Scheduling, Complexity, and Solution Methods for Space Robot On-Orbit Servicing

Susan E. Sorenson
University of Arkansas, Fayetteville

Scheduling, Complexity, and Solution Methods for Space Robot On-Orbit Servicing

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Industrial Engineering

by

Susan E. Sorenson
University of West Florida
Bachelor of Science in Mathematics, 2003
Air Force Institute of Technology
Master of Science in Applied Mathematics, 2005

August 2022
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Abstract

This research proposes problems, models, and solutions for the scheduling of space robot on-orbit servicing. We present the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots problem which considers on-orbit servicing across multiple orbits with moving tasks and moving refuelling depots. We formulate a mixed integer linear program model to optimize the routing and scheduling of robot servicers to accomplish on-orbit servicing tasks. We develop and demonstrate flexible algorithms for the creation of the model parameters and associated data sets. Our first algorithm creates the network arcs using orbital mechanics. We have also created a novel way to mathematically represent the movement of the tasks and refuelling depots and present algorithms for constructing both sets of data. We create robust case studies based on current operational satellites in Low Earth Orbit, Mid Earth Orbit, and Geosynchronous Earth Orbit. With these case studies we perform extensive computational experiments to present example insights about robot servicers, task completion, and their use of refuelling depots.

Building upon this work, we next focus on proving the computational complexity and generating fast, accurate algorithms and present and demonstrate two solution methods. The solution methods use node labels akin to those in Dijkstra's algorithm but include much more information about the servicers, tasks, and fuel levels. We use the labels to find the shortest paths to tasks which are in motion on the network. The first heuristic assigns servicers to tasks greedily and the second heuristic assigns tasks using a clustering algorithm. We use a case study to compare our heuristic time and solution performance with CPLEX with promising results.

In our final work, we address the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots with Known Task Times. Previously, we considered the tasks to have instantaneous completion which is realistic for survellance type tasks but not for the more intricate tasks, such as corrective maintenance or equipment upgrade. Thus we remove the assumption of instantaneous task completion and consider a known task processing time. We present a new mixed integer linear program model to optimize the routing and scheduling of robot servicers to accomplish the on-orbit servicing tasks. As both tasks and servicers move, considering task duration is complex because (i) the task and servicer must coincide at the correct location and time, (ii) the task and servicer must move through the network together for at least the duration of the processing time, and (iii) the completion of the task is at a different location and time than the start. The model accounts for the movement of the servicers, tasks, and refuelling depots and also for the task duration. We also present two related constructive heuristics for solving the problem. We also incorporate the task times into a case study which is based on satellites and orbits which are in use today. We use the case study to conduct computational experiments comparing the heuristic solving times and solution accuracy with CPLEX.

Contents

1 Introduction 1

Bibliography . . . 4

2 Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots 5

2.1 Introduction . . . 5

2.2 Literature Review . . . 8

2.3 Problem Statement and Methodology . . . 10

2.3.1 Problem Formulation and Algorithm Development . . . 10

2.3.2 Parameters, Sets, and Mathematical Model . . . 14

2.3.2.1 Parameters/Sets . . . 15

2.3.2.2 Decision Variables . . . 16

2.3.2.3 Model . . . 16

2.4 Data and Computational Results . . . 17

2.4.1 Network Construction . . . 17

2.4.2 Core Case Study . . . 22

2.4.2.1 Core Case Study: Test Design and Data Generation . . . 22

2.4.2.2 Core Case Study: Results and Insights . . . 26

2.4.3 Refuelling Case Study . . . 32

2.4.4 Single Versus Multi-Orbit Case Study . . . 34

2.5 Conclusions . . . 35

Bibliography . . . 38

Appendices 41

Appendix 2.A Appendix: Orbital Mechanics . . . 41

2.A.1 Orbital Parameters and Constants . . . 41

2.A.2 Hohmann Transfer . . . 42

2.A.3 Combined Hohmann Transfer with Inclination Change . . . 43

2.A.4 Phasing Maneuver . . . 44

Appendix 2.B Demonstration of Algorithms Presented in this Work . . . 46

2.B.1 Arc Creation Algorithm Example . . . 46

2.B.2 Sub Task Creation Algorithm Example . . . 46

2.B.3 Refueling Arc Designation Algorithm Example . . . 47

Bibliography . . . 49

3 Complexity and Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots 52

3.1 Introduction . . . 52

3.2 Literature . . . 54

3.3 Problem Statement . . . 56

3.4 Complexity . . . 57

3.5 Solution Methodology . . . 59

3.5.1 Label Making Algorthim . . . 60

3.5.2 Heuristic I: Greedy Task Assignment . . . 64

3.5.3 Heuristic II: Greedy Clustering Task Assignment . . . 66

3.6 Computational Results . . . 67

3.6.1 Greedy Task Assignment . . . 69

3.6.2 Greedy Clustering . . . 75

3.6.3 Greedy vs. Greedy Clustering vs. CPLEX Results . . . 82

3.7 Conclusions . . . 86

Bibliography . . . 87

4 Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots with Known Task Times 89

4.1 Introduction 89

4.2 Literature 91

4.3 Problem Statement 93

4.3.1 Model 96

4.4 Solution Methodology 98

4.4.1 Heuristic I: Fast Task Assignment 98

4.4.2 Heuristic II: Task Plus One Algorithm 100

4.5 Computational Results 103

4.5.1 Fast Task Algorithm 105

4.5.2 Task Plus One Algorithm 108

4.6 Conclusions 119

Bibliography 121

5 Conclusions and Future Work **123**

Bibliography 125

List of Tables

2.1 Orbital parameters for the nodes used in all case studies. Nodes are needed for every orbit on which a task or refuelling depot is located. 19
2.2 Summary of all arcs possible for nodes used in case studies. 21
2.3 The number of arcs for runs in the Single vs. Multi-Orbit Case Study. 22
2.4 Factors and levels for the Core Case Study. 23
2.5 Tasks and their associated starting nodes for a 16 task run in the Core Case Study. 24
2.6 When and Where Refuelling Arcs or Sub Task Completion Occur for Figure 2.2. 25
2.7 Solving results for the integer optimal and time limit reached runs for the Core Case Study. 28
2.8 Where tasks were completed versus the starting orbit of the completing servicer. . 30
2.9 Where servicers refuelled versus their starting location (starting node). 31
2.10 Distribution of refuelling events by servicer starting fuel. 32
2.11 Factors and levels for the focused Refueling Case Study. 33
2.12 How refuelling depots affected task completion. 33
2.13 Factors and levels for the focused Single vs. Multi-Orbit Case Study. 34
2.B.1 All Possible Arcs from Node $i = 3$ with Maneuver Types, Time and Fuel Costs . 46
2.B.2 MORSO Core Case Study: Sub Tasks for Task 4 47
2.B.3 Example: Refuelling Arcs for Depot Starting at Node 12 48

3.5.1 Components of labels used in both heuristics. 62
3.6.1 Factors and levels for the test cases. The combination of these factors and levels resulted in $1,860$ test cases which were solved with CPLEX and each of the heuristics. 68

3.6.2 A comparison of the optimality gaps and solving times for Greedy Task Assignment and CPLEX for runs with a 24 hour time horizon. In the last two columns, negative values indicate that, on average, the heuristic outperformed CPLEX for those instances. 71

3.6.3 A comparison of the optimality gaps and solving times for Greedy Task Assignment and CPLEX for runs with a 48 hour time horizon. In the last two columns, negative values indicate that, on average, the heuristic outperformed CPLEX for those instances. 74

3.6.4 A comparison of the optimality gaps and solving times for Greedy Clustering and CPLEX for runs with a 24 hour time horizon. In the last two columns, negative values indicate average amount that the heuristic outperformed (or not) CPLEX for that combination of factors. 80

3.6.5 A comparison of the optimality gaps and solving times for Greedy Clustering and CPLEX for runs with a 48 hour time horizon. In the last two columns, negative values indicate average amount that the heuristic outperformed (or not) CPLEX. 81

3.6.6 Table which counts the number of instances in which CPLEX or a heuristic obtained a better solution and also if there was a tie. We did not break out the heuristic by type because in most cases Greedy Clustering obtained the better solution. We can see that the heuristics performed better than CPLEX with a longer time horizon and that with the shorter time horizon, the heuristics perform as well as CPLEX in many cases. 84

3.6.7 Table with solving times for each methodology. The Greedy Clustering Task Assignment times do not include the pre-processing times. 85

4.4.1 Components of labels used in both heuristics. Fields marked with † indicate new fields added in this work. The other fields existed in [18]. 99

4.5.1 Factors and levels for the test cases. The combination of these factors and levels resulted in 1860 test cases which were solved with CPLEX and each of the algorithms. 104

