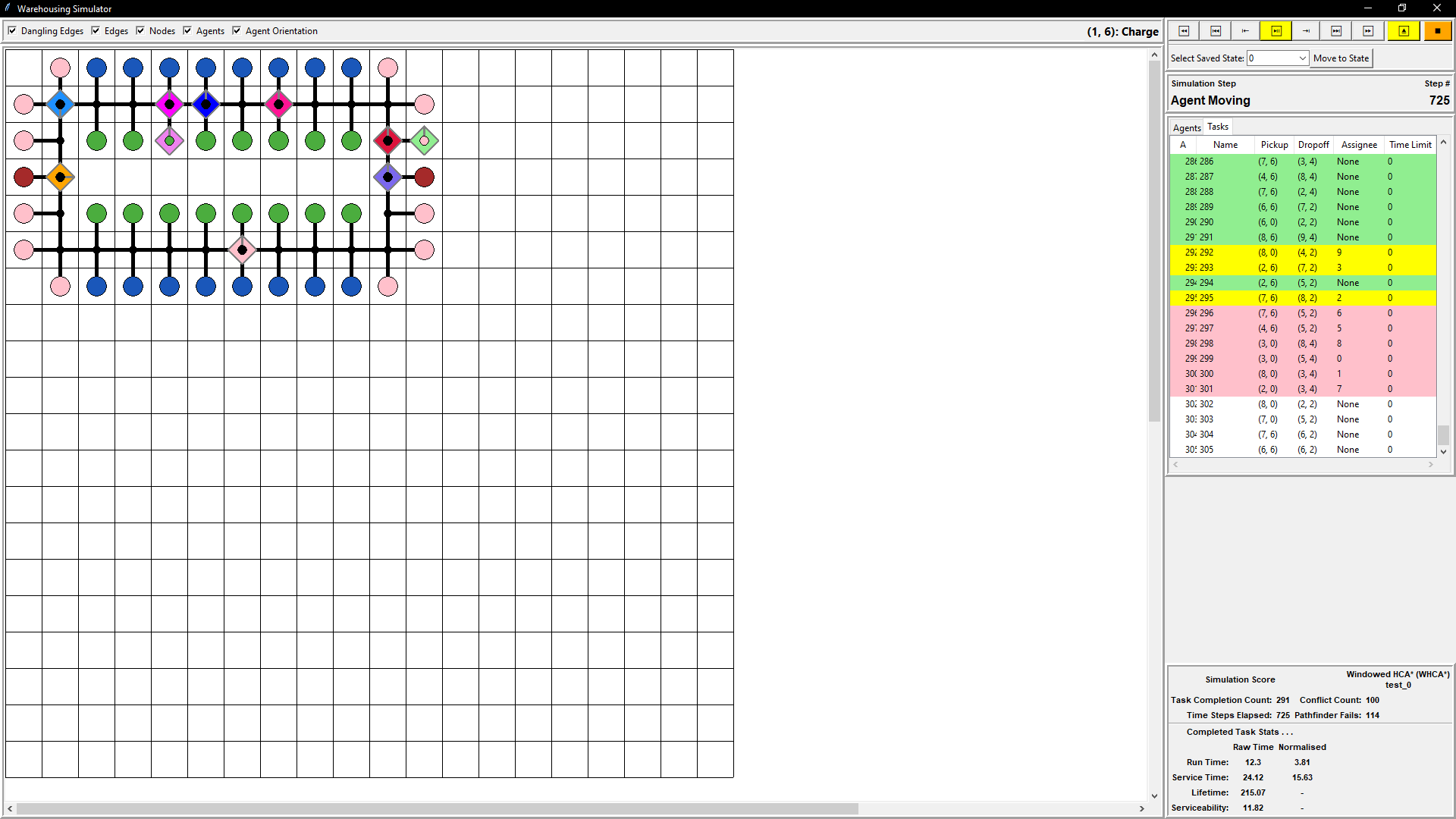


Figure 1: An image of the simulation window in FleetBench. A defined problem has been submitted and the program waits for a user to start simulating.

