Dynamic Task Allocation in Partially Defined Environments Using A* with Bounded Costs

James Hendrickson
Embry-Riddle Aeronautical University, hendrj11@my.erau.edu

Follow this and additional works at: https://commons.erau.edu/edt

Part of the Artificial Intelligence and Robotics Commons, Robotics Commons, and the Theory and Algorithms Commons

Scholarly Commons Citation
https://commons.erau.edu/edt/592

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in PhD Dissertations and Master's Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.
DYNAMIC TASK ALLOCATION IN PARTIALLY DEFINED ENVIRONMENTS
USING A* WITH BOUNDED COSTS

by

James Joseph Hendrickson

A Thesis Submitted to the College of Engineering Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

Embry-Riddle Aeronautical University
Daytona Beach, Florida
May 2021
DYNAMIC TASK ALLOCATION IN PARTIALLY DEFINED ENVIRONMENTS USING A* WITH BOUNDED COSTS

by

James Joseph Hendrickson

This thesis was prepared under the direction of the candidate’s Thesis Committee Chair, Dr. Eric Coyle, Professor, Daytona Beach Campus, and Thesis Committee Members Dr. Patrick Currier, Professor, Daytona Beach Campus, and Dr. M. Ilhan Akbas, Professor, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Thesis Review Committee:

____________________________________
Eric Coyle, Ph.D.
Committee Chair

_________________________________
Patrick Currier, Ph.D.
Committee Member

Jean-Michel Dhainaut
Jean-Michel Dhainaut, Ph.D.
Graduate Program Chair,
Mechanical Engineering

Maj Dean Mirmirani
Maj Mirmirani, Ph.D.
Dean, College of Engineering

_________________________________
M. Ilhan Akbas, Ph.D.
Committee Member

Eduardo Divo
Eduardo Divo, Ph.D.
Department Chair,
Mechanical Engineering

________________________________
Christopher Grant, Ph.D.
Associate Vice President of Academics

Date
Acknowledgements

I would like to first and foremost thank my wonderful parents whose constant love, support, and encouragement has helped me achieve all that I have in life. I would also like to thank my younger sister, Sarah, who I constantly strive to do my best for so I can be someone she can look up to. Their support has allowed me to have overcome all the trials and tribulations I have faced over the years and pushed me to keep challenging myself. I truly am blessed to call them my family. I would also like to thank my grandparents, may they rest in peace, for all their love and support when I was growing up. I wish you could’ve seen all I have been able to achieve and will do in the future. To the rest of my family, I love you all and thank you for all of your support over the years.

I would also like to thank my roommates Andrew Ferree, Daniel Bussey, & Alex Bassett. You all have helped to make this whole experience fun and tolerable and truly feel like brothers to me. I would like to thank my advisors, Dr. Coyle, Dr. Currier, & Dr. Akbas for putting up with me throughout this process and for helping me to overcome the obstacles I faced. I would like to thank the other faculty, particularly Dr. Reinholtz, Dr. Butka, & Dr. Hockley, I have had the pleasure of working with throughout college. You all shaped me into the engineer I am today. To my high school speech and debate coaches, Ms. Smith & Ms. White, I would like to thank you both for helping to pull me out of my shell and for honing my public speaking skills. I would also like to thank my high school FRC coach, Mr. Smith, for helping me find my passion for robotics.

Finally, more than anything else, I would like to thank Tijuana Flats. Your food has carried me through thick and thin, through good times and bad. You truly made college enjoyable. Also thanks to cigarette man for always being on the corner for us.
Abstract

Researcher: James Joseph Hendrickson

Title: DYNAMIC TASK ALLOCATION IN PARTIALLY DEFINED ENVIRONMENTS USING A* WITH BOUNDED COSTS

Institution: Embry-Riddle Aeronautical University

Degree: Master of Science in Mechanical Engineering

Year: 2021

The sector of maritime robotics has seen a boom in operations in areas such as surveying and mapping, clean-up, inspections, search and rescue, law enforcement, and national defense. As this sector has continued to grow, there has been an increased need for single unmanned systems to be able to undertake more complex and greater numbers of tasks. As the maritime domain can be particularly difficult for autonomous vehicles to operate in due to the partially defined nature of the environment, it is crucial that a method exists which is capable of dynamically accomplishing tasks within this operational domain. By considering the task allocation problem as a graph search problem, Minion Task, is not only capable of finding and executing tasks, but is also capable of optimizing costs across a range of parameters and of considering constraints on the order that tasks may be completed in. Minion task consists of four key phases that allow it to accomplish dynamic tasking in partially defined environments. These phases are a search space updater that is capable of evaluating the regions the vehicle has effectively perceived, a task evaluator that is capable of ascertaining which tasks in the mission set need to be searched for and which can be executed, a task allocation process
that utilizes a modified version of the A* with Bounded Costs (ABC) algorithm to select
the best ordering of task for execution based on an optimization routing, and, finally, a
task executor that handles transiting to and executing tasks orders received from the task
allocator. To evaluate Minion Task’s performance, the modified ABC algorithm used by
the task allocator was compared to a greedy and a random allocation scheme.

Additionally, to show the full capabilities of the system, a partial simulation of the 2018
Maritime RobotX competition was utilized to evaluate the performance of the Minion
Task algorithm. Comparing the modified ABC algorithm to the greedy and random
allocation algorithms, the ABC method was found to always achieve a score that was as
good, if not better than the scores of the greedy and random allocation schemes. At best,
ABC could achieve an up to 2 times improvement in the score achieved compared to the
other two methods when the ranges for the score and execution times for each tasks in the
task set as well as the space where these tasks could exists was sufficiently large. Finally,
using two scenarios, it was shown that Minion Task was capable of completing missions
in a dynamic environment. The first scenario showed that Minion Task was capable of
handling dynamic switching between searching for and executing tasks. The second
scenario showed the algorithm was capable of handling constraints on the ordering of the
tasks despite the environment and arrangement of tasks not changing otherwise. This
paper succeeded in proving a method, Minion Task, that is capable of performing
missions in dynamic maritime environments.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis Review Committee:</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>ii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Chapter I</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Significance of the Study</td>
<td>1</td>
</tr>
<tr>
<td>Problem Statement</td>
<td>6</td>
</tr>
<tr>
<td>Delimitations</td>
<td>6</td>
</tr>
<tr>
<td>Limitations and Assumptions</td>
<td>7</td>
</tr>
<tr>
<td>List of Acronyms</td>
<td>8</td>
</tr>
<tr>
<td>Chapter II</td>
<td>9</td>
</tr>
<tr>
<td>Review of the Relevant Literature</td>
<td>9</td>
</tr>
<tr>
<td>Agent Composition</td>
<td>10</td>
</tr>
<tr>
<td>Task Decomposition</td>
<td>14</td>
</tr>
<tr>
<td>Task Allocation</td>
<td>19</td>
</tr>
<tr>
<td>Chapter III</td>
<td>27</td>
</tr>
<tr>
<td>Methodology</td>
<td>27</td>
</tr>
<tr>
<td>Agent Definition</td>
<td>28</td>
</tr>
<tr>
<td>Task Definition</td>
<td>30</td>
</tr>
<tr>
<td>Process</td>
<td>32</td>
</tr>
<tr>
<td>Chapter IV</td>
<td>53</td>
</tr>
<tr>
<td>Results</td>
<td>53</td>
</tr>
<tr>
<td>Allocator Evaluation</td>
<td>53</td>
</tr>
<tr>
<td>Case Study</td>
<td>78</td>
</tr>
<tr>
<td>Chapter V</td>
<td>97</td>
</tr>
<tr>
<td>Discussion, Conclusions, and Recommendations</td>
<td>97</td>
</tr>
<tr>
<td>Discussion</td>
<td>97</td>
</tr>
<tr>
<td>Recommendations</td>
<td>99</td>
</tr>
<tr>
<td>Conclusions</td>
<td>100</td>
</tr>
</tbody>
</table>
List of Tables

Table

Table 1: Summary of task allocation methods in literature (Rizk et al., 2019) .................. 3
Table 2: DoD unmanned system roadmaps for interoperability, autonomy, and human-machine collaboration themes (Department of Defense, 2018) ......................................... 5
Table 3: Key elements defining an agent .............................................................................. 14
Table 4: Key elements defining a task .................................................................................. 19
Table 5: Task evaluation algorithm ....................................................................................... 38
Table 6: Reproduced A* with Bounded Costs algorithm (Logan & Alechina, 1998, p. 8). ........................................................................................................................................... 43
Table 7: Sample successor generation algorithm which handles direct/indirect predecessor and direct successor requirements for the task ordering ..................................... 49
Table 8: 100m x 100m region baseline configuration for 200 scenarios run. ..................... 57
Table 9: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 18. ................................................................. 58
Table 10: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 19. ................................................................. 58
Table 11: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 20. ................................................................. 59
Table 12: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 21. ................................................................. 60
Table 13: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 22. ................................................................. 61
Table 14: Summary of the average score differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13) .................. 61
Table 15: Summary of the maximum score differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13) .... 62
Table 16: Summary of the average time efficiency differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13) .... 62
Table 17: Configuration with expanded score range. ................................................... 64
Table 18: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 23. ......................................................... 65
Table 19: Configuration with expanded execution time range. .................................. 67
Table 20: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 24. ......................................................... 68
Table 21: Configuration with expanded score and execution time ranges. ................. 69
Table 22: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 24. ......................................................... 70
Table 23: Configuration for the larger task region. ...................................................... 72
Table 24: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 26. ......................................................... 73
Table 25: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 27. ......................................................... 73
Table 26: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 28. ................................................................. 74

Table 27: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 29. ................................................................. 75

Table 28: Summary of the average score differences between ABC vs greedy and random methods for each configuration and each configuration space size. ................................. 76

Table 29: Summary of the maximum score differences between ABC vs greedy and random methods for each configuration and each configuration space size. ....................... 76

Table 30: Summary of the average time efficiency differences between ABC vs greedy and random methods for each configuration and each configuration space size. .......... 76

Table 31: General scenario configuration parameters. .......................................................... 85

Table 32: Task assigned scores. .................................................................................... 86
List of Figures

Figure

Figure 1: Task decomposition workflow typically found in literature for MRS (Rizk et al., 2019). .............................................................. 15

Figure 2: Visibility region, $PFRD$, used for these scenarios. This region is based on the Minion ASV’s perception region with the vehicle centered at Northing: 0 m; Easting: 0 m in this FRD frame (Barnes et al., n.d.; D. Thompson et al., 2019). ................................. 30

Figure 3: General task allocation process. .............................................................. 33

Figure 4: (Left) Individual task search regions, $task.SA$, and mission objects for this configuration. (Right) Global search area, $GSA$, bounding all task search regions. .......... 36

Figure 5: (Left) $GSA$ and $PFRD$ before transit begins. (Right) $GSA$ representation as the mission progresses and clipping occurs. .......................................................... 37

Figure 6: (Left) Global Search Area, $GSA$, and Task 1 Search Area, $task1.SA$, prior to intersection operation. (Right) $GSA \cap task1.SA$ representation and predicted target search waypoint for the given vehicle position, $LocVeh$. ................................................................. 40

Figure 7: Sample task configuration visualized as a connectivity graph where the S node is the starting node and the remaining nodes are the task nodes. ................................. 41

Figure 8: Initialization of ABC for the sample task allocation problem with the start, $tskStart$, added to the open set. For this problem, task 2 is the goal task and a maximum allowable allocation time constraint is 450 seconds. .......................................................... 44

Figure 9: Top element in the open set is removed for evaluation. It is also simultaneously added to the closed set. ................................................................. 44
Figure 10: Successors, n’, are generated for the element, n, under consideration. The successors are costed and then checked for domination. In this iteration, no domination occurs; however, a case where domination occurs is presented later in the evaluation process. 45

Figure 11: All successors, n’, not dominated are added to the open set. The open set is then sorted based on the equivalence class sorting method utilized. Here, a single equivalence class is used, so the elements are simply sorted within that class. 45

Figure 12: Example iteration where a domination occurs. Nothing in the open set is dominated by the successor, n’. While n’ does have a score cost that is the same or better than the elements in the open set, the time cost for n’ is worse than ever element in the open set. So, the conditions for domination of n’ over the open set elements is not met. However, the successor, n’, is instead dominated by an element on the open set. Since an element on the open (circled in red) set has the same score cost as the successor and has a lower time cost, it dominates the circled successor. Thus, this successor is removed. 46

Figure 13: If no successors remain after checks for domination occur, then no elements are added to the open set. The open set is ordered as before. 47

Figure 14: When an element that has the goal state as the final point in the task allocation “path”, then the goal is considered to be reached and the process is finished. Should the open set become empty and the goal had not been reached, then the no viable allocation would be available. 47

Figure 15: Task Executor general process flow. Phase 1 (red) shows the processes for updating the current Task Allocation and Task Costs. Phase 2 (blue), 3 (yellow), and 4 (green) show the operations that occur based on the mode the executor is in currently. 51
Figure 16: 3 tasks per scenario for showing score convergence by scenario 200. .......... 54
Figure 17: 12 tasks per scenario showing score convergence by scenario 200. .......... 55
Figure 18: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points. ......................................................... 57
Figure 19: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points. ......................................................... 58
Figure 20: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points. ..................................................... 59
Figure 21: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points......................................................... 60
Figure 22: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points..................................................... 61
Figure 23: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points......................................................... 65
Figure 24: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points......................................................... 68
Figure 25: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points......................................................... 70
Figure 26: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region , an execution time range of 40-60 seconds, and a score range of 80-120 points......... 72
Figure 27: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region , an execution time range of 40-60 seconds, and a score range of 20-180 points............. 73
Figure 28: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 10-90 seconds, and a score range of 80-120 points.............. 74
Figure 29: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 10-90 seconds, and a score range of 20-180 points.............. 75
Figure 30: 16-foot Minion ASV based on the Marine Advanced Robotics WAM-V 16 ASV (Roddey, 2019). ....................................................................................................... 79
Figure 31: Task 1; gate entry. Requires two (2) tall buoys with reflectors ~10m apart.. 81
Figure 32: Task 2; Light tower observation. Requires one (1) light tower to start............ 82
Figure 33: Task 3; gate exit. Requires exiting the same gate as Task 1, but requires going through with the red buoy to the starboard side............................................................... 84
Figure 34: Locations of the objects and search areas for each task within the operational environment. The global search area (GSA) that would bound the task search areas is also shown. ........................................................................................................................................... 86
Figure 35: (A) Initial position for the vehicle. The initial allocation found, in order, is \( tsk_2, tsk_1 tsk_3 \). The system begins searching for \( tsk_2 \). (B) The system detects the Light Tower, the start queue for \( tsk_2 \). ................................................................. 87
Figure 36: (C) The system switches from searching for \( tsk_2 \) to head to its transit point. (D) The system arrives at the transit point for \( tsk_2 \) and begin heading for its final point. ........................................................................................................................................... 88
Figure 37: (E) The system completes \( tsk_2 \). The achieved score is now 500 points. The allocation becomes \( tsk_1, tsk_3 \). The system begins searching for \( tsk_1 \) and \( tsk_3 \). (F) The system detects the Green Tall Buoy with Reflector................................................................. 89
Figure 38: (G) The system detects the Red Tall Buoy with Reflector. (H) With both buoys detected, the system switches from searching for $tsk_1$ and instead heads to its transit point. .......................................................... 89

Figure 39: (I) $tsk_1$ transit point is achieved. The system begins to head to the final point for $tsk_1$. (J) The system completes $tsk_1$. The achieved score increases to 550 points. The allocation becomes $tsk_3$. $tsk_3$ has already been found, so the system heads to its transit point. ....................................................................................................................... 90