4.5.2 Results for the Fast Task Algorithm compared with CPLEX for the 24 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. Three runs failed to achieve any solution within the 12 hour solving time for CPLEX. 786/930 cases reached integer optimality and 141/930 cases hit the solving time limit for CPLEX. 107

4.5.3 Results for the Fast Task Algorithm compared with CPLEX for the 48 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. 48 runs failed to achieve any solution within the 24 hour solving time for CPLEX. 417/930 cases reached integer optimality and 465/930 cases hit the solving time limit for CPLEX. 109

4.5.4 Runs for which CPLEX did not find a solution in 24 hours. The optimality gap is that of the Fast Task Algorithm as compared to the reported CPLEX Best Upper Bound. 110

4.5.5 Results for the Task Plus One Algorithm compared with CPLEX for the 24 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. Three runs failed to achieve any solution within the 12 hour solving time for CPLEX. 786/930 cases reached integer optimality and 141/930 cases hit the solving time limit for CPLEX. 112

4.5.6 Runs for which CPLEX did not find a solution in 24 hours. The optimality gap is that of the Task Plus One Algorithm as compared to the reported CPLEX Best Upper Bound. 114

4.5.7 Results for the Task Plus One Algorithm compared with CPLEX for the 48 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. 48 runs failed to achieve any solution within the 24 hour solving time for CPLEX. 417/930 cases reached integer optimality and 465/930 cases hit the solving time limit for CPLEX. 115
4.5.8 Table with solving times for for each methodology. The Task Plus One Algorithm times do not include the pre-processing times. 116

List of Figures

2.1 Node and orbit locations relative to Earth for nodes used in this work. 20

2.2 Movement of refuelling depots or tasks over time. 25

2.3 Histogram of Gap Values. 27

2.4 How the number and fuel capacity of servicers affected task completion. 29

2.5 Where tasks were completed versus the starting orbit of the completing servicer. . 30

2.6 Where servicers refuelled versus their starting location (starting node). 31

2.7 Task completion by network configuration and time horizon. 35

2.A.1 Orbital Parameters 41

3.4.1 Graphical representation of a MORSO network with n tasks. The arc cost is equal to the processing time of the task p_j, for $j = 1,\ldots,n = |J|$ 58

3.6.1 Comparison of the distributions of the non-zero optimality gap values for Greedy Task Assignment (right) and CPLEX (left) with a 24 hour time horizon. 70

3.6.2 A comparison of the objective values for Greedy Task Assignment and CPLEX for runs with 24 Hour time horizon. Greedy Task Assignment matched or outperformed CPLEX in 354/930 cases. (38%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. . . 72

3.6.3 Distributions of non-zero gap values for Greedy Task Assignment (right) and CPLEX (left) with a 48 hour time horizon 73

3.6.4 A comparison of the objective values for Greedy Task Assignment and CPLEX for runs with 48 hour time horizon. The heuristic matched or outperformed CPLEX in 543/930 cases. (58%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 75

3.6.5 Distributions of non-zero gap values for Greedy Clustering (right) 254 and CPLEX (left) 523 with a 24 hour time horizon 77

3.6.6 A comparison of the objective values for Greedy Clustering and CPLEX for runs with 24 Hour time horizon. Greedy Clustering matched or outperformed CPLEX in 420/930 cases. (45%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 78

3.6.7 Distributions of non-zero gap values for Greedy Clustering (right) and CPLEX (left) for the cases/runs with a 48 hour time horizon 78

3.6.8 A comparison of the objective values for Greedy Clustering and CPLEX for runs with 48 hour time horizon. Greedy Clustering matched or outperformed CPLEX in 543/930 cases. (61.6%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 79

3.6.9 Histogram showing the number of times each methodology achieved the fastest solve time. 83

3.6.10 Histogram showing the number of times each methodology achieved the best solution, alone or in a tie. This has more than 1,860 because there were ties for the best solution on several runs. There were 397 three-way ties and 490 two-way ties for the best solution. 83

3.6.11 There were 757 cases in which the method that achieved the fastest solution time also achieved at least a tie for the best objective value solution. Here we see the distribution of those that led in both categories. 85

4.5.1 Distributions of non-zero optimality gap values for the Fast Task Algorithm (right) and CPLEX (left) with a 24 hour time horizon. In the Fast Task Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound. 106

4.5.2 Fast Task Algorithm with a 24-Hour time horizon compared with CPLEX. The Fast Task Algorithm matched or outperformed CPLEX in 399/930 cases (43%) and was within 1 task of CPLEX in 724/930 (77.8%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 108
4.5.3 Distributions of non-zero optimality gap values for the Fast Task Algorithm (right) and CPLEX (left) with a 48 hour time horizon. In the Fast Task Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound. 110
4.5.4 Fast Task Algorithm with a 48-Hour time horizon compared with CPLEX. The Fast Task Algorithm matched or outperformed CPLEX in 495/930 cases (53%) and was within 1 task of CPLEX in 565/930 (60.7%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 111
4.5.5 Distributions of non-zero optimality gap values for the Task Plus One Algorithm (right) and CPLEX (left) with a 24 hour time horizon. In the Task Plus One Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound. 111
4.5.6 Task Plus One Algorithm with a 24-Hour time horizon compared with CPLEX. The Task Plus One Algorithm matched or outperformed CPLEX in 543/930 cases (58.3%) and was within 1 task of CPLEX in 742/930 (79.7%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 113
4.5.7 Distributions of non-zero optimality gap values for the Task Plus One Algorithm (right) and CPLEX (left) with a 48 hour time horizon. In the Task Plus One Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound. 114

4.5.8 Task Plus One Algorithm with a 48-Hour time horizon compared with CPLEX. The Task Plus One Algorithm matched or outperformed CPLEX in 548/930 cases (59%) and was within 1 task of CPLEX in 644/930 (69.2%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better. 116

4.5.9 Histogram showing the number of times each methodology achieved the fastest solve time. 117

4.5.10 Histogram showing the number of times each methodology achieved the best solution. This has more than 1860 because there were ties for the best solution on several runs. 518 resulted in a three-way tie and 541 resulted in a tie for the best solution. 117

4.5.11 Cases in which the method that achieved the fastest solution time also achieved at least a tie for the best objective value solution. 118

List of Published Papers

Chapter 2:

S.E. Sorenson and S. G.Nurre Pinkley, "Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots," Computers and Industrial Engineering, Submitted, (2022).

Chapter 3:

S.E. Sorenson and S. G.Nurre Pinkley, "Complexity and Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots," Journal of Heuristics, Submission Pending, (2022).

Chapter 4:

S.E. Sorenson and S. G.Nurre Pinkley, "Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots with Known Task Times," Optimization Letters, Submission Pending, (2022).

1. **Introduction**

A repair to the Hubble Telescope in December of 1993 is considered to be one of the first instances of a field called On-Orbit Servicing [4]. In the almost 30 years that have passed since that time, on-orbit servicing (OOS) has dramatically increased, most notably in the past decade. The definition of on-orbit servicing used by NASA is, "a wide range of activities spanning fixing, improving, and reviving satellites and refers to any work to refuel, repair, replace or augment an existing asset in space." In the past two years, Northrup Grumman has completed two OOS missions in Geosynchronous Earth Orbit (GEO) [5] to refuel and upgrade two different satellites. GEO is arguably the most valuable orbit because the satellites placed there stay above the same relative position on Earth. OOS missions will continue to grow in number because they can help replenish, refuel, or refurbish satellites in GEO, Mid Earth Orbit (MEO) and Low Earth Orbit (LEO) which are now dead or in need of repair.

This research considers the optimization of these types of missions in space. Throughout this work, we make assumptions about the future of space technologies which do not yet exist, such as the fleet of highly maneuverable, refuellable space robots. We created these space robots with capabilities which do not yet exist to be our servicers and accomplish imagined tasks on satellites which are currently on orbit. We also hypothesize that eventually there will be widespread refuelling depots spread across LEO, MEO, and GEO for a robot servicer fleet to refuel and replenish in between accomplishing OOS tasks. In this vein, we present the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-orbit Servicing (MORSO) problem. This problem determines how to route and schedule a fleet of highly maneuverable space robots to accomplish tasks spanning LEO, MEO, and GEO. The MORSO problem is the basis for the research contained in this dissertation.

In Chapter 2 we present our formulation for a new mixed integer linear program (MILP) optimization model for MORSO which seeks to maximize the weighted number of completed tasks within a set time horizon. The model is complex because the servicers, the tasks, and the refuelling depots are all moving over time. We account for the movement mathematically in the model and present algorithms to create the model parameters and the data sets to represent the moving tasks and refuelling depots. We also present an algorithm to create a network with nodes as fixed locations in space and arcs representing the orbital maneuvers to transit between the nodes. Using this network and data, we explore three case studies using data based on satellites currently on orbit in LEO, MEO, and GEO.