A second application called GraphRendering was developed to provide a visual process for designing the system tile map. It currently produces 4-neighbor graphs, with the ability to set specific node roles such as pickup, delivery, and rest. This application is also built in Python, using the TkInter GUI library. It is possible to generate maps externally, using the map input file format given in Appendix ?.

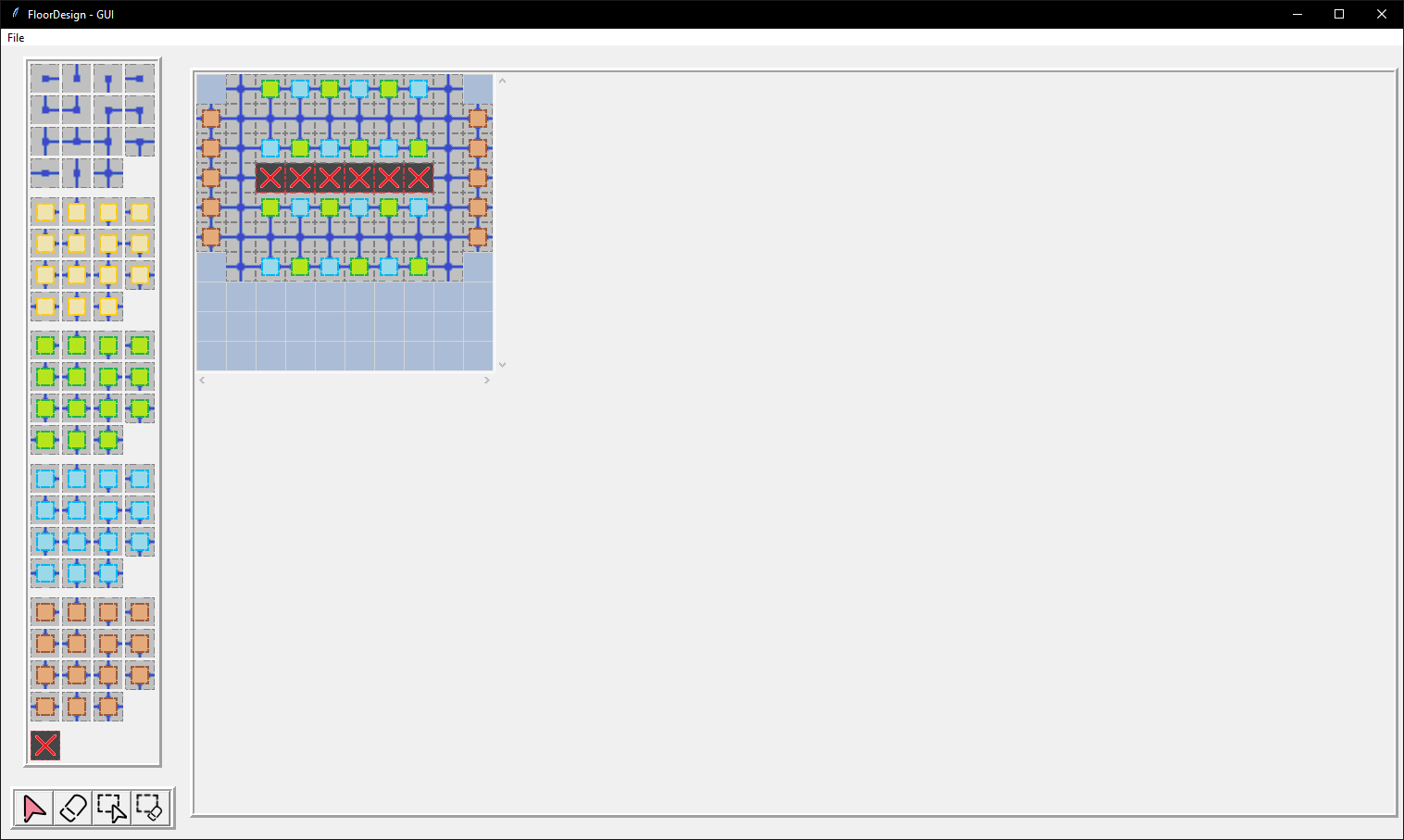


Figure 2: Screenshot of the map creation process in GraphRendering.