Figure 40: (K) The transit point for and $tsk_3$ is achieved. The system heads to the final point for $tsk_3$. (L) The system completes $tsk_3$. All task are now complete and the final score increases to 670 points, the maximum achievable. ................................................. 91

Figure 41: (A) Initial position for the vehicle. The initial allocation found, in order, is $tsk_1, tsk_2, tsk_3$. The system begins searching for $tsk_1$. (B) The system detects the Light Tower. However, it has not completed for $tsk_1$, so it does not commence $tsk_2$. ........... 92

Figure 42: (C) The Red Tall Buoy with Reflector is detected. (D) The Green Tall Buoy with Reflector is detected. Now all elements for $tsk_1$ have been detected....................... 93

Figure 43: (E) The system switches away from searching for $tsk_1$ and begins heading to its transit point. (F) The system reaches $tsk_1$’s transit point and begins heading to its final point. ......................................................................................................................... 93

Figure 44: (G) The final point for $tsk_1$ is achieved. The score increases to 50 points. The allocation is now is $tsk_2, tsk_3$. No additional searching is required as all task elements have been detected. The system heads to the transit point for $tsk_2$. (H) The transit point is achieved. The system heads for the final point for $tsk_2$.................................................. 94
Figure 45: (I) $tsk2$ final point is achieved. The achieved score increases to 550 points, and the allocation becomes $tsk3$. $tsk3$ has already been found, so the system heads to its transit point. (J) The system achieves transit point for $tsk3$. The system heads to the final point for $tsk3$. ................................................................................................................... 95

Figure 46: (K) The system completes $tsk3$. All task are now complete and the final score increases to 670 points, the maximum achievable.............................................................................................................. 95

Figure 47: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points. ........................................................... 109

Figure 48: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points. ......................................................... 110

Figure 49: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points. ......................................................... 110

Figure 50: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points. ......................................................... 111

Figure 51: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points. ......................................................... 111

Figure 52: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points. ......................................................... 112

Figure 53: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points. ......................................................... 112

Figure 54: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points. ......................................................... 113
Figure 55: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points. ....................................................... 113

Figure 56: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points. ....................................................... 114

Figure 57: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points. ....................................................... 114

Figure 58: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points. ....................................................... 115
Chapter I

Introduction

The sector of maritime robotics has seen a boom in operations in recent years. This is largely due to increased research and efforts around key maritime areas such as surveying and mapping (Kum et al., 2018; Ma et al., 2019), clean-up (Yuh et al., 2011), inspections (Bonnin-Pascual & Ortiz, 2019), and search and rescue (Dufek & Murphy, 2019) to name a few. The defense and law enforcement sectors have also seen an increase in usage of unmanned and maritime vessels for patrolling and tracking (Švec et al., 2014). This is particularly apparent in the United States of America’s Department of Defense, which recently announced its plans for integration of unmanned systems into its workflow and chain of command (Department of Defense, 2018). Following suit, the US Department of the Navy released its branch specific plans for integrating unmanned systems (Department of the Navy, 2018).

Significance of the Study

Single robot systems have been asked to accomplish an increasing number and complexity of tasks in recent years. In the maritime design domain, many efforts have been conducted to develop frameworks that allow single robots to conduct a range of tasks in an autonomous manner. The Autonomous Maritime Navigation project developed a system architecture based on the JPL Control Architecture for Robotic Agent Command and Sensing (CARACaS) engine that was capable of being deployed to a variety of maritime platforms and performed a range of tasks including patrol, interception, engagement, and sentry operations (Elkins et al., 2010). For the 2018
Maritime RobotX challenge, a number of teams developed autonomous surface vessels based on the Marine Advanced Robotics Wave Adaptive Modular Vessel 16 (WAM-V 16) to perform tasks ranging from docking, obstacle field navigation, color target identification, and acoustic recognition, to name a few (Barnes et al., n.d.; Dulle et al., n.d.; Lemanski et al., n.d.; Nieves et al., n.d.; Stanislav et al., n.d.; G. Su et al., n.d.). A survey of applications of maritime robotics systems (Yuh et al., 2011) found several instances of military robotics being tasked with a range of assignments including manipulation, patrolling, and detection activities.

Likewise, there has also been a growth in the sector of multi-robot systems that aim to leverage homogeneous and heterogeneous teams of agents to accomplish the tasks assigned to them. A survey of task allocation methods for autonomous maritime vehicle fleets noted that autonomous maritime vehicle (AMV) fleets were responsible for completing tasks in both cooperative and collaborative manners for completing single and multiple robot tasks with single and multiple task capable robots (F. Thompson & Guihen, 2019). Heterogeneous multi-agent task allocation was also examined in a survey conducted by (Rizk et al., 2019), which showed that a range of methods (Table 1) exist that seek to efficiently allocate tasks to the agents in the multi-robot system (MRS). The wide breadth of methods shown in Table 1 highlights how domain and system dependent the problem of task allocation is in single- and multi-agent systems (MAS).
Unmanned systems operating in maritime environments experience unique challenges due to the highly dynamic environment caused by typically sparse fleet distributions, inaccessibility between elements of the maritime domain (aerial, surface, and sub-surface), an unstructured and unknown environments, winds and currents, and spatial and temporal availability (F. Thompson & Guihen, 2019). This inherently means that tasks in the maritime domain tend to be very complex, requiring detailed decompositions to get them into their most primitive elements. Additionally, the need to efficiently allocate tasks to agents in such a way that performance parameters (such as lowest cost, highest points, shortest time, etc.) has been classified as an NP-Hard problem due to its comparability to issues such as the Traveling Salesman or the Knapsack problem that increase indefinitely with the number of task and agents available (MahmoudZadeh et al., 2019). This is due to the difficulty in determining the correct ordering, grouping, and assignment of tasks to agents in order to achieve the specified performance parameter. Thus, architectures that can show continued improvements to the
performance parameters and that scale well to a large number of both tasks and agents is a desirable trait of a task allocation engine. A key factor in the problem of task allocation centers around the need to quantify the cost and/or reward for undertaking a given task. This typically is caused by the difficulty in decomposing tasks as the tasks become more complex and/or dependent on multiple agents to solve them. Rizk, Awad, and Tunstel (2019) note that “as task become more complex, decision making algorithms struggle to recognize their complexity and decompose them to simpler tasks that can be solved efficiently.” Additionally, in a 2018 briefing, the United States of America’s Department of Defense (DoD) outlined their strategic roadmap for integrating and deploying unmanned systems. In this document, it was noted that four major themes would define the DoD’s challenges, advancements, and trends in unmanned systems. These themes were broadly found to be the following: interoperability, autonomy, network security, and human-machine collaboration (Department of Defense, 2018). The roadmaps for the interoperability, autonomy, and human-machine collaboration themes, Table 2, clearly show the need for a task allocation method that provides a common framework, that provides transparent and intelligent allocation of tasks, and that allows human-machine and machine-machine teaming.
Table 2: DoD unmanned system roadmaps for interoperability, autonomy, and human-machine collaboration themes (Department of Defense, 2018).

<table>
<thead>
<tr>
<th>Interoperability</th>
<th>2017</th>
<th>2029</th>
<th>2042</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Near-Term</td>
<td>Mid-Term</td>
<td>Far-Term</td>
</tr>
<tr>
<td>Common/Open</td>
<td>-Standardized C2 &amp; Reference Architectures</td>
<td>-Support Seamless, Agile, Autonomous Human-machine Collaboration and inter-machine collaboration</td>
<td></td>
</tr>
<tr>
<td>Architectures/AI Frameworks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity &amp; Parts Interchangeability</td>
<td>-Retrofit Existing Systems</td>
<td>-Plan Modularity into New Systems</td>
<td>-Rapid Upgrades and Configuration Changes</td>
</tr>
<tr>
<td>Compliance/Verification &amp; Validation</td>
<td>-New TEW Approach</td>
<td>-New V&amp;V Tools &amp; Techniques</td>
<td>-Highly Complex Autonomous Systems TEW</td>
</tr>
<tr>
<td>Data Transport Integration</td>
<td>-Common Data Repositories</td>
<td>-Integrated End-to-End Delivery</td>
<td>-Anti-Jamming</td>
</tr>
<tr>
<td>Data Rights</td>
<td>-Secure Needed Data Rights</td>
<td>-Evolve Data Rights Policy</td>
<td>-Maximum Mission Support Flexibility</td>
</tr>
<tr>
<td>Artificial Intelligence/ Machine Learning</td>
<td>-Private Sector Collaboration</td>
<td>-Augmented Reality</td>
<td>-Persistent Sensing</td>
</tr>
<tr>
<td>-Cloud Technologies</td>
<td>-Virtual Reality</td>
<td>-Highly Autonomous</td>
<td></td>
</tr>
<tr>
<td>Increased Efficiency and Effectiveness</td>
<td>-Increased Safety &amp; Efficiency</td>
<td>-Unmanned Tasks, Ops</td>
<td>-Swarming</td>
</tr>
<tr>
<td>Trust</td>
<td>-Tasking Guidance and Validation, Ethical Requirements for Human Decisions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weaponization</td>
<td>-DoD strategy Consensus</td>
<td>-Armed Wingman/Teammate (Human Decision to Engage)</td>
<td></td>
</tr>
<tr>
<td>-LAWS Assessment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human-Machine Interfaces</td>
<td>-Control Multiple Systems</td>
<td>-Human-Machine Dialog</td>
<td>-Infer Human Intent</td>
</tr>
<tr>
<td>-Human-Machine Roles/Cues</td>
<td>-“What-If” Scenario Processing</td>
<td></td>
<td>-Deep-Learning Machines</td>
</tr>
<tr>
<td>Human-Machine Teaming</td>
<td>-Load Lightening</td>
<td>-Fully Integrated Robot Teammates</td>
<td>-Reduced Warfighter Cognitive Load</td>
</tr>
<tr>
<td>-Reduce Sorties</td>
<td>-Released Maintenance Tasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-Certain Maintenance Tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Strategies</td>
<td>-Automatically Collect &amp; Process Data</td>
<td>-Deep Neural Networks</td>
<td>-Agile, Responsive, Adaptive</td>
</tr>
<tr>
<td></td>
<td>-Adjust Data Strategies Autonomously</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem Statement

The purpose of this study is to develop a method of dynamically identifying and allocating tasks based on expected task rewards. This will require developing a system for buildings task from primitive behaviors and assigning meaningful rewards for completing those tasks in the face of limited information in the dynamic environment of the maritime domain. The nature of the tasks to be accomplished will be known, but their exact location (and by extension knowledge about the order to complete them in) is unknown. Some of these tasks may also be time sensitive or may have to be completed before/after other tasks in the mission task list. Thus, it will also be necessary to develop a means to efficiently explore a defined operational area in such a way that tasks can be initiated when detected and that exploration of the domain for new task can be permitted.

The following contributions will be made through this work:

- Development and validation through case studies of an algorithm which can search for and execute tasks in a dynamic environment
- Utilizing a graph search-based task allocation to find the most efficient task ordering given a set of costs to optimize over and constraints bounding the allocation
- Evaluation of task allocation schemes to find the best scheme that optimizes a set of costs and is bounded by given constraints

Delimitations

It should be noted here that the process of effectively performing missions in a dynamic environment is dependent on a system level ability to detect, decide, and act on the information present within the environment. The 2018 Maritime RobotX competition
shows the importance of the various perception, planning, motion control, etc. systems that an autonomous system requires in order to successfully complete missions (Barnes et al., n.d.; Dulle et al., n.d.; Lemanksi et al., n.d.; Nieves et al., n.d.; Stanislas et al., n.d.; G. Su et al., n.d.). Thus, when scoping the problem of having autonomous vehicles efficiently completing tasks, it is important to scope in the portion of these capabilities under test. Thus, for this study, only the planning phase of the autonomous operation, often referred to as the mission planning stage shall be considered.

Limitations and Assumptions

Due to the scope of this study, several limitations and assumptions must be made. Some of the key assumptions made here center around the expectation of the performance of systems that interface with the mission planning process. Thus the following assumptions are made:

1. The system shall know prior to starting a mission all of the tasks that the system is expected to be able to complete during its mission
2. The system shall be able to detect and classify any and all objects that fall within its visibility horizon, as defined by Thompson, Coyle, and Brown (2019).
3. The system shall achieve all desired vehicle poses within its operational area within the time allotted if there is a path that is viable to the target position.
4. The system shall have knowledge prior to mission execution commencement of the regions within its operational domain whereby the system can be reasonably expected to find a task.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>A star</td>
</tr>
<tr>
<td>ABC</td>
<td>A* with Bounded Costs</td>
</tr>
<tr>
<td>AGV</td>
<td>Autonomous Ground Vehicle</td>
</tr>
<tr>
<td>AMV</td>
<td>Autonomous Maritime Vehicle</td>
</tr>
<tr>
<td>ASV</td>
<td>Autonomous Surface Vessel</td>
</tr>
<tr>
<td>AUV</td>
<td>Autonomous Underwater Vehicle</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent Systems</td>
</tr>
<tr>
<td>MRS</td>
<td>Multi-Robot Systems</td>
</tr>
<tr>
<td>MRTA</td>
<td>Multi-Robot Task Allocation</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>UGV</td>
<td>Unmanned Ground Vehicle</td>
</tr>
<tr>
<td>USV</td>
<td>Unmanned Surface Vessel</td>
</tr>
<tr>
<td>UUV</td>
<td>Unmanned Underwater Vehicle</td>
</tr>
<tr>
<td>WAM-V</td>
<td>Wave-Adaptive Modular Vessel</td>
</tr>
</tbody>
</table>
Chapter II

Review of the Relevant Literature

The challenge of completing complex tasks in dynamic environments by a single agent is typically overshadowed by studies on multiple agents that are attempting to solve the same issue. In fact, the field of multiple agents operating in the same domain has been studied extensively. Thus, for the purposes of this review, the literature on multi-agent task allocation and execution will be studied, with parallels to how these methods and assumptions can be made towards single agent operations.

In the multiple agent domain, the definition of the tasks and the agents is often left to the authors to decide based on the environment and types of tasks agents are attempting. Several survey papers in the areas of multi-agent systems (Dorri et al., 2018), multi-robot coordination (Yan et al., 2013), cooperative multi-agent planning (Torreño et al., 2018), and cooperative heterogeneous multi-robot systems (Rizk et al., 2019) noted several key components of multi-agent task allocation systems that includes: agent composition, agent coordination, task decomposition, and task allocation. They also spend time defining the structure and components associated with the agents and the tasks that will compose these multi-agent system’s missions. Thus, the following subsections will focus on defining and decomposing the key themes of multi-agent task allocation. This includes defining the agent, the tasks, and the task allocation methods currently available. Finally, this review will look at the limitations of these methods with the scope of the dynamic domain presented by operating within the maritime domain and will present requirements that must be met to be able to successfully complete missions within this domain.
Agent Composition

The composition of agents in a multi-agent system is typically dependent on the context of the problem presented and the types of systems being used. Rizk et al. (2019) notes that most research in the field tends to classify the type of system being presented based on how similar the agents are to each other and how the agents interact with one another.

Similarity

The first metric of agent composition is how similar the agents in the system are to one another. Early multi-agent systems tended to be built of agents that were identical in capability. These homogeneous multi-agent systems had several advantages. Multi-agent systems that had agents of only one type inherently share a common communication structure, something that heterogeneous systems cannot be assumed to have (ElGibreen & Youcef-Toumi, 2019). Thus, all agents were guaranteed to be able to communicate with each other (assuming they were within communication range) using known, standard (for that system) protocols and no agent was better than any other at completing a given task. They also had the advantage that losing any single agent in the system did not jeopardize the ability for the system to complete the tasks presented, except by running out of time, as no single agent had the sole ability to complete a given task. Thus, these types of systems had natural built-in redundancy. However, that is not to say that homogeneous systems were perfect, which is why they have fallen out of favor as of late. This was mostly due to homogeneous systems being limited in the types of missions that they could accomplish (Yan et al., 2013). Since all agents are expected to be the same, adding to the tasking capabilities of the system would mean upgrading the
capabilities of every agent in the system. This could be costly at best and impossible in some cases at worst.