The main contributions in Chapter 2 are: (i) We are the first to consider OOS over multiple orbits with moving tasks and moving refuelling depots; (ii) We formulate a mixed integer linear program to optimize the routing and scheduling of space robots to accomplish OOS; (iii) We develop and demonstrate flexible algorithms for the creation of model parameters which include a multiple orbit network, and a novel way to represent the movement of tasks and refuelling depots; (iv) We create robust case studies based on current operational satellites in LEO, MEO, and GEO and perform extensive computational experiments (v) and present example insights about robot servicers, task completion, and their use of refuelling depots.

In Chapter 3 we examine the complexity of the MORSO problem and prove that this problem with objective seeking to maximize the weighted number of completed tasks within a set time horizon is $NP-$Hard. We make the reduction from the known $NP-$Complete $1 \mid d_j = d \mid w_jU_j$ which is a single machine scheduling problem that seeks to minimize the weighted number of late tasks when all tasks have the same due date [3]. With the complexity established, we then developed two new constructive heuristics for solving the problem. Both methods use a node labeling approach which is based on ideas from Dijkstra's algorithm and an aircraft shortest path routing with refuelling algorithm [1, 2]. Dijkstra's algorithm uses, updates, and tracks node labels in order to find the shortest path though a network. We extend this idea and create, update, and track labels for the nodes in our network to find efficient paths to the moving tasks while also considering the moving refuelling depots. The first algorithm uses a greedy approach to assign robot servicers to tasks and the second algorithm uses a clustering approach to assign robot servicers to the first in a group of tasks. We demonstrate that the heuristics can find near optimal solutions in significantly less time than CPLEX.

The main contributions in Chapter 3 are (i) We prove the MORSO problem with objective seeking to maximize the weighted number of completed tasks is $NP-$Hard; (ii) We develop and demonstrate two constructive heuristics for solving the MORSO problem; (iii) We perform extensive computational experiments on a realistic data set based on active satellites; (iv) We summarize the results of the experiments thereby demonstrating the speed, accuracy, and scalability of our proposed heuristic methods.

In Chapter 4 we continue to examine the MORSO problem.. In Chapter 2, our MILP model considers a task complete as soon as a servicer arrives at a node at the time a task is also at that node. This equates to an instantaneous processing time for the tasks. In some cases, such as a fly-by inspection, the instantaneous task time might be correct, but for more complex service types, this is not reality. Any task processing time would be truly unknown until the servicer arrived at the task location and had completed the negotiation and rendezvous, connecting to the satellite where the task is located. Task processing times are an important part of this problem, and are uncertain. Thus, in this work, we take the first step towards the bigger problem with unknown task times.

We present the first step as a new problem: Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times (MORSO-KTT). MORSO-KTT determines how to best schedule and route a set of space based robotic servicers to complete a set of tasks with known task times on satellites spanning multiple orbits in space. The model is more complex than MORSO because the task and servicer must coincide at the correct location and time and must move through the network together for at least the duration of the processing time ending the service at a time and location different from the start. To account for this, we expand the model by creating sets of arcs for the tasks. Servicers must traverse the entire set of arcs consecutively, in order to complete a task. We also present an algorithm for creating the subtasks and subtask sets which takes the network, tasks, and task processing times as inputs. We significantly modify the constructive heuristics from Chapter 3 to account for the subtask sets and conduct extensive computational experiments, with a case study based on satellites and orbits currently in use today, to compare the modified constructive heuristics with CPLEX.

The main contributions of Chapter 4 are: (i) We present a new MILP formulation for the MORSO-KTT problem; (ii) We present an algorithm for the construction of the task data which includes the subtasks and the process for constructing the set of arcs for each subtask (iii); We develop and demonstrate two constructive heuristics for the solving the MORSO-KTT problem; (iv) We perform extensive computational experiments on a realistic data set based on active satellites and orbits demonstrating the speed and accuracy of our algorithms.

Bibliography

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J.

[2] Kannon, T. E., Nurre, S. G., Lunday, B. J., and Hill, R. R. (2015). The Aircraft Routing Problem with Refueling. *Optimization Letters*, 9(8):1609–1624.

[3] Lenstra, J. K., Kan, A. R., and Brucker, P. (1977). Complexity of Machine Scheduling Problems. In *Annals of Discrete Mathematics*, volume 1, pages 343–362. Elsevier.

[4] NASA (2021). About - Hubble Servicing Missions. https://www.nasa.gov/mission_pages/hubble/servicing/index.html. Last Accessed: July 4, 2022.

[5] Northrup Grumman (2020). Mission Extension Vehicle. https://news.northropgrumman.com/news/releases/northrop-grumman-and-intelsat-make-history-with-docking-of-second-mission-extension-vehicle-to-extend-life-of-satellite. Last Accessed: July 4, 2022.

2. **Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots**

Susan E. Sorenson Sarah G. Nurre Pinkley

Abstract: We present the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing optimization problem which determines how to best route and schedule a fleet of highly maneuverable and refuellable space robot servicers to complete a set of tasks orbiting in space. We formulate this problem as a mixed-integer linear program and seek to maximize the weighted number of completed tasks subject to constraints related to the movements of the space robots, refuelling depots, and tasks. We present and demonstrate algorithms for constructing the network and model using case studies with data based on satellites operating in the Low Earth, Mid Earth, and Geosynchronous Earth Orbits. Our results indicate the benefit of considering multiple orbits and policies related to the number and starting locations of robot servicers and refuelling depots.

2.1 Introduction

Soon after the Hubble Telescope was launched in 1990, scientists discovered a defect in one of its mirrors which made the images it collected useless [16]. This defect motivated one of the earliest realizations of on-orbit servicing (OOS) and in December 1993, NASA astronauts repaired and replaced parts, restoring the operational capability of the telescope [16]. NASA astronauts serviced the Hubble Telescope four additional times over the next 16 years [16]. More recently, Northrup Grumman executed two OOS missions for Intelsat satellites in 2020 and again in 2021 [18]. Their vision is to establish a fleet of servicers in Geosynchronous Orbit to address most any servicing need [18]. Although not yet commonplace, over the past two decades, researchers and engineers have made great strides in the technology which will eventually make OOS the norm, rather than a novel occurrence. We hypothesize that in the not too distant future, corporations will have on-orbit refuelling depots and fleets of refuellable space robots capable of accomplishing satellite servicing missions across Low Earth Orbit (LEO), Mid Earth Orbit (MEO), and Geosynchronous Orbit (GEO). The routing and scheduling of a set of OOS tasks over an upcoming day, week, or month will become a more common and important problem. This work demonstrates the formulation and solvability of this scheduling challenge using a Mixed Integer Linear Program (MILP). We begin the discussion with a review of the OOS technology which relates to the formulation of our problem.

NASA continues developing future concepts and architectures, such as the Lunar Gateway and manned Mars missions which will require significant use of OOS and manufacturing. One of these concepts is the On-orbit Servicing, Assembly and Manufacturing-1, a robotic spacecraft designed to operate in LEO, equipped with the tools, technologies, and techniques needed to extend satellites' lifespans – even if they were not originally designed to be serviced on-orbit [17]. In 2019, as part of a clean space initiative, the European Space Agency (ESA) announced a call for proposals from European companies to develop OOS robots for the safe removal of ESA-owned satellites [11]. We point the reader to Li et al. [14] for a review of over 130 launched or proposed engineering developments related to OOS missions. Additionally, Corbin et al. [4] provides a more recent review of international OOS developments for On-Orbit Servicing, Assembly or Manufacturing.

In the future, as OOS missions grow in number and magnitude, any long term OOS undertaking will require a space based refuelling depot so that on-orbit robot servicers can refuel themselves and conduct On Orbit Refueling (OOR) and OOS of other satellites. In late 2020, NASA awarded several contracts to companies for the development and/or demonstration of space and lunar based fuel depots [15]. An aerospace company based in the United Kingdom, Thales Alenia Space, will build a chemical refuelling station with an expected launch in 2027 [3]. Although fuel depots to support OOS missions are behind OOS technologies, they are undoubtedly in development and eventually will be able to support OOS missions as envisioned in this work.

We present the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-

Orbit Servicing (MORSO) problem which determines how to schedule and route a small fleet of highly maneuverable refuellable space robots to complete a set of OOS tasks spanning multiple orbits in space. We formulate MORSO as a new MILP optimization model which seeks to maximize the weighted number of completed tasks over a set time horizon. This model is complex because the tasks and refuelling depots move through an orbit over time. Our model considers these complexities and we present a novel way to represent the movement of tasks and refuelling depots over time. We present an algorithm to construct a network with arcs representing different types of orbital maneuvers and with this network as an input we present general, flexible algorithms to create data representing the movement of tasks and refuelling depots over time and space. Using these data sets, we demonstrate the model using three case studies.