The intended workflow for a user wishing to evaluate the performance of an algorithm on their multi-agent system is as follows:

* Develop a map either via the GraphRendering application or a custom script, adhering to the map file format given in Appendix ?.
* Create an implementation of the algorithm if one is not already provided, adhering to the extension documentation in Appendix ?.
* In FleetBench, create a new session using the map file.
* Design the initial placement of agents in the system.
* Optionally, define an initial set of tasks.
* Define simulation configuration options, including which algorithm should be used, how new tasks are introduced to the system and upon what conditions (if any) the simulation should end.
  + If a predefined list of tasks should be used, the user will need to provide a comma-separated values file as described in Appendix ?.
* Run the simulation, recording the resulting data for analysis.

Because of the effect the past has on how a simulation proceeds in the future, it is expected that variation of individual parameters will produce significant changes in the performance of an algorithm. Care should be taken to ensure that the results of different tests are treated fairly in analysis. Repeatability of results is an important factor, which FleetBench adheres to by using the same pseudorandom generator for all operations which require a “random” choice.

A screenshot of a computer

Description automatically generated

Figure 3: Data panel of a simulation in FleetBench.

**Design of Experiment**

Several test cases were produced to demonstrate FleetBench’s implementation of the MAPF and MAPD algorithms. Test cases were developed quickly using the GUI of GraphRendering for the map design and the features in FleetBench to design agents and task schedules, supporting the usability and flexibility of the applications. They are designed to show certain features and behaviors of algorithms and multi-agent problems in general. Case 1 shows a bottleneck around access to a single node which is used repeatedly. Case 2 demonstrates an analogy of a real-world warehouse problem. Case 3 demonstrates the problem of agents always seeking to minimize the A\* algorithm’s *h-Score* without considering the real-world cost of motion, as all implementations used consider all edge weights to be equivalent.

**Case 1**

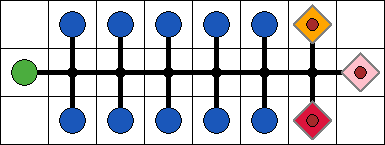


Figure 4: Test case 1. A 1-width corridor with three agents, shown in orange, pink, and red. The green, leftmost, node is the deposit location, while blue nodes are pickup locations. The agent start positions are endpoints satisfying TP and TPTS requirements.

The first case could be analogized as cooperatively retrieving books in a library. There is a single task delivery node (a cart of books), acting as the ending location for items retrieved from the storage system (the library shelves) managed by three agents. Several nodes exist to represent the many sorting locations at which items may be stored. In this relatively constrained space, the motion of agents presents a challenge wherein agents must dip into destination spaces to avoid collisions moving down the corridor, which may itself impact the delivery of task objects to their destinations.

The simulation configuration options used to obtain the results in the next section are given in Table 1. The data was collected once for HCA\* and WHCA\* and again for TP and TPTS to show performance differences between the “upgraded” versions of the algorithms, as well as between the two families of algorithms. Each choice for the simulation is configurable, encouraging experimentation and repeat trials.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table 1** System configuration for Case 1 experiments | | | | |
| **Option** | **HCA\*** | **WHCA\*** | **TP** | **TPTS** |
| Map Name | case\_1 | | | |
| Agent Starting Positions | {(6,0), (7,1), (6,2)} | | | |
| Initial Task Set | None | | | |
| A\* Heuristic Function | Manhattan Distance | | | |
| A\* Heuristic Relaxation Coefficient | 1 | | | |
| Window Size |  | 5 |  |  |
| Agent Collisions | Respected | | | |
| Task Interaction Time Cost | Instantaneous | | | |
| Task Schedule | case\_1\_schedule (Appendix ?) | | | |
| End Condition | All Scheduled Tasks Completed | | | |