A more common approach now is to have teams of heterogeneous agents. These systems may consist of some agents that are the same, but at least two of the agents must have some differing capabilities. The major disadvantage of this composition is the inherent increase in complexity of allocating tasks (Yan et al., 2013). Some of these complexities include more difficult communication and disproportionate and/or unknown capabilities (Badreldin et al., 2013; ElGibreen & Youcef-Toumi, 2019; Ramchurn et al., 2010; Rizk et al., 2019). Heterogeneous agents do not inherently come with a guarantee of being able to communicate with one another. This can be due to being in different domains (aerial, surface, underwater), having different communication protocols, and/or not knowing which systems it can communicate within its operational space (ElGibreen & Youcef-Toumi, 2019). Similarly, having agents with different capabilities means that, for some mission sets, a limited subset of the agents in the system may only be able to complete a given task. The agents in the system also may have no way of communicating their capabilities or which tasks they are working on, which can make efficient allocation of tasks even more difficult (Guoquan Wang et al., 2004; Y. Liu et al., 2013; Otte et al., 2020) However, these challenges are more than offset by the advantages gained by having a system composed of heterogeneous agents. The major advantage over homogeneous systems is that the most capable and available agent at a given time is able to be assigned to a given task (ElGibreen & Youcef-Toumi, 2019). This allows the systems to also be made cheaper by implementing task or capability specific agents rather than having a system of agents that must possess all capabilities needed to perform the
full range of possible tasks. Finally, as the agents are allowed to be heterogeneous, yet still able to accomplish the assigned tasks, any assortment of mobile robots may be used. This allows for cross-domain implementations and utilization of the robots which are available (assuming communication of some form is possible).

Thus, from these discussions, it is evident that there are several limitations to the type and range of missions that a single agent would be able to accomplish. As ElGibreen and Youcef-Toumi (2019) noted, the complexity and cost of the single agent system will increase as the range of tasks the system is expected to complete increases. However, the proper design of a task allocation scheme will allow it to easily interact with and integrate into larger multi-agent teams. This would expand the overall efficiency of the multi-agent operation if a singular agent is able to independently perform all operations available to it in the most efficient way possible.

Thus, as it is more in line with the planned integration route outlined by the Department of Defense’s integration strategy and because the general trend of modern multi-agent system research has shifted this way, the remainder of this literature review will be focused on systems of heterogeneous agents.

**Agent Definition**

Once the types of agents that are present in a system are known, the characteristics that are important in defining a single agent must be found. An agent is generally defined as any system capable of performing tasks within a given design domain. Agents tend to be defined by a few key elements including their location (Oh et al., 2017; Ramchurn et al., 2010; Rossi et al., 2015; X. Su et al., 2016; Whitbrook et al., 2019), speed (Badreldin et al., 2013; Oh et al., 2017; X. Su et al., 2016; Whitbrook et al.,
utility when attempting a given task (Badreldin et al., 2013; Y. Liu et al., 2013; X. Su et al., 2016; Tang et al., 2018; Whitbrook et al., 2019), work capabilities (generalized or for a specific task) (Badreldin et al., 2013; H. Liu et al., 2015; Oh et al., 2017; Oliver & Guerrero, 2011; Ramchurn et al., 2010; X. Su et al., 2016), current workload (ElGibreen & Youcef-Toumi, 2019; Jang et al., 2018; Oliver & Guerrero, 2011; Shi et al., 2018; Whitbrook et al., 2019), allocated tasks (H. Liu et al., 2015; Oh et al., 2017) and an identifier (ElGibreen & Youcef-Toumi, 2019; Shi et al., 2018; X. Su et al., 2016), to name a few. More recently, the impact on communications in the coalition and task allocation stages (Otte et al., 2020; X. Su et al., 2016) has also been considered, so the communication range may also be a parameter defining an agent. For this paper, a specific definition of an agent will be considered based on the work performed by Su et al. (2016) in their work on coordination and task allocation in multi-agent search and rescue challenges. Su et al. (2016) defined an agent by a six-tuple set where:

Equation 1

\[ A = \{ANo, Uti_l, Loc_l, MSp_l, Comm_l, Asta_l \} \]

Here, these features were defined such that “ANo is the ID of \( A_l \); Uti_l is the work efficiency of \( A_l \), which represents how many units of workload that \( A_l \) can perform per time unit; Loc_l is the current location of \( A_l \); MSp_l is the moving speed of \( A_l \), which represents how many units of distance that \( A_l \) can move per time unit; Comm_l is the communication range of \( A_l \), which represents the maximum units of distance that \( A_l \) can directly communicate with; and Asta_l is the status of \( A_l \), which can be either ‘available’
or ‘working’” (X. Su et al., 2016, p. 2). In this context, the work efficiency would be based on the definition of a workload as defined by Ramchurn et al. (2010), the paper Su et al. partially derived their work from. Ramchurn et al. (2010) defines a workload to be the amount of work in time units that has to be done to accomplish a given task. Thus, for most robots, this will be based on the number of tasks that can be completed at a given time unit. So, if an agent is a single-task agent, then its work efficiency would be 1 workload per unit time. This definition allows the more specific definition provided by Su et al. (2016) to be generalized out to a wider variety of task allocation problems. Thus, the terms used to define the agent are summarized in Table 3.

Table 3: Key elements defining an agent.

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANo</td>
<td>The unique ID associated with each agent, A_i</td>
</tr>
<tr>
<td>Uti</td>
<td>The workload units able to be accomplished by agent A_i per unit time</td>
</tr>
<tr>
<td>Loc</td>
<td>The POSE of the agent in ( \mathbb{R}^3 )</td>
</tr>
<tr>
<td>Spd</td>
<td>The rate of distance unit traveled per unit time of agent A_i</td>
</tr>
<tr>
<td>Asta</td>
<td>The status of agent A_i, defined as either ‘available’ or ‘working’</td>
</tr>
</tbody>
</table>

Task Decomposition

The decomposition of tasks into simpler sub-tasks or jobs is typical for complex tasks. A survey of multi-robot coordination by Yan, Jouandeau, and Cherif (2013) noted that task decomposition is a complex area of research that includes a number of techniques focused around using natural language processing techniques to decompose complex tasks into more manageable sub-tasks. However, a major issue with these methods is that their ability to successfully complete the intended objective of the task is highly dependent on the wording of the task description and the innate decomposition
capabilities of the processing system. An alternate, and typical, approach to this is to have a human expert that is familiar with the domain and the behaviors and requirements of the agent decompose the task into the primitive behaviors needed to complete a given task. Rizk, Awad, and Tunstel (2019), in a survey of cooperative heterogeneous systems, noted this taxonomy (Figure 1) was common in MRS applications across a range of implementations. This theory also aligns with the expectations of the DoD for integration of unmanned systems into the armed forces. The DoD outlined that having predictable behavior as a key enabler for unmanned systems in tight coordination and interaction with other unmanned systems and human operators (2018). Thus, the decomposition of tasks in this paper will be handled by individuals experienced with the operational capabilities of the agents and with the requirements for the tasks based on the task descriptions.

![Figure 1: Task decomposition workflow typically found in literature for MRS (Rizk et al., 2019).](image)

Another key factor in the task decomposition stage is the definition of what parameters are critical to defining a task. Similar to agents themselves, the definition of a task has found some consensus on a few key elements in the research on multi-agent task allocation problems. Some of the key elements identified thus far are a task identifier (Shi
et al., 2018; X. Su et al., 2016; Zhao et al., 2016), the workload required to complete a task (Irfan & Farooq, 2016; H. Liu et al., 2015; Oh et al., 2017; Oliver & Guerrero, 2011; Ramchurn et al., 2010; X. Su et al., 2016; Zhao et al., 2016), a reward for completing the task (Hunt et al., 2014; Lim & Choi, 2019; Oh et al., 2017; Oliver & Guerrero, 2011; Rossi et al., 2015), the cost for attempting the task (Hunt et al., 2014; H. Liu et al., 2015; Y. Liu et al., 2013; Shi et al., 2018; Zhao et al., 2016), and the status of a task such as being detected, allocated, in progress, completed, or failed (Oh et al., 2017; Shi et al., 2018; X. Su et al., 2016). If more than one agent is able or required to work on a given task, then this factor may also define a task (Jang et al., 2018; Oh et al., 2017). More recently, focus on temporally and spatially constrained tasks has come to the forefront of research. Thus, the time by which a task must be started and/or completed by (Lim & Choi, 2019; Nunes et al., 2017; Oh et al., 2017; Oliver & Guerrero, 2011; Ramchurn et al., 2010; X. Su et al., 2016; Tang et al., 2018; Whitbrook et al., 2019; Zhao et al., 2016) as well as the physical location (Lim & Choi, 2019; Y. Liu et al., 2013; Oh et al., 2017; Oliver & Guerrero, 2011; Ramchurn et al., 2010; Rossi et al., 2015; Shi et al., 2018; X. Su et al., 2016; Tang et al., 2018; Whitbrook et al., 2019; Zhao et al., 2016) of the task itself have become key factors in the task definition. Fewer still have also considered the need for some tasks to have a higher priority of execution compared to other task when other considerations (temporal, spatial, workload, etc.) are considered equal (X. Su et al., 2016). This could allow for instance, a task with a higher workload required and a high priority to be executed before a low workload and low priority task. These pieces all act as constraints on a given task that limit the point at which it can be assigned to the agent for execution. As with agents, the above elements showed up as the predominate features
considered in literature when defining a task’s structure, but they were far from being the
only elements considered.

To that end, Su et al. (2016) does an excellent job of capturing the need for
prioritizing tasks, for accounting for temporal and spatial constraints, and for handling
several other common elements in their definition of a task. They define a task, $T_{ij}$, based
on the jth task discovered by the ith agent. Thus, a task is a six-tuple set where:

$$T_{ij} = \{TNo, DL_{ij}, WL_{ij}, Loc_i, Emg_{ij}, Tsta_{ij}\}$$

First, $TNo$ is a unique task ID generated by $A_i$. In their implementation, a task’s
ID is assigned dynamically when it is identified. This means that two different agents
identifying the same task could produce task tokens with different task IDs for the same
task. This is originally handled by their proposed method by comparing the locations of
the task tokens as they filter up and removing duplicates. However, since it is assumed
here that all tasks are unique and known a priori, there is no need for unique allocation of
task IDs. Instead, this parameter will be for the unique ID defined in the mission task list.
The next element they propose, $DL_{ij}$, is the deadline by which the task will expire and
can no longer be completed. They bound this to the range of $[0, \infty)$, with 0 corresponding
to the start of the mission. If the mission has an operation time, $missionLength$, then
this range would instead be $[0, missionLength]$. $WL_{ij}$ is the number of work units
required to complete this task and the $Emg_{ij}$ term is the urgency degree of the workload,
constrained to $Emg_{ij} \in [1, 10]$. The work units can be directly correlated to the work units used in the definition of the agent’s utility parameter, $Uti$ (see Table 3). The urgency degree term, while providing a good metric for the need to prioritize two tasks of equal workload, is far too restrictive in its range of acceptable priority ranges to be used as is. This could be solved by simply opening up the range allowable by the urgency degree to that of the $[1, maxScore]$, where $maxScore$ is the maximum score or value gained by fully completing a given task. This would be able to be decided upon during the task decomposition process (and is given in the context of this paper by the competition scoring guidelines (2018 Maritime RobotX Challenge Task Descriptions and Specifications, 2018)). Finally, Su et al. define as the current status of the task. The four possible states it can have are set to be available, working, finished, or expired. Since all tasks are known a priori, an additional state, unavailable, is desirable as a means of knowing if all tasks have been discovered or not. As was stated in the assumptions for this study, a knowledge of where the task can be reasonably expected to be found within the operation domain is also required. This can be viewed as a consideration for the spatial constraints that were mentioned above. Additionally, the priority of execution for tasks presented by Su et al. can be refined down to the idea that a task having prior or direct predecessor tasks that needed to have been completed before this task could be attempted. Likewise, if a specific task was required to be completed after a given task was finished, then the follow up task could be seen as a direct successor task to the current task in question. These relation requirements can be seen as constraints on the task in question. Thus, the key elements that define a given task within a mission are defined in Table 4.
Table 4: Key elements defining a task.

<table>
<thead>
<tr>
<th>Element</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The unique ID associated with each task in the mission set</td>
</tr>
<tr>
<td>SA</td>
<td>The region whereby it is reasonably expected that the elements required to start and/or complete a given task can be found within the operational domain</td>
</tr>
<tr>
<td>Locs</td>
<td>The start position of the task in $\mathbb{R}^3$</td>
</tr>
<tr>
<td>Csts</td>
<td>The set of $m$ costs associated with the task that are to be optimized (either maximized or minimized)</td>
</tr>
<tr>
<td>Cons</td>
<td>The set of constraints placed on the task which can include, but is not limited to, elements such as required predecessors/successor tasks, time constraints, energy constraints, etc.</td>
</tr>
<tr>
<td>Status</td>
<td>The current status of the task, either: Available, Unavailable, Searching, or Found</td>
</tr>
</tbody>
</table>

Task Allocation

As was noted before, task allocation in literature tends to be focused on multi-robot task allocation (MRTA). However, ideas for how to efficiently assign tasks to a given agent in a multi-robot cluster can be applied to the single-agent task allocation problem. For background on the task allocation problem for multiple agents, it is important to quickly discuss the two main types of coordination between agents within these hierarchies. When it comes to allocating tasks to the agents in a multi-agent system, two forms of control are prevalent: centralized control and decentralized control. The former has a single agent or central system that handles formation control and task allocation about all agents in the system. The latter approach tends towards agents acting in smaller teams or completely independent from one another to accomplish the missions presented to them.
Centralized control of multi-agent systems is a typical scheme in multi-agent task allocation. This method, as noted before, has a single coordinator that takes in data about the environment, determines which tasks should be allocated, and handles allocation of the tasks to all agents in the network (Rizk et al., 2019; Torreño et al., 2018). Centralized multi-agent systems are characterized by high degrees of communication and collaboration between agents in the system (Yan et al., 2013). The central coordinator is made aware of all information regarding the current states of every agent and of all data collected about the operating environment. Typical task allocation strategies for centralized systems include auction (Oliver & Guerrero, 2011) and market-based strategies (Badreldin et al., 2013; ElGibreen & Youcef-Toumi, 2019) and swarm-based allocation schemes (Oh et al., 2017; Oliver & Guerrero, 2011).