Main Contributions. The main contributions of this work are as follows: (i) We are the first to consider OOS over multiple orbits with moving tasks and moving refuelling depots; (ii) We formulate a mixed integer linear program to optimize the routing and scheduling of space robots to accomplish OOS; (iii) We develop and demonstrate flexible algorithms for the creation of model parameters which include a multiple orbit network, and a novel way to represent the movement of tasks and refuelling depots; (iv) We create robust case studies based on current operational satellites in LEO, MEO, and GEO and perform extensive computational experiments (v) and present example insights about robot servicers, task completion, and their use of refuelling depots.

The remainder of this work is as follows. In Section 2.2, we summarize literature related to the optimization of different types OOS missions. In Section 2.3, we provide the algorithms to construct the network arcs and to create the data representing the moving tasks and refuelling depots and then present a formal definition of the model. In Section 2.4, we demonstrate the model and algorithms using a case study with data based on the orbital parameters of current satellites and provide operational insights. In Section 2.5, we present conclusions and areas for further study.

2.2 Literature Review

In this section, we summarize the literature relevant to the optimization of the routing and scheduling of autonomous satellites, space robots, to accomplish On-Orbit Servicing (OOS). OOS involves at least two participants, one who does the servicing and another that receives the service. We denote those who do the servicing as a servicer and those who receive the service as a task. Although refuelling is a type of OOS, when we use the term refuelling we refer to an action that a servicer will take at a space based refuelling depot to replenish themselves for the purpose of accomplishing OOS missions which may or may not include the on-orbit refuelling of tasks.

Many works address a class of on-orbit refuelling (OOR) problem in which there is no a priori designated "servicer." Instead, any participant can act as a servicer to any other participant. In these works, the refuelling is the sharing of fuel which is already on board one of the participants and usually occurs at a location away from both participants' starting locations. Dutta and Tsiotras [8] sought to minimize the overall ΔV expended during maneuvers with a greedy random adaptive search procedure and Dutta and Tsiotras [9] also sought to minimize the overall impulse per unit of spacecraft mass (ΔV), fuel, by optimally matching participants based on their fuel levels. Later, Dutta and Tsiotras [10] found a lower bound of the overall ΔV consumption using a network flow optimization. Du et al. [7] used a Mixed Integer Non-linear Programming (MINLP) formulation, with non-linear ΔV costs using the Tsiolkovsky rocket equation, to minimize the ΔV consumed and solved the problem using a Multi-island Genetic Algorithm. Shen and Tsiotras [20] applied a matching algorithm to minimize the mission time for a single orbit peer-to-peer refuelling problem. Yu et al. [22] addressed a similar problem by solving two sub-problems. The first problem optimized assignments to equally distribute the fuel across all participants and the second minimized the overall ΔV cost. In all of these works, the focus was on a single orbit OOR without any designated servicers, using fuel which is already on board the participants. In our work, we have multiple designated servicers, which can refuel and replenish at orbiting fuel depots located across multiple orbits in our network.

Some authors address problems with a single designated servicer which must complete a set of OOR or OOS tasks. These problems also focus on the minimization of time and/or ΔV, but differ in how the model is formulated. Alfriend et al. [1] addressed a single orbit, single servicer, OOR problem as a Travelling Salesman Problem (TSP) while Bourjolly et al. [2] and [12] used a Vehicle Routing Problem (VRP) formulation to model a multi-orbit single servicer, OOS problem. Zhang et al. [24] considered a multi-objective optimization model, seeking to simultaneously minimize the ΔV and mission time while routing a single servicer to tasks in multiple orbits within the same orbital band. [28] sought to determine the refuelling order, the optimal refuelling time, and the optimal orbital transfers while also minimizing the ΔV and mission time. Yu et al. [23] optimized the scheduling of a single servicer to multiple tasks in GEO, using a multi-objective optimization model seeking to minimize the fuel used, maximize the number of refuelling events, and maximize the sum of the weights of the tasks to be completed. Zhao et al. [26] used a MINLP model, with non-linear ΔV costs, to address an OOR problem with a single servicer in which the tasks requiring fuel moved to a servicing area for the OOR. These works all had only one servicer to accomplish the OOR and OOS tasks while minimizing the time and/or ΔV. Our work has multiple servicers operating in multiple orbital bands and we seek to maximize the weighted number of completed tasks.

Our work is the first to route and schedule multiple refuellable servicers across multiple orbital bands to accomplish multiple tasks moving over time. Our work is unique in that the servicers, the tasks, and the refuelling depots are all moving throughout the network over time. Zhou et al. [27] formulated a time-fixed OOR problem with multiple refuellable servicers based only in GEO. Although the servicers were moving the tasks were not and the model is not changing over time. Daneshjou et al. [6] examined a time-fixed, single orbit, multiple servicer OOS problem using a multi-objective optimization model to determine optimal servicer orbital parameters to minimize the ΔV and time used. Zhang et al. [25] used a location routing problem to solve an OOR problem with multiple refuelling depots and multiple servicers, but their problem was in a single orbit and also time-fixed. Hudson and Kolosa [13] addressed a single orbit multiple servicer OOS problem in GEO as a moving task TSP looking to maximize lifetime profit. Sarton du Jonchay et al. [19] examined a single orbit multiple servicer OOS mission using a strategic level MILP model with decisions about servicer types and servicing tools. Although their work is not time-fixed, they focus on strategic decisions to minimize costs and do not have moving tasks and refuelling depots. With the literature summarized, we proceed by formalizing the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem notation and mathematical model.

2.3 Problem Statement and Methodology

We seek to solve the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem by determining the maximum number of weighted tasks which can be accomplished within a set time horizon. A novel aspect of the problem is that the tasks, servicers and fuel depots are all are orbiting over time. We represent several orbits using a network and optimize the routing, scheduling, and refuelling of space robot servicers through the network over time to complete tasks. We proceed by explaining the network, tasks, and robot servicers and present algorithms for the construction of each. We then define the decision variables, parameters and sets, and model.

2.3.1 Problem Formulation and Algorithm Development

We consider a set of orbits and represent this set as a connected network of nodes, N and arcs, A. We represent each orbit with a subset of the nodes. We represent the nodes as stationary locations in space thus we treat the orbital parameter of true anomaly as a fixed value, but in convention it would indicate the position on the orbit that an object occupies at a given time. We connect a node i with another node j with an arc (i, j) if an orbital maneuver is possible. There are many maneuvers possible. We proceed by describing the four maneuvers we considered for our network. However, we note and emphasize that the model is suitable for networks with arcs representing other possible maneuvers which are outside of the scope of this work.

We add directed arcs to A to represent orbital maneuvers between nodes which share and do not share the same orbit. We add a directed arc (i,j) for an *orbiting maneuver* between nodes i and j which share the same orbit and are adjacent in the rotation direction of the orbit. When nodes i and j share the same orbit but are not adjacent in the rotation direction of the orbit, we add a directed arc (i,j) for a *phasing maneuver*. When nodes i and j are in different orbits, differ by 180° in their stationary true anomaly value, and are on the same plane and we add directed arc (i,j) as a *Hohmann transfer*. Finally, we add a directed arc (i,j) for a *Hohmann transfer with an inclination change* when nodes i and j are in different orbits, differ by 180° in their stationary true anomaly value, and are not on the same plane (inclination). For each arc $(i,j) \in A$, we associate known input parameter values. We denote τ_{ij} and ϕ_{ij} as the time and fuel (ΔV) needed to move along arc (i,j), respectively. We denote Ψ_{ij} as the per time period fuel needed to traverse arc (i,j) where $\phi_{ij} = \Psi_{ij}\tau_{ij}$. In Algorithm 1 we present our general pseudocode which dictates the construction of arcs in a network, given a set of nodes and their associated orbital parameters. Our algorithm only includes the four types of orbital maneuvers we considered, but can be modified to fit orbital maneuvers of any type. To consider other maneuvers, the the Ψ_{ij} and τ_{ij} values for each arc (i,j) are needed. To further aid in the explanation of the Arc Creation Algorithm, we provide a detailed example in 2.B.1. For this work, we assume the time and ΔV costs for servicers to rendezvous with their tasks are 0, however, these values could be included. Note, when we use the term fuel, we are referring to ΔV, or the scalar amount of velocity required for a spacecraft of unit mass to accomplish a maneuver. For this reason, throughout this paper we use fuel and ΔV interchangeably.