**Case 2**

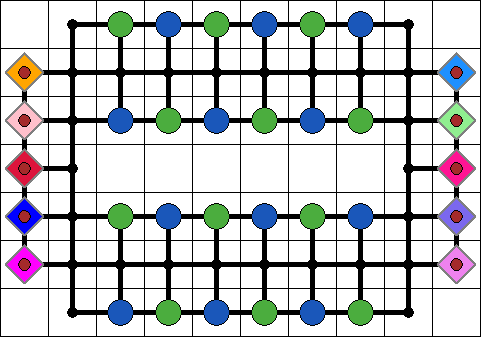


Figure 5: Test case 2. A warehouse reorganization problem with 10 agents, starting on the sides of the system map. The agents start on non-task endpoints, satisfying the TP and TPTS well-formed requirements.

A second test case is developed, very similar in form to the warehousing situation presented in [1]. In this case the analogy is closer to warehouse internal reorganization, with an arbitrary pattern of pickup and delivery nodes. Agents must again travel through corridors, only this time the spaces are more interconnected, allowing agents to move over task endpoints during their journeys. This widens the bottleneck considerably, allowing a greater number of agents to be present in the system.

As before, simulation configuration options are presented in Table ?. For this experiment only the implementations of WHCA\*, TP, and TPTS are compared. To demonstrate changes in performance of the WHCA\* algorithm as the window size is changed, the WHCA\* experiments are repeated for a few window sizes.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table 2** System configuration for Case 2 experiments | | | | | | | |  | |
| **Option** | **WHCA\*-3** | | **WHCA\*-5** | | **WHCA\*-10** | | **TP** | | **TPTS** |
| Map Name | case\_2 | | | | | | | | |
| Agent Starting Positions | {(0,1), (0,2), (0,3), (0,4), (0,5), (9,1), (9,2), (9,3), (9,4), (9,5),} | | | | | | | | |
| Initial Task Set | None | | | | | | | | |
| A\* Heuristic Function | Manhattan Distance | | | | | | | | |
| A\* Heuristic Relaxation Coefficient | 1 | | | | | | | | |
| Window Size | 3 | 5 | | 10 | |  | |  | |
| Agent Collisions | Respected | | | | | | | | |
| Task Interaction Time Cost | Instantaneous | | | | | | | | |
| Task Schedule | case\_2\_schedule (Appendix ?) | | | | | | | | |
| End Condition | All Scheduled Tasks Completed | | | | | | | | |

**Case 3**

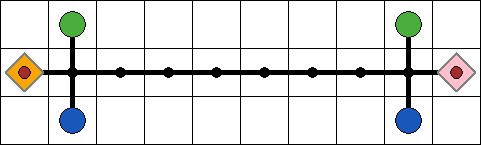


Figure 6: Test case 3. A 1-width bottlenecked corridor with task endpoints at each side. Two agents share the space and are forced to use the nodes at the ends of the corridor to exchange places.

This test case exposes a consequence of treating all edge weights as having the same value when finding paths using A\*. Two agents must share a long corridor with no ability to exchange positions except at the corridor’s ends. The heuristic defined by the code in Appendix ? encourages agents to always move in the direction of their goals as immediately as possible. This behavior can create situations in which agents perform pointless movement actions toward a goal that is impossible to travel to without backpedaling. The resulting sequence will show that they must move backward to avoid the path of the blocking agent until it is possible for the two agents to exchange positions via some rotation through nodes outside the corridor.

This behavior is discussed in the Results section of this chapter. Once again, the configuration options used for this simulation are presented in Table ?. The windowing behavior of WHCA\* is of special interest. Its performance is compared to HCA\*, which plans full paths when possible, and against CA\*. LRA\* simply fails to solve the problem, as noted in [2].