The other typical scheme for multi-agent systems is a decentralized control scheme. As noted before, this scheme focuses on smaller, independent coordination of agents to accomplish tasks. Yan et al. (2013) further breaks down decentralized approaches into two forms, distributed and hierarchal. In the distributed approach, task allocation can be allowed to be done on an individual agent-by-agent basis. Hierarchal task allocation, on the other hand, is a hybrid of the centralized and distributed structures whereby agents organize themselves into small teams with local coordinators that handle task allocation to the agents in their coalition. Distributed decision structures tend to occur when communication between agents is generally or always non-existent. In flat decision structures, agents are responsible for assigning themselves tasks based solely on the information that they have available and based on assumptions about what the other agents in the system will be doing. Task allocation schemes that include flat,
decentralized schemes include methods such as partially observable Markov-decision process (Capitan et al., 2013; Omidshafiei et al., 2017), game-theory approaches based on contract net proposals (Cui et al., 2013; Guoquan Wang et al., 2004; Lim & Choi, 2019), negotiation-based algorithms (ElGibreen & Youcef-Toumi, 2019; Rossi et al., 2015), and some auction and market-based approaches (Rizk et al., 2019), and other novel approaches (Jang et al., 2018; H. Liu et al., 2015). The other approach, a hierarchical organization of agents, takes a more middle ground approach, often straddling the line between centralized and decentralized schemes. This method sees small coalitions of agents with a central coordinator formed to accomplished tasks (Yan et al., 2013). The method maintains the advantages of the centralized coordination structure on a local scale. Hierarchal coordination allows agents to locally allocate tasks in an optimal manner based on the agent’s utility function (Tang et al., 2018). Typical methods for allocating tasks in hierarchal systems are auction-based (Binetti et al., 2013; Irfan & Farooq, 2016; Otte et al., 2020; Shi et al., 2018; Tang et al., 2018) and market-based strategies (Han-Lim Choi et al., 2009; Hunt et al., 2014; Y. Liu et al., 2013; Oh et al., 2017; Ramchurn et al., 2010; X. Su et al., 2016; Whitbrook et al., 2019; Zhao et al., 2016). Since hierarchal systems tend to provide the best parts of the centralized and decentralized coordination techniques with minimal costs overall, the remainder of this section will be focused on techniques that take advantage of hierarchal decentralized agent coordination.

With this background on the types of coordination between multi-agent systems in mind, it becomes possible to ask the question of what kinds of requirements are
required to be satisfied for a single agent to be able to find and allocate tasks in a dynamic environment. Some of these key requirements are as follows:

- Must be capable of searching for tasks within a pre-described region
- Must be able to determine if a task is available
- Must be able to estimate any costs and constraints associated with the task and handle dynamically linked tasks
  - Could be temporal, spatial, or other constraints associated with each task
  - Could also include tasks ordering constraints, time constraints, etc.
- Must be able to find the optimal ordering of tasks to execute based on one or more optimization features, possibly including, but not limited to: score, time, energy consumption, distance traveled, etc.

In the related field of Mobile Crowdsensing Systems (MCS), several of these requirements are attempted to be solved to combat mobile sensing problems. This includes solutions that provide for single-task and multi-task allocation solutions for multiple agents. One example of this is spacial crowdsensing which considers tasks with spatial and temporal constraints as well as costs such as the scale and difficulty of the task that may dictate the willingness of an agent to accept a task for allocation (Guo et al., 2018). A survey of these allocation schemes also noted that crowd sensing has some uses for autonomous vehicle task allocation problems (Capponi et al., 2019). However, this has limitations in the field of single agent task allocation. First and foremost, the methods do not tend to focus on applications with few agents and many tasks for each agent. Most importantly, they all lack true multi-cost optimization. Most only focus on temporal, spatial, and energy costs with possibly a few others considered. Additionally, they do not
allow for allocations that may not meet all of the constraints placed on the system if the allocation is over-constrained. However, they do consider the above requirements to be important considerations for the task allocation in MCS problems that still need to be solved (Capponi et al., 2019; Guo et al., 2018).

Several task allocation schemes also attempt to satisfy all or part of these requirements when formulating their task allocation algorithms for multi-robot problems. A very similar situation to the dynamic environment described above is search and rescue-based operations. Ramchurn et al. (2010), in their highly cited work, utilized a decentralized auction-based approach to assign tasks in coalitions of agents that were performing search and rescue tasks for the robocup rescue challenge. Their approach had agents form coalitions that could be created and disbanded dynamically as agents entered and left the rescue scene to handle the tasks identified by the agents in the coalition. In the coalition, the agents would utilize a Fast-Max Sum algorithm to converge on the optimal allocation of tasks to the agent in the coalition. However, it was noted that, to achieve this locally optimal allocation, required an increasing amount of communication between agents as the number of agents in the coalition grew and the complexity of the communications increased (X. Su et al., 2016). This ultimately made their approach difficult to expand out to dynamic domains where tasks and agents may be changing frequently. While this handled the issue of dynamic availability of tasks and searching of tasks, it lacked the ability to optimize the allocation found across a range of constraints. More recently, an auction-based strategy was used to allocate single tasks which require multiple robots to complete them, so called single-task, multi-robot (ST-MR) problems (Irfan & Farooq, 2016). As is typical, this work sought to form coalitions of agents that
had the best utility for completing a single task that would be allocated to the coalition leader. Thus, the primary goal of this work was to find the best coalition of agents to complete a given task rather than finding the best distribution of tasks amongst a coalition of agents. Irfan and Farooq sought to have the coalition auctioneers put their agents up for bid to other coalitions and took in other coalition’s agents as bids to determine if the utility provided by adding, removing, or exchanging agents would improve the coalition’s ability to complete a task as fast as possible and with the least number of agents required. This paved the way towards allocations that were able to handle varied constraints for each task, such as requiring multiple agents for completion, to be accomplished. However, this method did suffer from several issues, not least of which was a dynamic environment requiring a higher degree of communication between agents that their proposed method was not well equipped to handle.

However, the proposed algorithm requires a high degree of communication between agents and their coalition leader and between coalition leaders. This ultimately would increase the number and complexity of required communications as the number of agents and tasks increased. Another rescue environment task allocation approach was performed by Shi et al. (2018). Their work utilized an auction-based approach to maximize the value gained by having the best agent that can complete a task and to minimize the costs associated with that agent working on the task. As in other auction-based approaches, the method presented sought to match tasks one-to-one with the available agents. If the number of agents and tasks were equal, then the best paring of tasks to agents was done which maximized the coalition’s utility and minimized the cost of performing each task. If there were more tasks than agents, or vice-versa, than virtual
agents or tasks, respectively, would be generated and allocated to the agents/tasks that provided the worst utility and the highest cost to complete to the system. As agents would be freed up or tasks disappeared, these virtual tasks and agents would be removed. While this method worked well for the dynamic environments often seen in disaster rescue environments, having a single agent to perform the tasks available would mean that the system would simply be selecting and completing the highest utility task at any given point, possibly leading to an inefficient usage of time or resources if a task became available that the agent would be better off performing instead. Han-Lim Choi et al. (2009) presented one of the earliest versions of the consensus-based approach. Their consensus-based auction algorithm (CBAA) and consensus-based bundle algorithm (CBBA) worked to assign a single task to a single agent and bundles of tasks to a single agent, respectively. The latter of the two, the CBBA approach, was most akin to the problem presented here. In this paper, the authors selected a consensus-based approach to mitigate the issues associated with communications and connectivity in auction-based approaches. The key development of their work, CBBA, uses two phases, a bundle construction phase and a conflict resolution phase, to allow the system to converge on how to bundle tasks and on which bundles to assign to each agent by trading the bundles to achieve the highest group utility. This approach allowed for efficient bundling and assignment of multiple tasks to each agent in a multi-agent scheme. However, the major limitation of this allocation in the single agent context is the bundling process. The bundles here were formed by maximizing a global reward. This global reward was based around a score function that was tuned based on things like the path length, mission completion time, and value of the task. However, the issue with this is that it required
Thus, optimizations of the reward that, say, maximize the value gained for completing the task, while also minimizing the time to complete and the energy required natively, and without tuning, is not possible. While other algorithms (Hunt et al., 2014; Oh et al., 2017; Whitbrook et al., 2019) expanded on and improved this approach, these limitations in task allocation persisted.

Thus, this paper seeks to view the problem in a novel way compared to the previous paper’s views of task allocation. By focusing on a singular agent that has to search for or has found one or more tasks in its operational domain, the system should be able to independently ascertain the order to search for and complete the tasks in the current mission set by optimizing the reward gained. This reward would not need a single parameter that encompasses all of the costs under consideration to be refined or tuned. Instead, the following method is proposed that address these issues by viewing the task allocation problem as a graph search problem where the “path” through the tasks can be optimized to achieve the best allocation possible.
Chapter III

Methodology

In the previous sections, several key objectives for a task allocation scheme were outlined. These included the following:

- The ability to handling any constraints on the agent
- The ability to determine if a task is available
- The ability to estimate the costs for each task
- The ability to handling any constraints on a task
- The ability to find the optimal ordering of tasks for execution over multiple costs and constraints
- The ability to transit to a task
- The ability to execute a given task (including searching for said task)
- The ability to exit a task should it take too long to complete

In order to meet these objectives for task allocation, the following methodology, henceforth known as Minion Task, was proposed. Minion Task sought to provide an algorithm capable of dynamically assigning and executing tasks in a partially defined environment by efficiently searching for tasks and optimally allocating said tasks to a task execution algorithm. This approach also made the following assumptions for its operation:

- All tasks, as well as the operations required by the vehicle to complete these tasks, were known prior to the mission being commenced
• The general region where each task is to be searched for was known. This could be as broad as the entirety of the predicted operational space or a specific subset of this space.

• All tasks were expected to be decomposed into primitive actions to be completed by the system in order to complete said task.

To address the objectives for task allocation subject to these assumptions, Minion Task utilized variable search spaces, a task allocation scheme based on the A* with Bounded Costs (Logan & Alechina, 1998) graph search algorithm, and a dynamic task execution engine to accomplish a mission. The following sub-sections will outline the general definition of an agent and a task used by Minion Task, handling of the variable search space, the task evaluator, the task allocator, and the task execution engine used by this algorithm.

**Agent Definition**

The Agent has its definition drawn from common characteristics in the multi-agent domain described previously in Table 3. In addition to the key components listed previously, the area the agent is allowed to search is another key component that must be considered when defining the agent. Thus, the agent in general is defined as shown in Equation 3:

\[
A = \{\text{Loc}_{V_{eh}}, \text{Spd}, \text{Cons}_{V_{eh}}, P_{FRD}\}
\]
Whereby $Loc_{veh}$ is the instantaneous global pose estimate of the vehicle (in the same frame as the objects that will be used in tasking operations). This includes the position and heading of the vehicle in $\mathbb{R}^2$ space. It should be noted here that the position of the vehicle as well as all other parameters presented in this chapter assume an $\mathbb{R}^2$ spatial reference, nothing in this method inherently prevents it from being expanded into $\mathbb{R}^3$ space. $Spd$ is the instantaneous speed of the vehicle, $Cons_{veh}$ is the set of $J$ values used as constraints for adding this task in the task allocation stage such that $Cons_{veh} = \{con_{veh,1}, \ldots, con_{veh,J}\}$ and $con_{veh,j}$ is the $j^{th}$ constraint that the vehicle places on the task allocation. These constraints could include time constraints, energy usage constraints, path length constraints, etc. $P_{FRD}$ represents the visibility region around the vessel representing the vehicle’s exteroceptive range. This is similar to the region defined by Thompson, Coyle, & Brown (2019). They define the visibility region as the region around the vehicle where sufficient sensor returns occur such that the perception system were trusted to detect and classify objects of interest. A sample visibility region used in this same work is shown in Figure 2.
Figure 2: Visibility region, $P_{FRD}$, used for these scenarios. This region is based on the Minion ASV’s perception region with the vehicle centered at Northing: 0 m; Easting: 0 m in this FRD frame (Barnes et al., n.d.; D. Thompson et al., 2019).

It should be noted that the visibility region for the vehicle can be any number of polygons required to define the region covered by the agent’s perception systems at a given point. Additionally, the visibility region can be dynamic or static region(s). With this knowledge in hand, the general process for executing a given mission configuration can be examined.

**Task Definition**

The definition of a task takes utilizes the key elements that comprise a task that were specified previously in Table 4. The environment is known to have N possible tasks to complete. This set of tasks is defined by $\mathcal{T} = \{tsk_1, \ldots, tsk_N\}$. Here, $tsk_n$ is the $n^{th}$ task in the set such that $1 \leq n \leq N$. Additionally, each task can be further defined as shown in Equation 4:
Equation 4

\[ tsk_n = \{ID, SA, Loc_S, Csts, Status, Cons_{tsk}\} \]

In Equation 4, \( ID \) is the unique ID assigned to each task. \( SA \) is the Cartesian search area in \( \mathbb{R}^2 \) space that the vehicle should search and be reasonably expected to find the task in. \( Loc_S \) is the predicted start location for the task in \( \mathbb{R}^2 \) space that is updated by the Task Ready Check (defined later in Task Ready Check Evaluation). \( Csts \) is the set of M costs associated with the task that are to be optimized (either maximized or minimized) such that \( Csts = \{cst_1, \ldots, cst_M\} \). These costs could include elements such as the score for completing the task, the time required to execute the task, the energy required to perform the task, etc. \( Status \), is used to inform the system of the current state of the task. Each task has one of four states that it can be in: Unavailable, Available, Searching, Found. If the status of the task is Unavailable, then the task was either completed or timed out during execution and should no longer be considered for allocation. If the status of the task is instead Available, then the vehicle can attempt to either search for or execute this task. Should \( Status \) be set to Searching, then the vehicle would be expected to transit to the next point where it may be able to find the elements required to start the task. Finally, if \( Status \) goes to Found, then the task is ready to be executed; otherwise, the task should still be searched for. The last parameter each task has is a set of K constraints, \( Cons_{tsk} = \{con_{tsk,1}, \ldots, con_{tsk,K}\} \), that could include elements such as required predecessors/successor tasks, time constraints, energy constraints, etc.
As a short aside, the ability to define required execution of predecessor and/or successor tasks for a task is a critical feature of this method. This allows for specific ordering of tasks to be accomplished should a mission demand it. In the case of prior predecessors, this could include tasks that may provide information that increase the likelihood of success of completion. This might be a task that required searching for a specific indicator at some point before attempting this task so that the information required to complete this task would be known. A direct predecessor may be required if a task must be completed directly after another or not attempted at all. An example of this may be that an item can only be identified if it is immediately searched for after passing through a barrier. If the barrier is left in order to do another task, then the identification may not be attempted again. In a similar vein, requiring a direct successor to a task may be required if an operation must be followed up another in order for this task to be attempted. An example of this may be a deployment task requiring a recovery task immediately after it.

This information, together, helps to scope each task with respect to the others in the task set, TASKS. With this information in mind, it is possible to define the critical processes the Minion Task algorithm requires in order to complete a given mission.

Process

Minion Task has four key components that are run asynchronously as the system performs its mission. The four elements of Minion Task are as follows: search space updater, task evaluator, task allocator, and task executor. The high-level task allocation process is outlined in Figure 3.
As is shown above in Figure 3, Minion Task is a continuous process that runs until it is forced to stop or the system has completed all assigned tasks. After the mission has been loaded and the checks shown in the first part of Figure 3 have been passed, the system moves into the 4 asynchronous process that are required to complete a given mission set. The first of these asynchronous processes that is shown is the Search Space Updater.

The Search Space Updater receives information about the vehicle outlined by the agent definition in Equation 3. Using this information, the Search Space Updater aims to understand where the vehicle is likely to have effectively detected objects within the vehicle’s visibility horizon, $P_{FRD}$. The output of this process is a Global Search Area. This represents the region within the operational domain where the vehicle has not previously tried to perceive objects.
The Global Search Area is continually updated and passed from the Search Space Updater to the next part of the process, the Task Evaluator. The Task Evaluator determines which tasks are available for execution based on the items currently perceived by the system and the operational domain that has been covered by the visibility horizon thus far. For tasks that are not ready to be executed, it also determines where to search for the task ready-check elements within the Global Search Area. The Task Evaluator then outputs the list of tasks that need to be ordered for allocation, i.e. the Task List, to the Task Allocator. This task list contains all of the tasks that have not been accomplished or timed out during execution. Thus, information about where to go to execute or find a given task is then passed on to the Task Allocator.