Over time, through the network, tasks, servicers, and fuel depots are all moving. We proceed by describing the novel aspect of our work in which we model the movement of tasks, servicers, and fuel depots over time and space. We consider a discrete, finite time horizon, T, over which tasks are to be completed. We discretize this time horizon into distinct time intervals from $t = 0, \ldots, |T|$.

We denote B as the set of tasks where each task $v \in B$ is prioritized by a weight w_v. Be-

Algorithm 1 Arc Creation Algorithm

```
1:  Create arc set $A = \emptyset$.
2:  Input $\Delta t$ as the time period length, $N$ as the node set, and $T$ as the time horizon.
3:  **for** $i, j \in N,\ i \neq j$ **do**
4:      Set $\Delta\theta =$ the angle between nodes $i$ and $j$.
5:      **if** $i$ and $j$ are in the same orbit and are adjacent in rotation direction of orbit **then**
6:          Create $(i, j)$ representing an *orbiting maneuver*.
7:          Set $P$ as the period of orbit for nodes $i$ and $j$.
8:          Set $\tau_{ij} = \frac{\Delta\theta}{360} \times \frac{P}{\Delta t}$, $\phi_{ij} = 0$, $\Psi_{ij} = 0$.
9:      **else if** $i$ and $j$ are in the same orbit and are **not** adjacent in rotation direction of orbit **then**
10:         Create $(i, j)$ representing a *phasing maneuver*.
11:         Calculate the $\tau_{ij} =$ time and $\phi_{ij} = \Delta V$ as shown in 2.A.1.
12:     **else if** the inclination of $i$ = inclination of $j$ **AND** the semi-major axis of $i \neq$ semi-major
        axis of $j$ **AND** $\Delta\theta = 180$ **then**
13:         Create $(i, j)$ representing an *Hohmann transfer*.
14:         Calculate the $\tau_{ij} =$ time and $\phi_{ij} = \Delta V$ as shown in 2.A.1.
15:     **else if** $i \neq j$ **AND** the inclination of $i \neq$ inclination of $j$ **AND** the semi-major axis of $i \neq$
        semi-major axis of $j$ **AND** $\Delta\theta = 180$ **then**
16:         Create $(i, j)$ representing a *Hohmann transfer with an inclination change*.
17:         Calculate the $\tau_{ij} =$ time and $\phi_{ij} = \Delta V$ as shown in 2.A.1.
18:     **end if**
19:     **if** $\tau_{ij} \leq T$ **then**
20:         Add $(i, j)$ to $A$
21:     **end if**
22: **end for**
23: Return arc set $A$.
```

cause tasks are orbiting, each task is at a different location (node in the network) at different times. Thus, we represent each task $v \in B$ with a set of sub tasks, B_v, where each sub task $k \in B_v$ is represented by a node and a time pair, i.e., (n_{vk}, t_{vk}). This indicates that task v will be at node n_{vk} at time t_{vk}. Thus, for a given task v and its associated starting node and orbit, we first create a sub task $k \in B_v$. Next, we iteratively move in the rotational direction of the orbit along the input set of arcs representing *orbiting maneuvers*. With each new node visited, we create a sub task pair based on the node location and current time calculated based on the traversal time of the orbiting maneuver arc. We continuously proceed with this process until the time horizon is reached. We present explicit details of these steps in our Sub Task Creation Algorithm (see Algorithm 2). We consider task v complete when any subtask $k \in B_v$ is complete. To complete a subtask, a servicer must arrive at node n_{vk} at time t_{vk}. In other words, a servicer must leave any node i in the network along arc (i, n_{vk}) starting at time $t_{vk} - \tau_{in_{vk}}$. To aid in the explanation of the sub task creation, we provide an example in 2.B.2.

Algorithm 2 Sub Task Creation Algorithm

```
1: Input $|T|$ as the time horizon and network $(N, A)$.
2: Input task set $B$ where each task $v \in B$ has a known starting node and orbit.
3: for each task $v \in B$ do
4:     Create sub task set $B_v = \emptyset$.
5:     Set $current\_node$ = starting node.
6:     Set $current\_time = 0$.
7:     while $current\_time \leq |T|$ do
8:         Add sub task to $B_v$ associated with the ($current\_node$, $current\_time$).
9:         Set $next\_node$ = to the next node on the orbit in the rotational direction of the orbit.
10:        Set $current\_time = current\_time + \tau_{current\_node\ next\_node}$.
11:        Set $current\_node = next\_node$.
12:    end while
13: end for
14: Return $B_v$, $\forall v \in B$.
```

To complete the tasks, we consider a set of identical robot servicers, D, which move through the network along arcs $(i, j) \in A$ over time. Each servicer $d \in D$ begins the time horizon at a starting node $s_d \in N$ and we decide which first arc (s_d, j) the servicer moves along. If a task starting node equals s_d for any servicer $d \in D$ then this task is marked complete, and removed from the set of tasks, prior to solving the model. For all other tasks, the servicers must move through the network to complete the tasks. We model the movement of each robot servicer as flow throughout the network which must adhere to flow conservation (i.e., after moving on arc (s_d, j) the servicer can only move along arcs $(j, i) \in A$). Thus, over time, we determine if servicer d starts to move along arc $(i, j) \in A$ starting at time t. We assume that each servicer needs fuel to move throughout the network. Thus, over time, we track the fuel on-board of each servicer with f_{dt} where the maximum fuel each robot servicer $d \in D$ can carry is F_d. Thus, a servicer can only traverse arc (i, j) starting at time t if $f_{dt} \geq \phi_{ij}$.

Orbiting throughout the network are a set of fuel depots where servicers can refuel. Just like the tasks, the fuel depots are moving along an orbit and are located at specific locations at specific times. We assume that refueling requires at least one time period to conduct, thus, instead of associated refueling with node locations, we instead designate some of the orbital maneuver arcs as refueling arcs based on time.

We represent this set of arcs and times indicating where and when refuelling of robot servicers can occur with the set A_t^R. The triplet $(i, j, t) \in A_t^R$ indicates that a robot servicer traversing arc $(i, j) \in A$ starting at t can be refuelled up to F_d while moving with a fuel depot on arc (i, j) between time t and $t + \tau_{ij}$. On the refuelling depots, we assume there are ample locations for all robot servicers to refuel simultaneously. For a given refuelling depot r with an associated starting node and orbit, we designate the refuelling arcs in accordance with Algorithm 3. In this algorithm, we iteratively move in the rotational direction of the orbit, increment time based on the input arc τ_{ij} values, and designate refuelling arcs. We can continue in this manner, creating triplicates of the form (i, j, t) stopping when $t \geq T$. We show this using pseudocode in Algorithm 3. To aid in the explanation of the refuelling arc creation, we provide an example in 2.B.3.

Algorithm 3 Refueling Arc Designation Algorithm

```
1:  Input $|T|$ as the time horizon and network $(N, A)$.
2:  Input refueling depot set $R$ where each $r \in R$ has a known starting node and orbit.
3:  Create refuelling arc set $A_t^R = \emptyset$ for all $t \in T$.
4:  for each depot $r \in R$ do
5:      Set $current\_node = starting\_node$.
6:      Set $next\_node =$ to the next node on the orbit in the rotational direction of the orbit.
7:      Set $current\_time = 0$
8:      while $current\_time \leq |T|$ do
9:          Add arc $(current\_node, next\_node)$ to $A^R_{current\_time}$ as a refuelling arc.
10:         Set $current\_time = current\_time + \tau_{current\_node\ next\_node}$.
11:         Set $current\_node = next\_node$.
12:     end while
13: end for
14: Return $A_t^R$, $\forall t \in T$.
```

2.3.2 Parameters, Sets, and Mathematical Model

With the detailed problem formulation and algorithms presented, we continue with the formal definition of the the decision variables, parameters and sets, and model. The decision variables, parameters, and sets used in the mathematical model were introduced at the start of this work. As we consider this finite time horizon (i.e., one day, one week), we must determine what occurs when $t \geq |T|$. We assume that robot servicers may finish at any node in the network at $|T|$. We model this assumption through the addition of a super sink node E where each node $i \in N$ is connected to E with a directed arc (i,E) with unit traversal time and no fuel requirements. With this design, in the model, we force each robot to be at node E at time $|T|$. We model this problem using a mixed-integer linear programming (MILP) formulation as follows.