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 3** System configuration for Case 3 experiments | | | |
| **Option** | **CA\*** | **HCA\*** | **WHCA\*** |
| Map Name | case\_3 | | |
| Agent Starting Positions | {(0,2), (9,2)} | | |
| Initial Task Set | None | | |
| A\* Heuristic Function | Manhattan Distance | | |
| A\* Heuristic Relaxation Coefficient | 1 | | |
| Window Size |  |  | 5 |
| Agent Collisions | Respected | | |
| Task Interaction Time Cost | Instantaneous | | |
| Task Schedule | case\_3\_schedule (Appendix ?) | | |
| End Condition | All Scheduled Tasks Completed | | |

**Results**

Experiments are run on each test case in the manner described in the previous section, with the aim of evaluating the performance of each algorithm within the system. By equalizing the playing field as discussed, it is possible to draw quantitative conclusions. FleetBench implements a rudimentary set of datapoints for which data is collected during the simulation lifetime. The results are constantly displayed, as shown in Figure 3.

Because all test cases are equalized using the same task scheduling system, an obvious performance metric to examine is the amount of time taken to complete the schedule. This is simply equivalent to the number of timesteps elapsed while using a simulation end condition which triggers when all scheduled tasks are completed.

FleetBench also records some data regarding the timing of interactions with tasks as a measure of the moment-to-moment performance of the system. FleetBench names four different aspects of task completion, each defined by a different calculation. There are four moments in time relevant to the processing of a task:

* Task Creation: The timestep at which the task is entered into the simulation’s active task list.
* Task Assignment: The first timestep at which an agent is assigned to the task.
* Task Pickup: The timestep at which an agent executes the first portion of the task by “picking up” the task for delivery.
* Task Completion: The timestep at which an agent executes the last portion of the task by “delivering” the task to the endpoint.

From these definitions four time intervals of interest are developed. Run Time captures the time spent by agents completing a task once the agent reaches and completes the “pickup” portion of the task. Service Time measures the amount of time agents are preoccupied with moving toward or executing a task. Lifetime measures how long tasks spend in the system before being completed. As the Lifetime of tasks increases, the system is falling further and further behind the incoming task set. Serviceability is an approximation of how long it takes agents to reach the next task, upon assignment, from their current position. If agents travel very long distances to reach the next task (as may be the case in unoptimized assignment implementations) this number will be quite large. These definitions give rise to the following formula:

Two additional measures are provided as normalizations of the baseline measure. In this application, the normalization is intended to capture an idea of how much time is spent on a task compared to the optimal minimum time needed for any given interval. This value is only consistently defined using the minimum travel time from pickup endpoint to delivery endpoint of the task, as an agent assigned to a task could be anywhere in the system. This minimum time interval for the task to be completed is determined using an A\* search which ignores the presence of all agents and obstacles in the system, similar to the HCA\* approach presented in [2]. In FleetBench, the Run Time is normalized, capturing timesteps spent avoiding collisions during an agents path from pickup to delivery endpoints. The Service Time is also normalized, providing an idea of the combined time loss due to congestion (Run Time) and time lost due to inoptimal starting position of agents servicing a task. The two values are given by these formula:

The values reported by FleetBench are the simulation mean values for all above formula, calculated via summation of all values and division by the number of tasks in the system. Normalized values before the end of a simulation should be treated with caution, as the averaging of minimum times is done for completed tasks while the system may have many tasks in progress. No attempt is made to measure an agent’s progress on a per-timestep basis.

Two kinds of failure are possible during the execution of an algorithm. The first is a collision, where two agents plan paths which result in a vertex or edge conflict. Such failures are termed “Agent Conflicts” to distinguish from the second type of failure. When an agent seeks a path and is unable to find any route to its destination node the failure is called a “Pathfinder Failure”. This is frequently due to the agent searching for a path being trapped in its location, rather than the obstructions being distant as the search depth would simply increase until the path is found. Typically when this occurs the agent is about to experience a conflict anyway, so the failure counts tend to be coupled.

With metrics defined, the results of experiments on the three test cases provided can be discussed.

**Case 1**

Case 1 is simulated using the task schedule provided in Appendix ?, and the resulting data are provided in Table 4. All algorithms from Table 1 successfully solve the multi-agent problem such that all tasks are completed, although the WHCA\* family of algorithms notably suffers many agent conflicts and failed pathfinding operations. However, the fundamental operating principles of TP and TPTS prevent it from being performant in this situation. Because agents are prevented both from being assigned tasks and from planning paths ending in the same location as any other agents, the bottleneck becomes very problematic. With only one destination node, it is not possible for any second agent to plan paths in the system. Therefore all tasks are completed by the same agent in this case, resulting in a massive increase in elapsed timesteps. This further translates to a longer task lifetime.