The Task Allocator is responsible for finding the best allocation of the tasks subject to the task constraints, $Cons_{task}$. Utilizing a modified version of the A* with Bounded Costs (ABC) algorithm, the Task Allocator generates an ordering of the tasks in the Task List for completion based on the multi-variable and multi-constraint optimization process that the ABC algorithm performs. This generates a Task Allocation list, which specifies the order of completion of all or part of the tasks in the Task List, and a Task Costs term that denotes the value of the allocation presented with respect to the constraints on the system that is able to be passed on to the Task Executor.

Finally, the Task Executor continually receives the Task Allocation and the Task Costs from the Task Allocator. The Task Executor then determines, based on this allocation, which task the vehicle should attempt and works to execute the current tasks in the task allocation is has deemed to be acceptable. How each of these four elements operates is identified in following subsections.
**Search Space Updater**

As discussed in Chapter II, the tasks required to be accomplished in each mission, as well as all agents that can accomplish those missions are known prior to mission start. What is not known prior are the exact locations of each task within the operational domain. However, as was specified in the assumptions for Minion Task, the search region(s) for each task are known ahead of time. Thus, during most parts of a mission, an agent will not be allocated to a given task, but to instead search for a specific task. So, while an agent is not executing a task, it shall instead be searching the available mission space for the tasks in the mission list that have not previously been attempted. Thus, efficiently searching for task ready check elements is key to efficient mission execution. It is the Search Space Updater that is responsible for determining what portion of the search space has been perceived by the agent. When the mission is initialized, the search area for all tasks, $\text{tsk}_{n,SA}$, in $\text{TASKS}$ are evaluated to find the bounding box around all task’s search areas. This bounding box was then used to define the Global Search Area, $\text{GSA}$, that encompasses all of the individual task search areas. An example of this is shown in Figure 4.
Figure 4: (Left) Individual task search regions, \textit{task}.\textit{SA}, and mission objects for this configuration. (Right) Global search area, \textit{GSA}, bounding all task search regions.

Once the mission has started, this vehicle’s visibility horizon, \(P_{FRD}\), is used to update \(GSA\). This happens by transforming \(P_{FRD}\) into the global frame using position and heading information within \(Loc_{Veh}\) and subtracting the transformed \(P_{FRD}\) region from \(GSA\) region, as shown by Equation 5.

\textbf{Equation 5}

\[
P_{NED} = \left(NED_{FRD}\right) T * P_{FRD}
\]

\[
GSA = GSA - P_{NED}
\]

Thus, as the vehicle transits the operational domain in search of task elements, \(GSA\) is gradually reduced in area, indicating more and more of the space has been effectively searched. This process is portrayed in Figure 5.
This reduced global search area is then used by each task’s Search Ready Check to determine the optimal location to continue searching for the given task. The process for checking for a task’s ready status or where to search for a task next is handled by the Task Evaluator, which is described in the following section.

**Task Evaluator**

The Task Evaluator process receives the global search area, $GSA$, from the Search Space Updater and uses this information as well as the vehicle’s current location, $Loc_{veh}$, in order to determine which tasks are available to be searched or executed. This process returns a Task List, $TL$, that is a subset of $TASKS$, as shown in Equation 6.
Equation 6

\[ TL \subseteq TASKS \]

With this information in mind, the algorithm used to evaluate the subset of tasks to include in the task list is detailed in Table 5.

Table 5: Task evaluation algorithm.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSA, Loc\textsubscript{Veh}, TASKS</td>
<td>TL</td>
</tr>
</tbody>
</table>

1: \( TL = \text{empty} \)
2: \( IF \ (\text{anyTaskAvailable}(TASKS)) \)
3: \( FOR \ EACH \ tsk \ in \ TASKS \)
4: \( IF \ (\text{tsk}_{\text{status}} \text{ is not Unavailable}) \)
5: \( \text{tsk}_{\text{TaskReadyCheck}}(GSA, Loc) \)
6: \( IF \ (\text{tsk}_{\text{status}} \text{ is Found}) \)
7: \( \text{Add tsk to TL} \)
8: \( ELSE \)
9: \( \text{tsk}_{\text{SearchReadyCheck}}(GSA, Loc) \)
10: \( IF \ (\text{tsk}_{\text{status}} \text{ is Searching}) \)
11: \( \text{Add tsk to TL} \)
12: \( \text{Return TL} \)

In the above algorithm, the anyTaskAvailable() function returns true if there are any tasks that have whose Status is not Unavailable; otherwise, it returns false. The status for each task, \( \text{tsk}_{n,\text{status}} \), in TASKS is initialized to Available when the mission is started. The TaskReadyCheck() and SearchReadyCheck() are used to determine if the task has been detected, and if not, where the vehicle should consider searching for the task if it still has regions where it believes the task may be found.
Task Ready Check Evaluation

Each task, $tsk_n$, is given a task ready check operation, $tsk_{n,TaskReadyCheck}$, that is responsible for taking in any parameters it needs from the system to determine if the required elements for the task have been identified. If the elements required to start the task have been identified, then $tsk_{n,Status}$ is set to Found. This could include if a prior predecessor had been completed, if a specific object had been identified, if a time threshold had been passed, etc. Additionally, $tsk_{n,LocS}$ is set to the location where the task is to be started. While the activation condition has not been met, $tsk_{n,Status}$ and $tsk_{n,LocS}$ are not changed from their previous values. If the task has not been detected, then the Search Ready Check is additionally performed to determine where to consider looking for this task next.

Search Ready Check Evaluation

The Search Ready Check is responsible for returning the desired search waypoint for a task while the conditions needed to start the task have not yet been met. In this function, the desired search waypoint is calculated by first finding the region of overlap between the global search area, $GSA$, and the task’s search area, $tsk_{n,SA}$. The result of this is the overlapping search region(s), $OSA$, as shown in Equation 7:

Equation 7

$$OSA = GSA \cap tsk_{n,SA}$$
If OSA is empty, indicating that the entirety of the expected search area has been covered, then $tsk_{n,\text{Status}}$ and $tsk_{n,\text{LocS}}$ are not changed from their previous values. Otherwise, the closest point on each polygon in OSA to the agent is calculated with respect to the current vehicle position, $\text{Loc}_{veh}$. Among these, the closest of these points is then selected as the next search point for the system for this task. Finally, then $tsk_{n,\text{Status}}$ is set to Searching and $tsk_{n,\text{LocS}}$ becomes the target waypoint. A sample of the above process can be seen in Figure 6.

![Figure 6: (Left) Global Search Area, $GSA$, and Task 1 Search Area, $tsk_{1,SA}$, prior to intersection operation. (Right) $GSA \cap tsk_{1,SA}$ representation and predicted target search waypoint for the given vehicle position, $\text{Loc}_{veh}$.](image)

With all tasks evaluated, the Task List is then sent to the Task Allocator so that the available tasks may be sorted for search or execution order.
**Task Allocator**

As was described before, the task allocation phase can be viewed as a graph search problem with bounded costs constraining the ordering of the tasks. Thus, upon receipt of the task list, $T_L$, from the Task Evaluator, the system determines the order of task execution. This is accomplished by utilizing a variation of A* with Bounded Costs, or ABC, (Logan & Alechina, 1998), which allows for various hard- and soft-constraints to be considered. This section will define how the Task Allocator uses ABC to select the order in which tasks should be executed for optimal behavior. An example of the task allocation in a graph search format is in Figure 7. This configuration will be utilized in the process of explaining the ABC algorithm in the context of task allocation in subsequent figures with the costs to be optimized here being the score and time costs.

![Figure 7: Sample task configuration visualized as a connectivity graph where the S node is the starting node and the remaining nodes are the task nodes.](image-url)

---

**Task Allocator**

As was described before, the task allocation phase can be viewed as a graph search problem with bounded costs constraining the ordering of the tasks. Thus, upon receipt of the task list, $T_L$, from the Task Evaluator, the system determines the order of task execution. This is accomplished by utilizing a variation of A* with Bounded Costs, or ABC, (Logan & Alechina, 1998), which allows for various hard- and soft-constraints to be considered. This section will define how the Task Allocator uses ABC to select the order in which tasks should be executed for optimal behavior. An example of the task allocation in a graph search format is in Figure 7. This configuration will be utilized in the process of explaining the ABC algorithm in the context of task allocation in subsequent figures with the costs to be optimized here being the score and time costs.
The task allocation is handled by formatting the tasks as nodes in a connectivity graph and then finding the optimal path that meets the constraints imposed by the task list, $TL$, via the ABC algorithm (Logan & Alechina, 1998). ABC is a generalization of the A* algorithm that allows for the system to find the optimal path, in this context the path that provides the task allocation order, that satisfies the set of constraints imposed on the tasks rather than needing to formulate a single criterion to minimize or maximize. This is achieved by allowing for a state space search that prioritizes soft constraints on the ordering. These soft constraints are constraints that can be violated. However, if a configuration can be generated that does not violate a soft constraint, then this configuration would be prioritized over a configuration that violated this same constraint. This allows for a simple means of ordering allocations based on when constraints are and are not violated as well as ordering of the priority of violations that are allowed to occur. To formalize this, each proposed allocation is assigned to an equivalence class that represents the constraints that are satisfied. The equivalence classes are then able to be ordered based on what the operator defines as being more critical for the mission in question. An example of this might be prioritizing meeting energy usage constraints over meeting path length constraints. ABC also presents a method for eliminating redundant proposals from the set of available proposals. This is handled by a special case of pointwise ordering that the paper notes as path domination. It is noted that “One path $p_a$ dominates another path $p_b$ if both paths terminate in the same state, and there is at least one cost $f_i$ such that $f_i(p_a) < f_i(p_b)$ and there is no cost $f_j$ such that $f_j(p_a) > f_j(p_b)$” (Logan & Alechina, 1998, p. 6). If this condition is met, then the dominated path is removed from the set that contains it. A sample problem based on the configuration
shown in Figure 7 is used to illustrate this algorithm. For this example, a single
equivalence class is utilized. In this class, a single constraint (available time) is present,
and the class seeks to optimize the following costs: maximize the score achieved,
minimize the time required. With this in mind, the ABC algorithm is reproduced in Table
6 and the sample problem presented in Figure 7 is worked through using the ABC
algorithm in Figure 8 through Figure 14.


<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>TL, tsk\textsubscript{Start}</td>
<td>TaskAllocation, TaskCosts</td>
</tr>
</tbody>
</table>

1 : \textit{OPEN} = \texttt{[Start]}
2 : \textit{CLOSED} = \texttt{empty}
3 : \texttt{Repeat}
4 : \texttt{IF OPEN is empty}
5 : \texttt{Return false}
6 : \texttt{Remove n, the least member of the first non-empty equivalence class, from OPEN and place it on CLOSED}
7 : \texttt{IF n is a solution}
8 : \texttt{Return [TaskAllocation, TaskCosts] = n}
9 : \texttt{FOR EACH successor, n', of n}
10 : \texttt{Cost n' and determine its equivalence class}
11 : \texttt{Remove from OPEN and CLOSED all paths dominated by n'}
12 : \texttt{IF n' is dominated by any path on OPEN or CLOSED}
13 : \texttt{Discard n'}
14 : \texttt{ELSE}
15 : \texttt{Add n' to OPEN, in order}
Figure 8: Initialization of ABC for the sample task allocation problem with the start, $t_{task}^{start}$, added to the open set. For this problem, task 2 is the goal task and a maximum allowable allocation time constraint is 450 seconds.

<table>
<thead>
<tr>
<th>Open Set</th>
<th>Score Cost</th>
<th>Time Cost</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Closed Set</th>
<th>Score Cost</th>
<th>Time Cost</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
Figure 10: Successors, n’, are generated for the element, n, under consideration. The successors are costed and then checked for domination. In this iteration, no domination occurs; however, a case where domination occurs is presented later in the evaluation process.

Figure 11: All successors, n’, not dominated are added to the open set. The open set is then sorted based on the equivalence class sorting method utilized. Here, a single equivalence class is used, so the elements are simply sorted within that class.
Figure 12: Example iteration where a domination occurs. Nothing in the open set is dominated by the successor, n’. While n’ does have a score cost that is the same or better than the elements in the open set, the time cost for n’ is worse than ever element in the open set. So, the conditions for domination of n’ over the open set elements is not met. However, the successor, n’, is instead dominated by an element on the open set. Since an element on the open (circled in red) set has the same score cost as the successor and has a lower time cost, it dominates the circled successor. Thus, this successor is removed.
Figure 13: If no successors remain after checks for domination occur, then no elements are added to the open set. The open set is ordered as before.

**Open Set**

<table>
<thead>
<tr>
<th>Score Cost</th>
<th>Time Cost</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>366.84</td>
<td>[3,1,2]</td>
</tr>
<tr>
<td>125</td>
<td>316.38</td>
<td>[3,3,2]</td>
</tr>
<tr>
<td>95</td>
<td>260.48</td>
<td>[3,1,2]</td>
</tr>
<tr>
<td>75</td>
<td>210.05</td>
<td>[3,2]</td>
</tr>
</tbody>
</table>

**Closed Set**

<table>
<thead>
<tr>
<th>Score Cost</th>
<th>Time Cost</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[3]</td>
</tr>
<tr>
<td>145</td>
<td>60.48</td>
<td>[3,1]</td>
</tr>
<tr>
<td>145</td>
<td>118.08</td>
<td>[3,3,2]</td>
</tr>
<tr>
<td>145</td>
<td>166.74</td>
<td>[3,3,1]</td>
</tr>
<tr>
<td>145</td>
<td>271.34</td>
<td>[3,2,1]</td>
</tr>
</tbody>
</table>

Figure 14: When an element that has the goal state as the final point in the task allocation “path”, then the goal is considered to be reached and the process is finished. Should the open set become empty and the goal had not been reached, then the no viable allocation would be available.
It should also be noted that the generation of successors is a key area of this algorithm which allows for constraints on the tasks, $tsk_{k,cons}$, to be considered. An example of how successor generation can be configured to handle constraints is outlined in the algorithm in Table 7 (here prior predecessor, direct predecessor, and direct successor requirements are considered). While this is but one set of constraints that could be placed on the tasks, it is important to consider that this could be configured for any set of constraints on the system.
Table 7: Sample successor generation algorithm which handles direct/indirect predecessor and direct successor requirements for the task ordering.

<table>
<thead>
<tr>
<th>Input</th>
<th>TaskAllocation, TaskCosts, TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>successors</td>
</tr>
</tbody>
</table>

1: successors = empty  
2: Get the end task, tsk_final in the TaskAllocation  
3: IF tsk_final has a direct successor, DSucc  
4: IF tsk_final, DSucc is not in the TaskAllocation  
5: addSuccToSet = true  
6: IF tsk_final, DSucc has prior predecessors, PPred  
7: IF all tsk_final, DSucc, PPred are not in TaskAllocation  
8: addSuccToSet = false  
9: IF tsk_final, DSucc has direct predecessor, DPred  
10: IF tsk_final, DSucc, DPred is not tsk_final  
11: addSuccToSet = false  
12: IF addSuccToSet is true  
13: successors = tsk_final, DSucc  
14: Return successors  
15: FOR EACH tsk in TL  
16: addTaskToSet = true  
17: IF tsk is not in TaskAllocation  
18: IF tsk has prior predecessors, PPred  
19: IF all tsk, PPred are not in TaskAllocation  
20: addTaskToSet = false  
21: IF tsk has direct predecessor, DPred  
22: IF tsk, DPred is not tsk_final  
23: addTaskToSet = false  
24: ELSE  
25: addTaskToSet = false  
26: IF addTaskToSet is true  
27: Add tsk to successors  
28: Return successors

The result of the ABC algorithm is an ordering of tasks, TaskAllocation, that is a subset of TL such that TaskAllocation \( \subseteq TL \). Additionally, the costs associated with
attempting this allocation, \textit{TaskCosts}, are also output from this process. By utilizing the modified A* with Bounded Costs algorithm to find the best ordering of tasks for execution, the system is effectively able to optimize the completion of tasks based on the constraints that make most sense in the context of the current mission. More importantly, it also allows the system to consider complex task interconnectivity by considering the direct and indirect predecessors or the direct successors that that task may have to consider. Once the ABC algorithm has been run, a Task Allocation and Task Cost for the currently available tasks has been generated. This is then passed on to the Task Executor that will determine if this new allocation should be utilized.