2.3.2.1 Parameters/Sets

T :	Set of time periods, where $\lvert T \rvert$ is the last time period
N :	Set of nodes $, i \in N$
E :	Super sink node, $E \in N$
B :	Set of tasks, $v \in B$
Γ :	Set of refuelling depots, $r \in \Gamma$
B_v :	Set of sub tasks associated with $v \in B, k \in B_v$
A :	Set of directed arcs, $(i,j) \in A$
A_t^R :	Set of refuelling arcs at time $t \in T, A_t^R \subset A$
D :	Set of robot servicers, $d \in D$
s_d :	Starting node of servicer $d \in D$
F_d :	Maximum fuel capacity of servicer $d \in D$, in ΔV
w_v :	Weight of task $v \in B$
n_{vk} :	The node location of sub task $k \in B_v$ of task $v \in B$
t_{vk} :	Time period of sub task $k \in B_v$ of task $v \in B$
τ_{ij} :	Time periods to traverse arc $(i,j) \in A$
ϕ_{ij} :	Fuel to traverse arc $(i,j) \in A$, in ΔV
Ψ_{ij} :	Per time period fuel needed to traverse arc $(i,j) \in A$, in ΔV

2.3.2.2 Decision Variables

$$\beta_{vk} = \begin{cases} 1, & \text{if sub task } k \text{ of task } v \text{ is completed, for } v \in B \\ 0, & \text{otherwise} \end{cases}$$

$$f_{dt} = \text{fuel level of robot } d \text{ at time } t; \text{ for } d \in D \text{ and } t \in T$$

$$y_{dijt} = \begin{cases} 1, & \text{if robot servicer } d \text{ initiates move on arc } (i,j) \in A, \text{ at time } t \in T \\ 0, & \text{otherwise} \end{cases}$$

2.3.2.3 Model

$$\max \sum_{v \in B} w_v \sum_{k \in B_v} \beta_{vk} \tag{2.1}$$

$$\text{s.t.} \sum_{k \in B_v} \beta_{vk} \leq 1, \qquad \text{for } v \in B \tag{2.2}$$

$$\sum_{d \in D} \sum_{\substack{i:(i,n_{vk}) \in A \\ t_{vk} - \tau_{in_{vk}} \geq 0}} y_{din_{vk}t_{vk}-\tau_{in_{vk}}} \geq \beta_{vk}, \qquad \text{for } v \in B, k \in B_v \tag{2.3}$$

$$\sum_{\substack{j:(j,i) \in A \\ t-\tau_{ji} \geq 0}} y_{djit-\tau_{ji}} - \sum_{j:(i,j) \in A} y_{dijt} = \begin{cases} -1, & \text{if } i = s_d \text{ and } t = 0 \\ 1, & \text{if } j = E \text{ and } t = |T| \\ 0, & \text{otherwise} \end{cases}$$
$$\text{for } i \in N, d \in D, t \in T \tag{2.4}$$

$$f_{dt} \leq f_{dt-1} - \left(\sum_{(i,j) \in A} \sum_{s=\max\{t-\tau_{ij},0\}}^{t} \left(y_{dijs} * \Psi_{ij} \right) + \sum_{\substack{(i,j) \in A_t^R \\ t-\tau_{ij} \geq 0}} \left(y_{dijt-\tau_{ij}} * F_d \right) \right)$$
$$\text{for } t \in T \setminus |T|, d \in D \tag{2.5}$$

$$f_{d0} = F_d, \qquad \text{for } d \in D \tag{2.6}$$

$$0 \leq f_{dt} \leq F_d, \qquad \text{for } d \in D, t \in T \tag{2.7}$$

$$y_{dijt}, \; \beta_{vk} \in \{0,1\}, \qquad \text{for } d \in D, (i,j) \in A, t \in T, v \in B, k \in B_v \tag{2.8}$$

In Equation, (2.1) we present the objective function seeking to maximize the weighted number of tasks completed. In Constraints (2.2), we ensure that for each task, at most one sub task is completed. In Constraints (2.3), we link the movement of robot servicers with the completion of subtasks. In Constraints (2.4), we balance the flow of robot servicers through the network by ensuring that each robot servicer must start at their designated start node, only leave a node they are at, and must finish at the super sink E (i.e., any node in the network because each node $i \in N$ is connected to E with arcs (i,E)). In Constraints (2.5), we update the fuel on-board of each servicer at each time based on the prior time period fuel level, movement, and refuelling decisions. In Constraints (2.6), we set the starting fuel level of all robot servicers to the maximum amount. In Constraints (2.7), we force the fuel level of each servicer over time to be between 0 and the maximum capacity. In Constraints (2.8), we place the binary restriction on some decision variables. Next, we demonstrate the validity and benefits of the model with a case study.

2.4 Data and Computational Results

In this section, we use case studies to demonstrate and solve the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) Mixed Integer Linear Program (MILP) model to gain insights about the possible strategic and operational decisions of multiple servicer on-orbit servicing (OOS) with refuelling. To deduce these insights, we performed case studies using a data set we created to represent the current active satellites found in Low Earth Orbit (LEO), Mid Earth Orbit (MEO), and Geosynchronous Orbit (GEO) [21]. Using this data set, we varied the number of refuelling depots, robot servicers, the number of tasks, robot servicer maximum fuel capacity, robot servicer starting locations and network configurations. In the following, first, we describe the details of constructing the network used for our case studies. Next, we describe the details, results, and insights of our three case studies investigating general behavior, refueling, and inter orbit movements.

2.4.1 Network Construction

For this work, we conducted three cases studies, each with similar networks in that the nodes for all three case studies were the same but the number and types of arcs were not. The network nodes were chosen based on where our notional tasks were located so that the tasks would intersect. In this section, we present an in depth breakdown of the network for the largest case study, called the *Core Case Study* and then explain the differences between the networks for the two additional case studies.

To create the network for the Core Case Study, we begin by selecting the time horizon and the tasks. Our created dataset has a time horizon of 24 hours and is broken into $T = 96$ distinct time units where one time unit represents 15 minutes. We choose the orbits for the network based on where our tasks are operating. For the cases studies presented here, we had tasks in eight different orbits which span the three orbits closest to Earth. The band closest to Earth, LEO, includes orbits with a semi-major axis of $6,558$ to $8,378$ kilometers. The next band called MEO, includes orbits with a semi-major axis of $8,378$ to $42,158$ kilometers. Finally GEO, is for objects with semi-major axes over $42,158$ kilometers. Next, we describe the specifics of the nodes in each orbital band.

We considered three different circular orbits in the LEO band, denoted $1^L, 2^L$, and 3^L. Each of these orbits have a semi-major axis of 6652.55 km and a period of 90 minutes. We placed nodes 15 minutes (1 time unit) apart on the orbit, so that each orbit has 6 nodes each spaced 60 degrees apart. True anomaly indicates where on an orbit, in degrees, an object is located. Because our nodes are "fixed" locations in space and tasks and servicers move "between" nodes, we treated the true anomaly as a constant position. As an example, using orbit 1^L, the nodes are spaced 60 degrees apart, so we have 6 nodes with "fixed" true anomalies of $0°, 60°, 120°, 180°, 240°, 300°$. The inclination and RAAN values in the LEO orbits were chosen based on where SpaceX Starlink Space-Track.org [21] satellites reside, as if our tasks were on the same orbits as the SpaceX Starlink satellites.

We considered four different circular orbits in the MEO band, denoted $4^M, 5^M, 6^M$, and 7^M. The nodes within each of these orbits have a semi-major axis of $26,610$ kilometers and a period of 720 minutes or 12 hours. We placed nodes 30 minutes (2 time units) apart so that each orbit has 24 nodes. There are 96 total nodes in MEO on four different circular orbits. The inclination and RAAN values in the MEO orbits were chosen based on where the U.S. Global Positioning System (GPS) Space-Track.org [21] satellites reside, as if our tasks were located on the same orbits.

Lastly, we considered one GEO orbit, denoted 8^G which has a semi-major axis of $42,241$ kilometers and a period of 1440 minutes, or 24 hours. We placed nodes 30 minutes (2 time units) apart resulting in 48 nodes. The node locations on the orbit were determined as they were for LEO and MEO. The inclination and period in the GEO orbit were chosen based on where the U.S. Wideband Global SATCOM 9 (WGS) Space-Track.org [21] satellite resides, as if our tasks were located on the same orbit. Figure 2.1 shows a visualization of all nodes and their associated orbits relative to Earth.

Table 2.1: Orbital parameters for the nodes used in all case studies. Nodes are needed for every orbit on which a task or refuelling depot is located.

Orbit ID	Nodes Start,…,End	Semi-Major Axis (km)	Inclination	RAAN	Based on	Period (min)	Separation (min/deg)
1^L	$1,\ldots,6$	6652.550	52.986	290.269	Starlink	90	15.0/60.0
2^L	$7,\ldots,12$	6652.550	52.989	120.183	Starlink	90	15.0/60.0
3^L	$13,\ldots,19$	6652.550	53.000	64.290	Starlink	90	15.0/60.0
4^M	$19,\ldots,42$	26610.000	0.000	0.000	GPS	720	30.0/15.0
5^M	$43,\ldots,66$	26610.000	55.000	300.000	GPS	720	30.0/15.0
6^M	$67,\ldots,90$	26610.000	55.000	45.000	GPS	720	30.0/15.0
7^M	$91,\ldots,114$	26610.000	55.000	225.000	GPS	720	30.0/15.0
8^G	$115,\ldots,162$	42241.000	0.000	0.000	WGS	1440	30.0/7.5

$\#^i$ Indicates the orbit ID number and orbit location where L represents Low Earth Orbit, M represents Mid Earth Orbit, and G represents Geosynchronous Orbit.