However, because all tasks assigned are the best tasks for an agent to be executing given its position, the normalized task run times are very optimal. The serviceability measure in this case largely represents information about the system map, as agents mostly travel through the same space repeatedly in all algorithms tested, never having to travel inordinately far to reach an outlier task.

HCA\* compared to WHCA\* takes overall slightly fewer timesteps to finish solving the problem. This is because HCA\*, by planning complete paths, avoids a small number of frivolous movements due to staggered path planning and WHCA\* window size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table 4** Results of experimentation on Case 1 | | | | |
| **Option** | **HCA\*** | **WHCA\*** | **TP** | **TPTS** |
| Tasks Completed | 90 | 90 | 90 | 90 |
| Timesteps Elapsed | 377 | 383 | 1008 | 729 |
| Agent Conflicts | 76 | 76 | 0 | 0 |
| Pathfinder Failures | 59 | 60 | 0 | 0 |
| Run Time | 6.58 | 6.86 | 4.42 | 4.1 |
| Run Time (normalized) | 2.48 | 2.76 | 0.32 | 0 |
| Service Time | 12.38 | 12.61 | 10.87 | 8.09 |
| Service Time (normalized) | 8.28 | 8.51 | 6.77 | 3.99 |
| Lifetime | 110.88 | 117.73 | 425.87 | 231.93 |
| Serviceability | 5.8 | 5.76 | 6.44 | 3.99 |

These data support initial hypotheses about the performance of these algorithms based on their properties. According to this experiment, for a small and relatively simple situation involving object retrieval to a single deposit location, HCA\* is the ideal algorithm to be used. It is possible that performance issues arise in larger and more complicated scenarios (which can also be tested in FleetBench), in which case using WHCA\* (with an appropriate choice for window size) appears likely to provide a sufficient solution.

**Case 2**

Case 2 is simulated using the task schedule provided in Appendix ?, and the resulting data are provided in Table 5. Being a very interconnected graph, there are many degrees of freedom for most agents attempting to find paths. This fact appears to produce comparable solutions with all algorithms, and few conflicts when using algorithms which do not assure generality. All five algorithms seem to complete the task schedule in a similar amount of time, except for TP. The very similar run time values indicate that agents’ paths were typically unobstructed after picking up the task. WHCA\*-3 did experience elevated run times, likely because of its low planning depth leading to more frequent replanning to avoid collisions as agents more easily claim paths which end up disrupting other agents. Increasing the window size appears to result in more collisions, as a consequence of greater planning depth introducing more reservations in any given area over time.

TPTS experiences some deviation in timing values, each of which have several explanations. By carefully examining the playback of the simulation, it becomes obvious that in certain situations agents plan excessively large numbers of waiting moves, likely waiting for their objectives to be clear of agents which are waiting for new tasks. This results in high service times and is a known issue in TP and TPTS [3]. Additionally, agents tend to begin the simulation by moving in similar directions. This results in task selections clogging up areas with the restriction that no plan end in the same location as another plan in the system. As a result, agents are frequently forced to choose tasks which are actually further away and resolve in an entirely different area of the system, where there are not agents currently operating. This results in higher service times. Alternatively, certain tasks are ignored for a significant amount of time as they happen to not be near agents when agents are free. Some benefits are still retained, however, as agents will still attempt to complete the fastest tasks soonest, reducing the overall average lifetime of tasks. These patterns, not as visible in TP, prompt questions about the task assignment optimization.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table 5** Results of experimentation on Case 2 | | | | | |
| **Option** | **WHCA\*-3** | **WHCA\*-5** | **WHCA\*-10** | **TP** | **TPTS** |
| Tasks Completed | 150 | 150 | 150 | 150 | 150 |
| Timesteps Elapsed | 232 | 225 | 224 | 275 | 249 |
| Agent Conflicts | 1 | 4 | 5 | 0 | 0 |
| Pathfinder Failures | 1 | 4 | 5 | 0 | 0 |
| Run Time | 7.53 | 7.09 | 7.07 | 7.89 | 7.38 |
| Run Time (normalized) | 1.53 | 0.9 | 0.88 | 1.7 | 1.19 |
| Service Time | 14.79 | 14.25 | 14.01 | 14.69 | 25.38 |
| Service Time (normalized) | 8.6 | 8.07 | 7.82 | 8.5 | 19.19 |
| Lifetime | 71.57 | 66.35 | 63.39 | 73.08 | 56.09 |
| Serviceability | 7.25 | 7.17 | 6.94 | 6.8 | 18.0 |