\textit{Task Executor}

The Task Executor receives the Task Allocation and Task Costs set from the Task Allocator and determines if a new allocation order should be implemented. The Task Executor operates in one of three modes at any given instance. These modes are as follows: waiting, transit, and task. By transitioning through these three modes, the algorithm finds, approaches, and execute tasks in the currently selected Task Allocation by continuously transmitting a target waypoint. Upon initialization, the Task Executor defaults its operational state to waiting in anticipation of the first Task Allocation being received. From this point on, the Task Executor follows the process outlined in Figure 15.
Figure 15: Task Executor general process flow. Phase 1 (red) shows the processes for updating the current Task Allocation and Task Costs. Phase 2 (blue), 3 (yellow), and 4 (green) show the operations that occur based on the mode the executor is in currently.

Figure 15 is divided into four phases of operation, the last three of which are accessed depending on which mode the executor is in at that instance. In phase one (red), the system checks to see if a new Task Allocation and Task Costs have been received. If they have not, then the system continues executing its previous mode. If new data is received from the Task Allocator, then the executor compares the old and new allocations based on cost. Rather than updating the task allocation and cost every time an allocation with better costs is received, the system requires new Task Costs to meet a tunable minimum threshold of improvement before switching. This prevents the allocation from switching every time a marginally better allocation is received. Finally, if the Task Allocation and Task Costs are updated, this phase sets the current mode to transit and stops any currently running tasks. The task that is terminated in order to begin the new
allocation is not flagged as completed or failed and thus is still available for allocation by the Task Evaluator and Task Allocator in future iterations.

Phase two through four show the different processes carried out for each of the modes the executor can be in. Phase 2 (cyan) occurs when the vehicle is in the waiting mode and is intended to keep the vehicle stationary while it waits for a task to be assigned for execution. If the vehicle is in the transit mode, then it will perform the operations in Phase 3 (yellow). This phase has the vehicle transiting to the start point, $t_{\text{start}}$, for the current task in the Task Allocation until that point is achieved. If the $t_{\text{status}}$ is Searching, then this would be the next target waypoint in the search. If $t_{\text{status}}$ is Found, then the vehicle would transit to the start point for the current task.

Finally, in phase 4 (green), the steps needed to complete the task are implemented. If the task has not been completed and not timed out, then the processes outlined in the task’s Task Execution operation are executed. For $t_{\text{status}}$ being set to Searching, this would immediately complete as the target waypoint has been reached. Should $t_{\text{status}}$ be set to Found, then the task would execute the primitive behavior that it was programmed with prior to the mission being started in order to complete this task. If the task is completed or times out, the task is set to be no longer available, $t_{\text{status}} = \text{Unavailable}$, and the vehicle returns to the waiting state. The algorithm outlined in Figure 15 is continued until all tasks are no longer available. This algorithm allows for dynamic execution of and switching between task in an unknown environment as tasks are discovered.
Chapter IV

Results

The proposed method will be evaluated in two ways. The first evaluation will compare the Task Allocator to random and greedy task allocation schemes. The second evaluation will be a case study utilizing a simulation engine of an unmanned vehicle system that will be tasked with accomplishing a series of missions in a dynamic environment. Thus, in this second evaluation phase, the efficacy of the Search Space Updater, Task Evaluator, Task Allocator, and Task Executor will all be studied.

Allocator Evaluation

For the allocation evaluation, three different task allocation schemes were to be compared. For these schemes, the costs under consideration were to be score and time. The schemes considered here were the modified A* with Bounded Costs, a greedy scheme that orders tasks from most to least rewarding, and a scheme that executes the tasks randomly. The greedy scheme was selected as it both was similar to the task allocation scheme utilized by the Minion ASV (Barnes et al., n.d.) and was a method that would only optimize over the score cost. Likewise, the random allocation was selected as it did not order the tasks in the allocation based on any of the costs under consideration. Thus, these methods should cover the full gambit of consideration of costs in their allocation attempts. Finally, it should be noted that all three methods are required to end on a specified final task.

The scenarios for this evaluation will be randomly generated. For each set of scenarios, the following parameters will be manually controlled: number of tasks, region
of task placement, vehicle speed, range of costs, and time available. A fixed number of scenarios will be run for each unique configuration of parameters. The number to be run was determined by evaluating when the scores appeared to have converged. The process for determining convergence can be seen in Figure 16 and Figure 17.

Figure 16: 3 tasks per scenario for showing score convergence by scenario 200.
Figure 16 showed that, for a low task count, it took between 150 and 200 scenarios for the score to converge for all three algorithms. For higher task counts, convergence happened faster. It appeared that the scores for all three methods had converged between 100 and 150 scenarios for a 12 task configuration, as shown in Figure 17. Thus, it was decided that 200 scenarios would be run for each configuration as all methods appeared to have converged in score by that point for all ranges of tasks available for allocation.

In each scenario, the following elements will be randomly generated for each task comprising the scenario: task location, execution time, and score received for completion. These elements are all generated within a region where tasks can be placed, a range of execution times, and a range of scores. The cost used for evaluation purposes in this evaluation will be the time required to execute the task and the score received from completing the task. These costs were to be optimized in order to get the best score.
achievable in the time allowed. The time allowed acted as the sole constraint placed on the allocation. This constraint comes in the form of the time that is available to perform the allocation. All allocations must take less than or equal to complete than the time available or the allocation will not be accepted. The time available will be varied from 0 to a maximum available time. The maximum time available will be unique to each scenario and is calculated as shown in Equation 8:

Equation 8

\[ t_{travel} = \frac{\sqrt{(RWidth^2 + RHeight^2)}}{Spd} \]

\[ Max Time Available = (NumTask - 1) \times t_{travel} + \sum_{i=1}^{NumTasks} (t_{execution,i}) \]

The first configuration will have 200 scenarios run. For this set, the manually controlled parameters will be as shown in Table 8 and the results for this configuration can be seen in Figure 18 to Figure 22.
Table 8: 100m x 100m region baseline configuration for 200 scenarios run.

<table>
<thead>
<tr>
<th>Number of tasks (NumTasks)</th>
<th>3, 5, 8, 10, or 12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: Width (RWidth)</td>
<td>100m</td>
</tr>
<tr>
<td>Region: Height (RHeight)</td>
<td>100m</td>
</tr>
<tr>
<td>Costs: Score generation range</td>
<td>Between 80 and 120 points</td>
</tr>
<tr>
<td>Costs: Execution time range</td>
<td>Between 40 and 60 seconds</td>
</tr>
<tr>
<td>Time available</td>
<td>Increases from 0 to Max Time Available (defined below and unique to each scenario) seconds in increments of 2% of the Max Time Available</td>
</tr>
</tbody>
</table>

Figure 18: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points.
Table 9: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 18.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>1.64%</td>
<td>8.10%</td>
<td>1.76%</td>
</tr>
<tr>
<td>Random</td>
<td>2.33%</td>
<td>9.34%</td>
<td>1.08%</td>
</tr>
</tbody>
</table>

Figure 19: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points.

Table 10: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 19.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>3.34%</td>
<td>9.08%</td>
<td>2.15%</td>
</tr>
<tr>
<td>Random</td>
<td>4.77%</td>
<td>12.04%</td>
<td>1.76%</td>
</tr>
</tbody>
</table>
Figure 20: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points.

Table 11: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 20.

<table>
<thead>
<tr>
<th></th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>4.82%</td>
<td>9.35%</td>
<td>3.18%</td>
</tr>
<tr>
<td>Random</td>
<td>6.77%</td>
<td>13.38%</td>
<td>3.50%</td>
</tr>
</tbody>
</table>
Figure 21: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points.

Table 12: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 21.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>5.49%</td>
<td>10.36%</td>
<td>3.53%</td>
</tr>
<tr>
<td>Random</td>
<td>7.23%</td>
<td>13.67%</td>
<td>3.67%</td>
</tr>
</tbody>
</table>
Figure 22: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 80-120 points.

Table 13: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 22.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>5.90%</td>
<td>11.04%</td>
<td>4.43%</td>
</tr>
<tr>
<td>Random</td>
<td>7.91%</td>
<td>14.71%</td>
<td>4.41%</td>
</tr>
</tbody>
</table>

Table 14: Summary of the average score differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13).

<table>
<thead>
<tr>
<th>Number of Tasks</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.64%</td>
<td>2.33%</td>
</tr>
<tr>
<td>5</td>
<td>3.34%</td>
<td>4.77%</td>
</tr>
<tr>
<td>8</td>
<td>4.82%</td>
<td>6.77%</td>
</tr>
<tr>
<td>10</td>
<td>5.49%</td>
<td>7.23%</td>
</tr>
<tr>
<td>12</td>
<td>5.90%</td>
<td>7.91%</td>
</tr>
</tbody>
</table>
Table 15: Summary of the maximum score differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13).

<table>
<thead>
<tr>
<th>Number of Tasks</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8.10%</td>
<td>9.34%</td>
</tr>
<tr>
<td>5</td>
<td>9.08%</td>
<td>12.04%</td>
</tr>
<tr>
<td>8</td>
<td>9.35%</td>
<td>13.38%</td>
</tr>
<tr>
<td>10</td>
<td>10.36%</td>
<td>13.67%</td>
</tr>
<tr>
<td>12</td>
<td>11.04%</td>
<td>14.71%</td>
</tr>
</tbody>
</table>

Table 16: Summary of the average time efficiency differences between ABC vs greedy and random methods for varying numbers of tasks (from Table 9 through Table 13).

<table>
<thead>
<tr>
<th>Number of Tasks</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.76%</td>
<td>1.08%</td>
</tr>
<tr>
<td>5</td>
<td>2.15%</td>
<td>1.76%</td>
</tr>
<tr>
<td>8</td>
<td>3.18%</td>
<td>3.50%</td>
</tr>
<tr>
<td>10</td>
<td>3.53%</td>
<td>3.67%</td>
</tr>
<tr>
<td>12</td>
<td>4.43%</td>
<td>4.41%</td>
</tr>
</tbody>
</table>

From the plots showing the normalized scores and the percent time utilized (or the measure of how efficiently the allocation utilized the time available) in Figure 18 through Figure 22, it was evident that A* with Bounded Costs generated higher average scores and had a higher time utilization efficiency than the other two methods being compared. Additionally, when looking at the available time when each method had all 200 scenarios achieve the maximum available score (i.e. normalized score was 1), it was evident that ABC reached the maximum achievable score faster than the other methods under consideration. This is shown in the Average Times plots in Figure 18 through Figure 22 as the point past which the time usage efficiency is no longer calculated (as all tasks would be guaranteed to be allocated by this point) for each method. It should also be noted here that the efficiency of the time utilized tapered off as the maximum achievable
score was reached. This was caused by the number of viable allocations decreasing as the
maximum allowable time increased, effectively reaching a point where only the last few
tasks were waiting to have enough time to be added. Additionally, when evaluating the
average scores achieved for each method, it is clear from the average score plots in
Figure 18 through Figure 22 showed ABC achieved the same or better average scores
than either the greedy or random methods for any numbers of tasks tested for this
configuration.

This trend was also shown to continue when comparing the average and
maximum difference between the average scores generated for ABC versus the greedy
and random methods, as shown in the summaries in Table 14 and Table 15. It should also
be noted from the summaries in Table 14 and Table 15 that the difference between the
average and maximum average scores achieved between ABC and the greedy and
random methods increased as the number of tasks increased. From 3 to 12 tasks, the
average and maximum differences for ABC compared to greedy increased from 1.64% to
5.90% and from 8.10% to 11.04%, respectively. Likewise, comparing ABC to the
random method for 3 to 12 tasks, the average and maximum differences increased from
2.33% to 7.91% and from 9.34% to 14.71%, respectively. Finally, the time efficiency
ABC also increased as the number of tasks increased when compared to the greedy and
random methods, as shown in the summary of results in Table 16. These results were
likely caused due to the increasing number of tasks likewise having an increasing number
of permutations for available for task allocation configurations. Thus, ABC, which
optimized over both costs, was able to make use of that optimization to find the optimal
allocation for each scenario in this configuration subject to the costs presented. Whereas
the greedy and random methods were only able to optimize over a single and over none of the costs, respectively, which ultimately lowered their time usage efficiency (and by extension their average score achieved for a given maximum allowable time).

Following this, additional configurations were generated by changing the range of task costs. This allowed for the impact of varying the available scores and available execution times for tasks on the average score and average time to complete to be assessed with respect to the baseline configuration. All other parameters remained constant. For this configuration, the same range of tasks was tested as before. However, for the sake of brevity, only the 12 task result for this configuration is presented here. The results for the other task counts for this configuration can be found in Appendix A. Table 17 shows the parameters for this configuration with the updated parameters bolded for clarity and, the results for 12 tasks shown in Figure 23.

<table>
<thead>
<tr>
<th>Number of tasks (NumTasks)</th>
<th>3, 5, 8, 10, or 12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: Width (RWidth)</td>
<td>100m</td>
</tr>
<tr>
<td>Region: Height (RHeight)</td>
<td>100m</td>
</tr>
<tr>
<td>Costs: Score generation range</td>
<td>Between 20 and 180 points</td>
</tr>
<tr>
<td>Costs: Execution time range</td>
<td>Between 40 and 60 seconds</td>
</tr>
<tr>
<td>Time available</td>
<td>Increases from 0 to Max Time Available seconds in increments of 2% of the Max Time Available</td>
</tr>
</tbody>
</table>
Figure 23: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points.

Table 18: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 23.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>4.14%</td>
<td>7.19%</td>
<td>3.92%</td>
</tr>
<tr>
<td>Random</td>
<td>13.06%</td>
<td>24.35%</td>
<td>3.96%</td>
</tr>
</tbody>
</table>

Figure 23 shows a continuation of the trends found in Figure 18 through Figure 22 despite expanding the range of scores that could be assigned to each task. That is, ABC continued to utilize the available time more efficiently to maximize score and still achieved a higher score for all available times under consideration. However, in these plots, an interesting difference in the average score generated was noted. In Figure 23, the gap between the ABC and the random method increased, whereas the gap between ABC and the greedy method shrank, compared to the results in Figure 22. This is clear when
comparing the results presented in Table 13 to Table 18. With the score range expanded, the average and maximum differences for ABC compared to greedy fell from 5.90% to 4.14% and from 11.04% to 7.19%, respectively. On the other hand, comparing ABC to the random method, the average and maximum differences increased from 7.91% to 13.06% and from 14.71% to 24.35%, respectively. These results were caused by the wider score range effectively causing the score cost to become cost in the optimization. Thus, the greedy algorithm, which solely focused on optimizing cost, was able to slightly close on the ABC method as ABC was still attempting to simultaneously optimize the score and time costs. However, it was clear that, as the range of possible scores available for tasks widened, the random ordering method tended to produce far lower value allocations for the time available since it was not focusing on optimizing the now more impactful score cost. Finally, it should be noted that the difference in the efficiency of time usage did drop for this scenario when comparing ABC to either the greedy or random methods, as shown by comparing the results in Table 13 to those in Table 18. However, this was likely caused by the increased ranges of scores becoming the more dominant cost, which ultimately reduced the ability to efficiently utilize the time available as the goal is to maximize score for the least amount of time used (assuming the available time constraint is not violated).