In Table 2.1, we show the parameters of the orbits and nodes for all case studies. The first column is the orbit ID and the second column shows the nodes associated with each orbit. The middle three columns are the orbital parameters for each orbit. Eccentricity is not included because all orbits were assumed circular. However, we note that this is not a necessary assumption for the model outlined in Section 2.3. Instead, this is an assumption we made when creating the case study data set. The fourth column indicates on which real-world satellite the parameters were based. Finally, the last two columns show the period and spacing of the nodes on that orbit. We constructed equidistant nodes for each orbit so that the time between two nodes which are adjacent in the rotation direction of the orbit is 1 or 2 time units, 15 or 30 minutes (Δt), respectively. To accompany Table 2.1, in Figure 2.1, we present a visualization of the nodes relative to

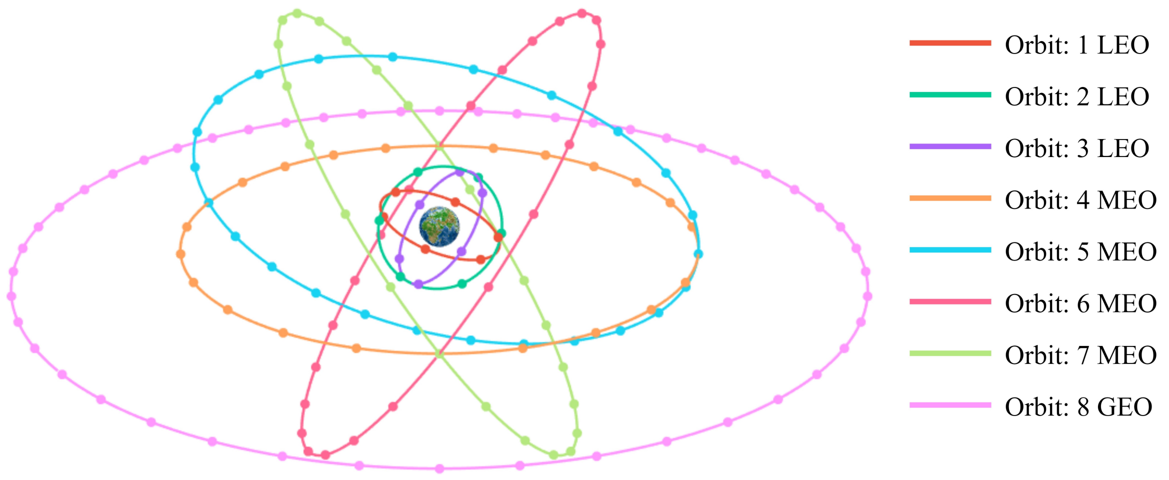

Figure 2.1: Node and orbit locations relative to Earth for nodes used in this work.

Earth. With the node locations for the case study established, we proceed by explaining our arc construction.

To construct the network arcs for the Core Case Study, we considered four different types of orbital maneuvers: *Orbiting Maneuvers, Hohmann Transfers, Phasing Maneuvers,* and *Hohmann Transfers with an inclination change*. With the set of nodes (N) and the time period length (Δt) as inputs, we constructed the network arcs using Algorithm 1. For the Core Case Study, we consider $T = 24$ hours. In Algorithm 1 we exclude any arcs with a traversal time $\geq T$. After following the algorithm using our set of 162 nodes, the network had $|A| = 1,381$ total directed arcs for the 24 hour time horizon. Additionally, each node is connected to a sink node for $1,381 + 162 = 1,543$ total arcs.

In Table 2.2, we show the arcs for the set of nodes used in our case studies. The arcs are organized by the direction and the type of maneuver. For the Core Case Study, any arcs with a time cost which exceeded the time horizon of 24 hours were excluded.

In Table 2.2, the first column indicates the direction of the maneuvers as a change in the orbit altitude. The next column indicates the type of maneuver, we only considered four types of maneuvers in this work, but any number of maneuver types could be considered in the network construction as only the time and ΔV for each arc are needed. The next two columns indicate the starting and ending orbital bands. The fifth column, "Count" shows the number of arcs which fall into that category. The "Time" columns show the minimum, mean, and maximum traversal times for arcs in that category. The "ΔV" columns show the minimum, mean, and maximum ΔV costs for arcs in that category. The final columns show the minimum, mean, and maximum ΔV per hour costs for arcs in that category. Because the weight of the robot servicer is unknown, ΔV is used throughout the case study, but this could easily be adjusted once the weight and other properties of the chosen robot servicer are known. The time and ΔV costs shown were calculated as outlined in [5].

Table 2.2: Summary of all arcs possible for nodes used in case studies.

Direction	Maneuver Type	Starting Orbit	Ending Orbit	Count	Time (*hours*)			Δ V			ΔV per Hour		
					Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Ascending	Hohmann	MEO	GEO	24	12.00	12.00	12.00	1.54	1.54	1.54	0.13	0.13	0.13
	Hohmann w/Inclination Change	LEO	MEO	72	6.00	6.00	6.00	7.42	7.64	8.24	1.24	1.27	1.37
		LEO	GEO	18	12.00	12.00	12.00	7.02	7.02	7.02	0.59	0.59	0.59
		MEO	GEO	72	12.00	12.00	12.00	4.46	4.46	4.46	0.37	0.37	0.37
Descending	Hohmann	GEO	MEO	24	6.00	6.00	6.00	1.63	1.63	1.63	0.27	0.27	0.27
	Hohmann w/Inclination Change	MEO	LEO	72	0.75	0.75	0.75	3.32	3.54	4.14	4.43	4.71	5.52
		GEO	LEO	18	0.75	0.75	0.75	2.15	2.15	2.15	2.87	2.87	2.87
		GEO	MEO	72	6.00	6.00	6.00	3.63	3.63	3.63	0.61	0.61	0.61
Same	Orbit	LEO	LEO	18	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.00	0.00
		MEO	MEO	96	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00
		GEO	GEO	48	0.50	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00
	Phasing	LEO	LEO	75	2.00	3.74	5.00	0.74	1.83	2.39	0.37	0.49	0.54
		MEO	MEO	2116	13.00	28.54	46.00	0.10	0.84	1.29	0.01	0.03	0.03
		GEO	GEO	2209	25.00	56.55	94.00	0.04	0.65	1.04	0.00	0.01	0.01
Overall				4934	0.25	37.03	94.00	0.00	0.98	8.24	0.00	0.14	5.52

As eluded to earlier, we also conducted two smaller case studies. We examined the effect of refuelling in the *Refuelling Case Study* and the effect of inter-orbit transfers in the the *Single vs. Multi-Orbit Case Study* on the number of tasks completed. In later sections, we present the detailed factors and levels for each of these case studies, but now we will talk about the network differences.

The network for the Refuelling Case Study used the same node set, N, as the Core Case Study, but used a 48 hour time horizon. Thus, any arcs with a traversal time, $\tau_{ij} \geq 48$ hours were excluded. After following the algorithm using our set of 162 nodes, the network had $|A| = 3,590$ total directed arcs for the 48 hour time horizon plus the arcs connected to a sink node for $3,590+$

$162 = 3,752$ total arcs. Additionally, one half of the runs allowed refuelling arcs in the network while the other half did not allow refuelling arcs.

The network for the Single vs. Multi-Orbit Case Study also used the same node set, N, as the Core Case Study, but had multiple different time horizons of $24, 36, 48, 72,$ and 96 hours. In the same manner as described previously, for each run, the network was reconfigured to exclude arcs which exceeded the designated time horizon for that run. In addition, for the single orbit runs we exclude the arcs representing Hohmann transfers and Hohmann transfers with an inclination change and only considered orbiting and phasing maneuvers. All runs in the Single vs. Multi-Orbit Case Study allowed refuelling. The number of arcs for the Single vs. Multi-Orbit Case Study are shown in Table 2.3 broken down by the network configuration and time horizon. In all case studies which allow inter orbit transfers, there are 48 Hohmann transfer arcs, 324 Hohmann transfer with an inclination change arcs, and 162 orbiting arcs.

Table 2.3: The number of arcs for runs in the Single vs. Multi-Orbit Case Study.