Once again the results seem to support the hypothesized behavior of these algorithms, affirming the efficacy of FleetBench in testing warehousing situations with a variety of settings. It also enabled the user to investigate specific performance cases, resulting in a deeper understanding of how and why TP and TPTS may struggle to optimize certain axes of performance.

**Case 3**

Case 3 is simulated using the task schedule provided in Appendix ?, and the resulting data are provided in Table 6. This test case is largely intended to be watched rather than analyzed, and exposes a consequence of failing to perfectly analogize the system to real-world applications. Specifically, the FleetBench implementation assumes that a movement from one node to another incurs the same costs as staying in place. Because the path planner always seeks to move in the direction of least distance from the goal, an agent may pointlessly advance toward its goal even if it knows it will have to move backward in the future to avoid the path of another agent sharing the space. In a real-world application, this would result in wasted fuel costs, unnecessary wear, and a greater risk of agents experiencing crashes or operational faults. This sequence is represented in Figure ?.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) | (b) | (c) |
|  |  |  |
| (d) | (e) | (f) |

Figure 7: Sequential moves by two agents. Pink is attempting to move left, while orange moves right. It can be seen in images (a) and (b) that pink makes moves in its goal direction which it must then undo in images (c) and (d). In (e) and (f), the orange agent moves out of the way, allowing pink to move to its goal direction unimpeded.

WHCA\* seems to slightly underperform compared to HCA\*, just as in Case 1, further supporting the idea that optimal paths are found when full plans are made. CA\* and HCA\* perform identically. As expected, the optimization of the abstract hierarchy is merely one of computation time and data reuse.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 6** Results of experimentation on Case 3 | | | |
| **Option** | **CA\*** | **HCA\*** | **WHCA\*** |
| Tasks Completed | 26 | 26 | 26 |
| Timesteps Elapsed | 159 | 159 | 187 |
| Agent Conflicts | 5 | 5 | 5 |
| Pathfinder Failures | 5 | 5 | 5 |
| Run Time | 6.08 | 6.08 | 7.92 |
| Run Time (normalized) | 0.31 | 0.31 | 2.15 |
| Service Time | 11.92 | 11.92 | 14.08 |
| Service Time (normalized) | 6.15 | 6.15 | 8.31 |
| Lifetime | 49.96 | 46.96 | 63.5 |
| Serviceability | 5.85 | 5.85 | 6.15 |

These results demonstrate that the algorithms perform in a consistent manner, providing similar results as in case Case 1 but in a different environment. It also demonstrates the utility of FleetBench’s playback function as a visual tool for gaining insight into the operating principles of the underlying implementation.

[1] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, “Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks.” arXiv, May 30, 2017. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1705.10868

[2] D. Silver, “Cooperative Pathfinding,” *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.*, vol. 1, no. 1, pp. 117–122, Sep. 2021, doi: 10.1609/aiide.v1i1.18726.

[3] F. Grenouilleau, W.-J. V. Hoeve, and J. N. Hooker, “A Multi-Label A\* Algorithm for Multi-Agent Pathfinding,” *Proc. Int. Conf. Autom. Plan. Sched.*, vol. 29, pp. 181–185, May 2021, doi: 10.1609/icaps.v29i1.3474.