To test the idea of a cost becoming more dominate when the cost’s range was expanded, the next test sought to verify the impact of expanding the times required to complete each task when the range of scores was narrow. Thus, the costs were again confined to the smaller range of between 80 and 120 points. The execution time range was then increased to between 10 and 90 seconds. The expectation for this configuration
would be that the time efficiency for ABC versus the greedy and random methods would increase, but it would do so at the expense of the difference in the average score achieved. As in the previous configuration, only the results for the 12 task configuration are shown here. The results for the other task counts for this configuration can be found in Appendix A. The configuration under test here can be seen in Table 19 with the results gathered for the 12 task configuration shown in Figure 24.

Table 19: Configuration with expanded execution time range.

<table>
<thead>
<tr>
<th>Number of tasks (NumTasks)</th>
<th>3, 5, 8, 10, or 12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: Width (RWidth)</td>
<td>100m</td>
</tr>
<tr>
<td>Region: Height (RHeight)</td>
<td>100m</td>
</tr>
<tr>
<td>Costs: Score generation range</td>
<td>Between 80 and 120 points</td>
</tr>
<tr>
<td>Costs: Execution time range</td>
<td>Between 10 and 90 seconds</td>
</tr>
<tr>
<td>Time available</td>
<td>Increases from 0 to Max Time Available in increments of 2% of the Max Time Available</td>
</tr>
</tbody>
</table>
Figure 24: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points.

Table 20: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 24.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>8.51%</td>
<td>15.85%</td>
<td>4.75%</td>
</tr>
<tr>
<td>Random</td>
<td>10.73%</td>
<td>20.09%</td>
<td>4.74%</td>
</tr>
</tbody>
</table>

The same general trend of ABC being the best performer in terms of score accrual and efficient time usage continues with the wider range of task execution times, as demonstrated by Figure 24. As was expected, when comparing the results for the difference in average time usage efficiency in Table 13 versus Table 20, the ABC method was able to increase the difference between the greedy and random methods results from 4.43% to 4.75% and from 4.41% to 4.74%, respectively. However, unexpectedly, this also resulted in the average and maximum average score difference between ABC and
both the greedy and random methods to also increase over the results for the narrow ranges for both scores in the configuration for Figure 22. This was likely caused due to the increased value of the time cost meaning allocation which inefficiently made use of ordering by time were not able to produce allocations which had scores that were within the maximum allowable time constraint. Thus, ABC, which was able to handle both costs, was likely able to use multiple lower cost options that had low time costs to make more efficient usage of the available time while also achieving a higher average score.

Finally, for the 100m x 100m region, a configuration that featured both an expanded score range and an expanded execution time range was generated. This wide range of variance in task scores and completion times was intended to make any inefficiencies in selection obvious as it would exacerbate any biases in selecting based on score or time. As before, only the 12 task result was displayed, with the other task configurations being found in Appendix A. Table 21 outlined the parameters for this configuration, and Figure 25 reported the results generated for this configuration.

Table 21: Configuration with expanded score and execution time ranges.

<table>
<thead>
<tr>
<th>Number of tasks <em>(NumTasks)</em></th>
<th>3, 5, 8, 10, or 12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: Width <em>(RWidth)</em></td>
<td>100m</td>
</tr>
<tr>
<td>Region: Height <em>(RHeight)</em></td>
<td>100m</td>
</tr>
<tr>
<td>Costs: Score generation range</td>
<td>Between 80 and 120 points</td>
</tr>
<tr>
<td>Costs: Execution time range</td>
<td>Between 10 and 90 seconds</td>
</tr>
<tr>
<td>Time available</td>
<td>Increases from 0 to Max Time Available seconds in increments of 2% of the Max Time Available</td>
</tr>
</tbody>
</table>
Figure 25: Average of 200 scenarios with 12 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points.

Table 22: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 24.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>5.86%</td>
<td>12.58%</td>
<td>4.53%</td>
</tr>
<tr>
<td>Random</td>
<td>14.30%</td>
<td>27.87%</td>
<td>4.60%</td>
</tr>
</tbody>
</table>

As was to be expected, this configuration showed the ABC algorithm eclipsing the results for the random ordering. As demonstrated through Figure 25, this configuration showed the greatest gap between the average score achieved with ABC being an average of 14.30% and a maximum of 27.87% greater than the average achieved score of the random method. Likewise, since it was biased towards one of the parameters being heavily varied, the sorted order ended up performing midway between the ABC
algorithm and the random ordering algorithm. ABC found allocations that lead to an average score that was on average 5.86% and at best 12.58% better than the average allocation scores found by the greedy method. This result was expected as sometimes the highest score tasks would have the lowest times required for execution. However, this was balanced by some of the high score tasks also having some of the highest times to complete. Finally, as was expected, the difference in average time usage efficiency was better for this configuration than the narrow baseline (Table 13) or expanded score range (Table 18) configurations, but was slightly lower than the purely expanded range for the time cost (Table 20). This was to be expected as the configuration that lead to the results in Table 22 naturally had greater fluctuations in combinations of time and score costs, which allowed for allocations that could make the most efficient usage of the maximum time available when both costs were optimized.

Finally, a configuration was created to ascertain the impact of the transit time on the allocation efficiency. This was done by expanding the region that the tasks could be randomly placed in so that the transit time could dominate over the task execution time. For comparison sake with the previous configurations for the 100m x 100m region, only 12 tasks were used for all of the configurations here. The configuration can be seen in Table 23 and the results generated are in Figure 26 through Figure 29.
Table 23: Configuration for the larger task region.

<table>
<thead>
<tr>
<th>Number of tasks ($NumTasks$)</th>
<th>12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region: Width ($\text{Width}$)</td>
<td>1000m</td>
</tr>
<tr>
<td>Region: Height ($\text{Height}$)</td>
<td>1000m</td>
</tr>
<tr>
<td>Costs: Score generation range</td>
<td>Either between 80 and 120 points or between 20 and 180 points</td>
</tr>
<tr>
<td>Costs: Execution time range</td>
<td>Either between 40 and 60 seconds or between 10 and 90 seconds</td>
</tr>
<tr>
<td>Time available</td>
<td>Increases from 0 to Max Time Available seconds in increments of 2% of the Max Time Available</td>
</tr>
</tbody>
</table>

Figure 26: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 40-60 seconds, and a score range of 80-120 points.
Table 24: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 26.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>13.27%</td>
<td>33.17%</td>
<td>6.11%</td>
</tr>
<tr>
<td>Random</td>
<td>14.87%</td>
<td>36.78%</td>
<td>6.08%</td>
</tr>
</tbody>
</table>

Figure 27: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 40-60 seconds, and a score range of 20-180 points.

Table 25: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 27.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>9.78%</td>
<td>25.57%</td>
<td>6.93%</td>
</tr>
<tr>
<td>Random</td>
<td>16.60%</td>
<td>41.67%</td>
<td>5.45%</td>
</tr>
</tbody>
</table>
Figure 28: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 10-90 seconds, and a score range of 80-120 points.

Table 26: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 28.

<table>
<thead>
<tr>
<th>ABC vs</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>13.10%</td>
<td>32.90%</td>
<td>4.94%</td>
</tr>
<tr>
<td>Random</td>
<td>14.71%</td>
<td>36.82%</td>
<td>6.48%</td>
</tr>
</tbody>
</table>
Figure 29: Average of 200 scenarios with 12 tasks per scenario, 1000m x 1000m region, an execution time range of 10-90 seconds, and a score range of 20-180 points.

Table 27: Average score and time efficiency comparison for ABC vs greedy and random methods for the configuration in Figure 29.

<table>
<thead>
<tr>
<th>ABC vs:</th>
<th>Average Score Difference</th>
<th>Maximum Score Difference</th>
<th>Average % Time Utilized Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>10.52%</td>
<td>25.68%</td>
<td>5.47%</td>
</tr>
<tr>
<td>Random</td>
<td>17.92%</td>
<td>42.19%</td>
<td>5.45%</td>
</tr>
</tbody>
</table>
Table 28: Summary of the average score differences between ABC vs greedy and random methods for each configuration and each configuration space size.

<table>
<thead>
<tr>
<th>Cost Ranges</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100m x 100m</td>
<td>1000m x 1000m</td>
</tr>
<tr>
<td>Score: 80-120 Time: 40-60</td>
<td>5.90%</td>
<td>13.27%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 40-60</td>
<td>4.14%</td>
<td>9.78%</td>
</tr>
<tr>
<td>Score: 80-120 Time: 10-90</td>
<td>8.51%</td>
<td>13.10%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 10-90</td>
<td>5.86%</td>
<td>10.52%</td>
</tr>
</tbody>
</table>

Table 29: Summary of the maximum score differences between ABC vs greedy and random methods for each configuration and each configuration space size.

<table>
<thead>
<tr>
<th>Cost Ranges</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100m x 100m</td>
<td>1000m x 1000m</td>
</tr>
<tr>
<td>Score: 80-120 Time: 40-60</td>
<td>11.04%</td>
<td>33.17%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 40-60</td>
<td>7.19%</td>
<td>25.57%</td>
</tr>
<tr>
<td>Score: 80-120 Time: 10-90</td>
<td>15.85%</td>
<td>32.90%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 10-90</td>
<td>12.58%</td>
<td>25.68%</td>
</tr>
</tbody>
</table>

Table 30: Summary of the average time efficiency differences between ABC vs greedy and random methods for each configuration and each configuration space size.

<table>
<thead>
<tr>
<th>Cost Ranges</th>
<th>ABC vs Greedy</th>
<th>ABC vs Random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100m x 100m</td>
<td>1000m x 1000m</td>
</tr>
<tr>
<td>Score: 80-120 Time: 40-60</td>
<td>4.43%</td>
<td>6.11%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 40-60</td>
<td>3.92%</td>
<td>6.93%</td>
</tr>
<tr>
<td>Score: 80-120 Time: 10-90</td>
<td>4.75%</td>
<td>4.94%</td>
</tr>
<tr>
<td>Score: 20-180 Time: 10-90</td>
<td>4.53%</td>
<td>5.47%</td>
</tr>
</tbody>
</table>
Figure 26 through Figure 29 utilized similar cost ranges to those used in the 12 task configurations found in Figure 22 through Figure 25. Thus, direct comparisons could be made between the results for these configurations. The results from those plots were compared side-by-side in Table 28 through Table 30. The key difference, as was expected, was that the transit time did end up dominating in each of these configurations (Table 23). Thus, as was shown in the case of solely expanding the range of the time cost, having the time cost dominate the variance of the costs ultimately caused the ABC algorithm to show improvements in the average and maximum average scores achieved as well as the average time efficiency usage for all configurations when compared to both the greedy and random methods. In fact, in the 1000m x 1000m region configurations, the maximum score difference was found. At most, when 12 tasks were considered and both cost’s ranges were expanded, the difference between the maximum average score achieved for ABC was found to be 42% greater than that of the random method. However, as was shown before, since the greedy method did optimize over the score cost (unlike the random method which did not optimize over either), the greatest maximum average score difference between the ABC and greedy method was nearly 33%. From the comparisons between the results shown in Table 28 through Table 30, the ABC algorithm, which had a superior handling of time and score constraints compared to the other methods, maintained a significant lead in average score achieved for all configurations. Additionally, the plots for the average times for all of the configurations tested (Figure 18 through Figure 29) showed that ABC was able to achieve the maximum achievable score with the least amount of required time.
Case Study

The other form of verification that occurred was a case study which evaluated the performance of the full-stack algorithm in simulated environments. The scenarios used in this case study evaluated the performance of the algorithm when faced with dynamic task discovery and tasks with predecessor/successor requirements. Naturally, as is assumed by this method, the tasks in each scenario were known prior to mission execution, but the exact locations of each task were not known until they were discovered in the operational region.

Simulation Environment

The scenarios utilized here are motivated by the Maritime RobotX competition with the simulated vehicle under test being taken from Embry-Riddle Aeronautical University’s Team Minion’s entry to the 2018 competition. The Minion ASV is a maritime research platform operated by Team Minion of Embry-Riddle Aeronautical University’s (Daytona Beach, FL, USA). According to Barnes et al. (n.d.), the design of the Minion ASV is based on the Marine Advanced Robotics Modular Wave Adaptive Vessel (WAM-V) 16 ASV base platform, shown in Figure 30.
Minion utilizes four light detection and ranging (LiDAR) sensors and two cameras for object detection and classification. The four LiDARs are a forward Velodyne HDL-32E, a forward-port and a forward-starboard Velodyne VLP-16 HD, and an aft Velodyne VLP-16. The two cameras are FLIT Blackfly BFLY-PGE-31S4C-C forward facing high-definition cameras. The system localizes itself and its sensor data into a NED global reference frame based on GPS data from its TORC Pinpoint GPS/INS. These sensors were used to generate the visibility region, $P_{FRD}$. For this case study, $P_{FRD}$ was the set to be the same as shown in Figure 2.

The simulation environment utilized in this case study sought to simplify many of the systems on the Minion ASV that would normally be responsible for perceiving and navigating through the environment. The visibility region shown in Figure 2 was simulated such that when this polygon passed over the center of a polygon representing an object of interest to the ASV, the object would immediately show as being detected and classified. Navigation was also simplified in this simulation to remove any impact of
a path planning or controls system on the evaluation of Minion Task. Thus, when Minion Task sent a new target waypoint to the path planning system, the simulator would generate straight line from the current position of the simulated ASV to the new target waypoint. The simulator would then use the speed of the ASV to step the simulated vehicle along this straight-line path such that the vehicle would reach the target point in the amount of time required to reach it if traveling that straight line at that speed. This would effectively move the vehicle “on-rails” from the current position to the target point at the desired speed.

**Task Configuration**

The scenarios that the simulated Minion was to be subjected to completing would include objects and tasks reminiscent of those in the 2018 Maritime RobotX competition. This would include the tall buoys as well as the light tower utilized by the competition as object that the system was to identify and then perform a variety of actions around. In order to boil down the complexity of the tasks so that the pure capabilities of the task allocation method, rather than the operator’s ability to correctly decompose tasks, was evaluated, a simplified set of behaviors was identified for completion. This would include finding gates to transit through, similar to the Entry/Exit gates challenge, and finding a light tower to stare at. This simplified set of behaviors, finding an object of interest, transiting to a desired start point, and then completing an action, would show the capabilities of Minion Task. The scenario configurations to follow all utilize the three tasks presented in Figure 31 through Figure 33 as part of their missions.
Task 1

Figure 31: Task 1; gate entry. Requires two (2) tall buoys with reflectors ~10m apart.

The first task, shown in Figure 31, requires a Green Tall Buoy with Reflector as well as a Red Tall Buoy with Reflector to be detected. For this task, the task ready check, \( tsk_{1, TaskReadyCheck} \), is also found to have all starting elements found when:

1. A Red Tall Buoy with Reflector is detected and classified
2. A Green Tall Buoy with Reflector is detected and classified
3. The spacing between a green/red buoy pair is less than 15m apart

If the above conditions are met, \( tsk_{1, TaskReadyCheck} \) sets \( tsk_{1, Status} \) to Found; otherwise, it is left unchanged. The green and red buoys effectively act as the gate that the system is to navigate through. The system also sets its transit point \( (tsk_{1, Locs}) \), shown in the figure, to be just before the gate. After reaching this point, the vehicle is to transit 55m from the transit point through the center of the gate in the direction +90° from the
angle formed by the vector pointing from the red buoy to the green buoy. Another way to think of it is that the vessel must transit through the gate with the green buoy to its starboard side. This task partially emulates the entry part of the entry/exit gates challenge in the 2018 Maritime RobotX competition (2018 Maritime RobotX Challenge Task Descriptions and Specifications, 2018). However, to prevent any complexities in the task potentially preventing the capabilities of Minion Task from being demonstrated, it has instead been tuned down to a single gate instead of a string of 3 gates.

Task 2

![Diagram of Task 2](image)

Figure 32: Task 2; Light tower observation. Requires one (1) light tower to start.

Task 2 emulates the scan the code challenge in the Maritime RobotX competition. For this task, the task ready check, $t_{sk2,TaskReadyCheck}$, was found to have all starting elements found upon detecting and classifying the Light Tower object.
4. A Red Tall Buoy with Reflector is detected and classified
5. A Green Tall Buoy with Reflector is detected and classified
6. The spacing between a green/red buoy pair is less than 15m apart

If this condition was met, $t_{sk_2,TaskReadyCheck}$ would then set $t_{sk_2,Status}$ to Found. As shown in Figure 32, the system, upon detecting the light tower object, was to set its transit point ($t_{sk_2,Loc_3}$) to a point 30m to the East of the center of the light tower and then face West towards the tower. This acts as the transit point for this task. After achieving this point, the vessel is to approach closer to the tower, emulating getting closer to scan the code. The final distance the vessel is to get to is 10m to the East of the tower, facing said tower. Again, to reduce complexity, this task is considered complete one that final point is achieved.
Task 3

Figure 33: Task 3; gate exit. Requires exiting the same gate as Task 1, but requires going through with the red buoy to the starboard side.

Finally, task 3 represents the exit gate task of the entry/exit challenge. For this task, the system is to again utilize the same buoys as in Task 1 to act as a gate to transit this time. Thus, task 3’s task ready check, $tsk_{3, TaskReadyCheck}$, is also found to have all starting elements found when:

7. A Red Tall Buoy with Reflector is detected and classified
8. A Green Tall Buoy with Reflector is detected and classified
9. The spacing between a green/red buoy pair is less than 15m apart

If the above conditions are met, $tsk_{3, TaskReadyCheck}$ sets $tsk_{3, Status}$ to Found; otherwise, it is left unchanged. The green and red buoys once again act as the gate that the system is to navigate through. However, this time, as demonstrated in Figure 33, the
vessel is to transit to the transit point \((t_{sk3, loc3})\), 5m before the gate, facing the gate with the red buoy to the vehicle’s starboard side. The vessel is then to finish the task by transiting 15m forward, through the gate, to the final point on the side \(+90^\circ\) from the angle formed by the vector pointing from the green buoy to the red buoy. These tasks are then arranged into the environment so that the mission preforming capabilities of Minion Task may be tested.

For the two following scenarios, the environment was configured such that, by adding a simple previous predecessor requirement into the system, it would be possible to see two different missions for the same field and mission stack. For both scenarios, Task 3 was selected as the goal state task. Finally, between both scenarios, the vehicle configuration (Table 31), task configuration (Table 32), task object and search areas (Figure 34), and visibility region (the same as was used by in the sample shown in Figure 2) were held constant.

Table 31: General scenario configuration parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>3 tasks</td>
</tr>
<tr>
<td>Vehicle Speed</td>
<td>2 m/s, comparable to the max speed for the Minion ASV (Barnes et al., n.d.)</td>
</tr>
<tr>
<td>Starting NED Position</td>
<td>Northing: 100 m; Easting: 0 m</td>
</tr>
<tr>
<td>Starting Heading</td>
<td>180°</td>
</tr>
</tbody>
</table>
Table 32: Task assigned scores.

<table>
<thead>
<tr>
<th>Task</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1 $tsk_1$</td>
<td>50 points</td>
</tr>
<tr>
<td>Task 2 $tsk_2$</td>
<td>500 points</td>
</tr>
<tr>
<td>Task 3 $tsk_3$</td>
<td>120 points</td>
</tr>
</tbody>
</table>

Figure 34: Locations of the objects and search areas for each task within the operational environment. The global search area ($GSA$) that would bound the task search areas is also shown.
It should be noted at this stage that, due to network delays in transporting the different messages carrying the position and object list messages, some of the results shown may be desynchronized from the results reported. This was caused by the either the position or object list messages using their peer’s old message as the processes that generated these messages were run asynchronously.

**Scenario 1**

With this configuration in mind, the first scenario was commenced. For this scenario, no predecessor or successor requirements were present for any of the tasks. The system was to simply search for and complete the tasks as it traversed the environment. The results for this scenario can be seen in Figure 35 through Figure 40.

![Figure 35: (A) Initial position for the vehicle. The initial allocation found, in order, is $[t_{sk_2}, t_{sk_1} t_{sk_3}]$. The system begins searching for $t_{sk_2}$. (B) The system detects the Light Tower, the start queue for $t_{sk_2}$.](image-url)
Figure 36: (C) The system switches from searching for $tsk_2$ to head to its transit point. 
(D) The system arrives at the transit point for $tsk_2$ and begin heading for its final point.
Figure 37: (E) The system completes $tsk_2$. The achieved score is now 500 points. The allocation becomes $[tsk_1, tsk_3]$. The system begins searching for $tsk_1$ and $tsk_3$. (F) The system detects the Green Tall Buoy with Reflector.

Figure 38: (G) The system detects the Red Tall Buoy with Reflector. (H) With both buoys detected, the system switches from searching for $tsk_1$ and instead heads to its transit point.
Figure 39: (I) $t_{sk_1}$ transit point is achieved. The system begins to head to the final point for $t_{sk_1}$. (J) The system completes $t_{sk_1}$. The achieved score increases to 550 points. The allocation becomes $[t_{sk_3}]$. $t_{sk_3}$ has already been found, so the system heads to its transit point.
Figure 40: (K) The transit point for and $t_{sk_3}$ is achieved. The system heads to the final point for $t_{sk_3}$. (L) The system completes $t_{sk_3}$. All task are now complete and the final score increases to 670 points, the maximum achievable.

Throughout the key event from scenario 1, it was clear that Minion task was capable of dynamically searching for tasks in an unknown environment and dynamically accomplishing those tasks as they became available. Figure 35 through Figure 37 demonstrated the system’s ability to find an initial task within the environment based on the outlined plan and that it could complete the task as outlined. The ability to resume searching for tasks if they were still not found was also demonstrated in Figure 37 and Figure 38. Additionally, Minion Task demonstrated that it was capable of observing the requirement for task 3 to be the end state, even though completing it would have given a higher immediate reward than completing Task 1 did. Finally, Figure 38 through Figure 40 showed that the system was capable of completing tasks with overlapping search areas as well as ensuring that it was capable of completing all tasks while time was still
available. In total, this scenario took around 120 seconds in simulation time to complete. Scenario will look to evaluate the performance of the system when a predecessor requirement is included.

**Scenario 2**

For scenario 2, the system configuration outlined above was maintained sans one difference. In this scenario, both Task 2 and Task 3 had Task 1 as a prior predecessor. Thus, Task 1 had to be completed at some point before Task 2 or Task 3 could be searched for or executed. The results for this scenario can be found in Figure 41 through Figure 46.

![Figure 41: (A) Initial position for the vehicle. The initial allocation found, in order, is \([t_{sk_1}, t_{sk_2}, t_{sk_3}]\). The system begins searching for \(t_{sk_1}\). (B) The system detects the Light Tower. However, it has not completed for \(t_{sk_1}\), so it does not commence \(t_{sk_2}\).](image)
Figure 42: (C) The Red Tall Buoy with Reflector is detected. (D) The Green Tall Buoy with Reflector is detected. Now all elements for $t_{sk_1}$ have been detected.

Figure 43: (E) The system switches away from searching for $t_{sk_1}$ and begins heading to its transit point. (F) The system reaches $t_{sk_1}$’s transit point and begins heading to its final point.
Figure 44: (G) The final point for $t_{sk_1}$ is achieved. The score increases to 50 points. The allocation is now is $[t_{sk_2}, t_{sk_3}]$. No additional searching is required as all task elements have been detected. The system heads to the transit point for $t_{sk_2}$. (H) The transit point is achieved. The system heads for the final point for $t_{sk_2}$. 
Figure 45: (I) $t_{sk_2}$ final point is achieved. The achieved score increases to 550 points, and the allocation becomes $[t_{sk_3}]$. $t_{sk_3}$ has already been found, so the system heads to its transit point. (J) The system achieves transit point for $t_{sk_3}$. The system heads to the final point for $t_{sk_3}$.

Figure 46: (K) The system completes $t_{sk_3}$. All task are now complete and the final score increases to 670 points, the maximum achievable.
The scenario demonstrated by Figure 41 through Figure 46 highlights the ability for Minion Task to handle system constraints, including predecessor/successor constraints. Whereas the first scenario (Figure 35), which had no constraints, had the vehicle completing Task 2 as soon as the Light Tower was discovered, the system was instead able to find and complete the prior predecessor requirement instead. This then opened up access to Task 2 and Task 3 being attempted (as demonstrated by Figure 41 through Figure 43). Finally, Figure 43 through Figure 46 showed that Minion Task was capable of immediately starting missions, one after the other, if all task elements for those tasks had been discovered. At around 123 seconds to complete the mission, scenario 2 was slightly slower to finish than scenario 1. However, this was to be expected as the predecessor requirement prevented the system from exploring for and completing tasks in the optimal way it would have wanted.
Chapter V

Discussion, Conclusions, and Recommendations

Discussion

A novel method that enables a singular agent to search for and execute one or more tasks in a dynamic environment has been presented. In Chapter II, the need for a clear definition of an agent and for tasks in the context of a single agent operating in a dynamic environment was presented. Additionally, by reviewing the relevant literature, it was determined that other multi-agent task allocation methods that focused on solving tasks in dynamic environments, such as in search and rescue scenarios, proved ineffective when reduced to a single agent as they did not effectively allow for optimization over a range of constraints.

Thus, in the methodology presented, Minion Task, was shown to be capable meeting these needs of a single agent that were presented at the beginning of Chapter III. Minion Task was capable of handling constraints on both the agent and the tasks within the mission through its Search Space Updater and Task Evaluator algorithms. For the tasks in the mission configuration, Minion Task could determine if a task was available, could determine the costs for a task, and could finding the optimal ordering of tasks for execution over multiple costs and constraints through its Task Allocator algorithm, which utilized the A* with Bound Costs algorithm to find the optimal allocation given a set of costs to optimize and a set of constraints to consider. Finally, through the pipeline presented in Minion Task for the Task Executor algorithm, it was shown that the ASV was capable of transiting to and executing the tasks in a given task allocation.
The results that were presented in Chapter IV demonstrated that the original research contributions were able to be achieved. Through the task allocator evaluation, a graph search-based task allocation, the A* with Bounded Costs (Logan & Alechina, 1998) algorithm, was shown to produce allocations which produced higher average scores with more efficient usage of the allowable time for the allocation when compared to greedy and random allocation algorithms under the costs and constraints considered. When comparing these algorithms in the context of optimizing score and time costs while staying within a maximum allowable time constraint, it was found that ABC produced average scores that were always the same or better than the average scores of the greedy and random methods, for the configurations tested. At most, when 12 tasks were considered, the difference between the maximum average score achieved for ABC was found to be 42% greater than that of the random method and nearly 33% greater than the greedy method. Additionally, across all of the configurations tested for 12 tasks, ABC was shown to have a difference in the average time usage efficiency that was at most 6.93% greater than the greedy method and 6.48% greater than the random method.

In the case study presented in Chapter IV, it was shown that Minion Task was capable of handling dynamic task discovery in a simulated environment that was close to that of the challenges presented in the 2018 Maritime RobotX Challenge (2018 Maritime RobotX Challenge Task Descriptions and Specifications, 2018). By simulating Team Minion’s entry, the Minion ASV (Barnes et al., n.d.), the method was able to show a practical usage of Minion Task in the context of the RobotX challenge. Through the first scenario presented, it was shown that Minion Task was capable of handling dynamic switching between searching for and executing tasks. It was also shown that the proposed
method could complete a mission in a space where the search areas for the tasks were known, but the exact locations of the tasks were not known prior to mission execution. Finally, the second scenario showed that Minion Task was able to handle the same environment when constraints were placed on the order the tasks were able to be searched for and executed. Through the results gathered in Chapter IV, it was clear that the method presented here met the goal of creating a method for single agents to efficiently perform dynamic task allocation in partially defined environments.

**Recommendations**

While the work presented here provided a number of features that are needed for single agents to perform dynamic tasking in partially defined environments, there is still plenty of room for future work in this area. Directly building off this method, additional work should be conducted to improve the method for searching for a task. The process outlined in Chapter III: *Search Ready Check Evaluation* for finding the transit points when searching for a task could be refined to select the points that would cover the remaining portion of the search region while using the shortest path, least energy, etc. Additionally, alternate task allocation algorithms should also be evaluated and compared to the modified A* with Bound Costs algorithm used in Chapter III: Task Allocator. As this is inherently a traveling salesman problem, alternate graph search and/or route planning algorithms may prove to be a better allocator in certain scenarios. Finally, future work on Minion Task assuredly includes greater robustness testing of the algorithm. This includes edge case testing of the allocations and execution. It also includes testing the algorithm on other platforms. A final test of the robustness of the algorithm would be seeing if it could be used in the multi-agent context as well.
Conclusions

The work presented in this paper sought to find a method capable of expanding the usage of autonomous systems in the maritime domain (Yuh et al., 2011) to be able to effectively perform dynamic task allocation in partially defined environments. This paper succeeded in this endeavor, presenting a method, Minion Task, that was capable of performing missions in dynamic maritime environments, such as those found in the Maritime RobotX competition (2018 Maritime RobotX Challenge Task Descriptions and Specifications, 2018). In the task allocation evaluation performed on the A* with Bounded Costs algorithm, it was shown that a graph search-based algorithm could be used task allocation to produce the most efficient task ordering given a set of costs to optimize over and a set of constraints bounding the allocation. Additionally, this evaluation also showed that the ABC algorithm produced better average costs and had higher average time usage efficiency when compared to greedy and random allocation algorithms for the costs and constraints considered. Finally, through the case study used to evaluate the overall performance capabilities of Minion Task, this method was shown to be an algorithm that was capable of search for and execute tasks in a dynamic environment. This ultimately helps satisfy the goals of numerous bodies and agencies that are seeking to find ways to allow their systems to operate in this challenging domain.
References


http://dx.doi.org.ezproxy.libproxy.db.erau.edu/10.1007/s10846-017-0493-x


https://doi.org/10.5772/13281


https://doi.org/10.1145/3128584

https://doi.org/10.1007/s10489-018-1169-3

https://doi.org/10.5772/57313


https://doi.org/10.1109/TCYB.2015.2418052
Appendix A

Allocator Evaluation Results Continued

The following results were generated for the configuration presented in Table 17:

Figure 47: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points.
Figure 48: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points.

Figure 49: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points.
Figure 50: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 40-60 seconds, and a score range of 20-180 points.

The following results were generated for the configuration outlined in Table 19:

Figure 51: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points.
Figure 52: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points.

Figure 53: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points.
Figure 54: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 80-120 points.

The following results were obtained from the configuration outlined in Table 21:

Figure 55: Average of 200 scenarios with 3 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points.
Figure 56: Average of 200 scenarios with 5 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points.

Figure 57: Average of 200 scenarios with 8 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points.
Figure 58: Average of 200 scenarios with 10 tasks per scenario, an execution time range of 10-90 seconds, and a score range of 20-180 points.