Network Configuration	Time Horizon (*hours*)	Number of Arcs
Multi-Orbit	24	1381
	36	2582
Single Orbit	24	1009
	48	3218
	72	3986
	96	4934

2.4.2 Core Case Study

2.4.2.1 Core Case Study: Test Design and Data Generation

In the previous section, we covered the network generation for each of our three case studies. With the network established, we will now focus on the Core Case Study and the details of how the data for the tasks, subtasks, and refuelling arcs were created. We begin with the test design as it contains the information needed for data generation.

To generate the data for the tasks, subtasks, and refuelling arcs, we need to know the number and starting location of the tasks, the number and starting locations for the refuelling depots, and the number, fuel capacity, and starting location of the robot servicers. All of this information for each of the case studies is in the case study test design. For this case study, the Core Case Study, we examined the largest set of factors and levels hoping to draw big picture insights. In Table 2.4, we show the factors and levels used in the Core Case Study. We considered between one and five fuel depots and between one and five robot servicers. For all case studies, the refuelling depot starting location, was some combination of the following: $12, 32, 80, 125, 150$. These nodes are in orbits $2^L, 4^M, 2^M,$ and 8^G. We also assumed that robot servicers always started at a fuel depot. Thus, these are also the values for s_d. When the number of robot servicers equals the number of depots, one servicer started at each depot. We enumerated all cases for one to five depots and their starting locations and this resulted in 31 cases for the combined factor of Fuel Depot Starting Location. We also considered the robot servicer fuel capacity and placed one or two tasks per orbit which resulted in 8 or 16 tasks. We continue with the generation of the data for the refuelling arcs and subtask locations.

Table 2.4: Factors and levels for the Core Case Study.

Factor	**Levels**
Fuel Depot Starting Locations	31 cases[†]
Number of Robot Servicers	1, 2, 3, 4, 5
Robot Servicer Fuel Capacity (ΔV)	10, 15, 25
Number of Tasks	8, 16

[†] The *Fuel Depot Starting Locations* factor is a combination of factors for the number of servicers, number of depots, and the fuel depot starting locations.

Now with the parameters for the refuelling depot(s) from the test design, starting location and orbit of the refuelling depots, the time horizon ($|T|$), the set of nodes (N), and the set of arcs (A) we have all the inputs to use Algorithm 3 to determine the elements of A_t^R. Figure 2.2 presents a visual example of refuelling depot movement which is implemented in Algorithm 3. Next we generate the subtasks from the tasks. In our set of arcs A there are 162 arcs which are orbiting maneuvers. Depending on the number of refuelling depots and the time, every arc on an orbit that has a refuelling depot becomes a refuelling arc.

For all case studies, we considered runs with either $|B| = 8$ or $|B| = 16$ tasks indicating one or two tasks per orbit. We numbered the tasks sequentially and assigned a unit weight to each task. The locations of tasks are moving over time, thus given the location of a task (i.e., the node closest to its starting position) at the start of the time horizon, we can approximate where and when the task is in our network at a given time. When we had two tasks on the same orbit, we started them at nodes on opposite sides of the orbit. In application, the starting node for a task should be the node closest to its location on the orbit at the start of the time period. With all of the input parameters (the time horizon ($|T|$), the set of nodes (N), the set of arcs (A), the task set (B), and the tasks ($v \in B$) with their starting nodes and orbits for Algorithm 2 we generated the subtasks, ($B_v, \forall v \in B$). Figure 2.2 and Table 2.6 present a visual and tabular example of task movement.

Table 2.5: Tasks and their associated starting nodes for a 16 task run in the Core Case Study.

Orbit	Tasks	Starting Nodes
1^L	1, 9	1, 4
2^L	2, 10	7, 10
3^L	3, 11	13, 16
4^M	4, 12	19, 31
5^M	5, 13	43, 55
6^M	6, 14	67, 79
7^M	7, 15	91, 103
8^G	8, 16	115, 139

$\#^i$ Indicates the orbit ID number and orbit location where L represents Low Earth Orbit, M represents Mid Earth Orbit, and G represents Geosynchronous Orbit.

In Figure 2.2, we depict the movement over time of a task or fuel depot moving through an orbit between times t to $t+2$. For this figure, we assume that the task (or fuel depot) is at node 2 at time t and the nodes in this orbit are $\Delta t = 15$ minutes (1 time unit) apart. On the far left we depict a task's subtask at node 2 or a refuelling depot at node 2 at time t resulting in a refuelling arc on $(2,3)$ at time t. In the middle we see the same subtask now at node 3 at time $t+1$ or a refuelling arc on $(3,2)$ at $t=t+1$. On the right, as time proceeds, the same subtask is now at node 4 or a refuelling arc at time $t+2$ on $(4,5)$. In Table 2.6, we present a tabular representation of Figure 2.2 indicating where tasks could be completed or the robot servicers might refuel, depending on if the figure represents depots or tasks.

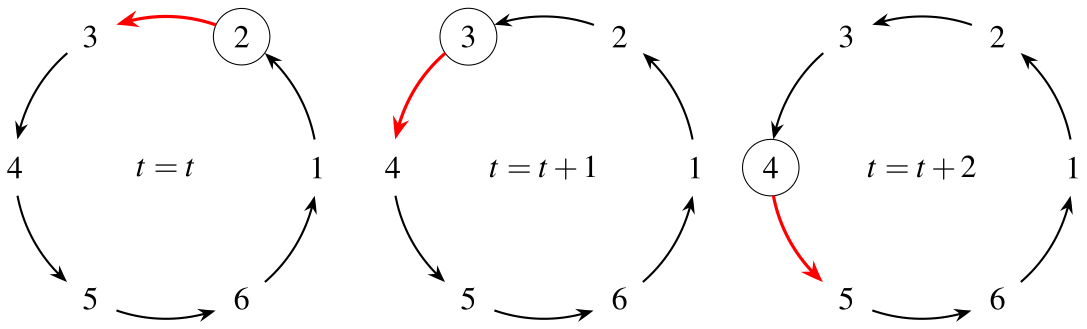

Figure 2.2: Movement of refuelling depots or tasks over time.

Table 2.6: When and Where Refuelling Arcs or Sub Task Completion Occur for Figure 2.2.

Activity Possible	Where	When
If the figure represents depots		
Refuelling	Arc $(2,3)$	$t=t$
Refuelling	Arc $(3,4)$	$t=t+1$
Refuelling	Arc $(4,5)$	$t=t+2$
If the figure represents tasks		
Sub Task Location	Node 2	$t=t$
Sub Task Location	Node 3	$t=t+1$
Sub Task Location	Node 4	$t=t+2$

With the network, parameters and sets created for the Core Case Study, we used the test design in Table 2.4 to create a full factorial design with 930 runs ($31 \times 5 \times 3 \times 2 = 930$). All of these runs were conducted on a High Performance Computer (HPC) using IBM Decision Optimization for CPLEX (DOcplex). IBM DOcplex is an optimization software package written for use in Python. Next we present the results and insights from the 930 runs in the Core Case Study.

2.4.2.2 Core Case Study: Results and Insights

Now we present insights about solving statistics, servicer behaviors, refuelling, and network configuration for the Core Case Study. In every case study, the goal of the optimization model was to maximize the weighted number of completed tasks. In all results, when we say *completed tasks*, we are actually referring to the *weighted number of completed tasks*. This is interchangeable in our case study because all of our tasks had an equal weight of one. During the analysis, we sometimes examine and refer to another response variable: *proportion of weighted tasks completed*. This proportion allows a discussion of task completion without categorizing the results based on the number of tasks, because maximizing the completed number of tasks also maximizes this proportion. For each run, we calculated this value as follows: Proportion of Weighted Tasks Completed $= \frac{\text{Objective Function Value}}{\text{Number of Tasks}}$. For the Core Case Study, this proportion varied from $0.12 - 1.00$, with an overall mean of 0.59 and a standard deviation of 0.26. This means that over all 930 runs, between $12 - 100\%$ of the tasks were completed and that on average 59% of the tasks were completed.

To further examine the response, we fit a regression model with the the main effects and the two factor interactions of our design factors shown in Table 2.4. The most influential factors, by far, on the proportion of weighted of tasks completed are the servicer starting fuel and the number of servicers, accounting for over 80% of the variability in the response. The completed tasks (objective values) results are shown in Table 2.7 grouped by the number of servicers and the fuel capacity as these were the two most influential factors.

Next, we will cover the solving statistics for the Core Case Study. We begin with a brief explanation of terminology and then proceed with the details of the solving statistics. The goal when solving any optimization model to find an *optimal* solution. For our problem, this translates to finding a solution for which the servicers complete as many tasks as possible. When solving our problem, the software stopped when it found an integer optimal solution to the problem, reached the time limit, or ran out of memory.

Of the 930 runs, 523 reached integer optimality. When solving, we set a 12 hour solv-

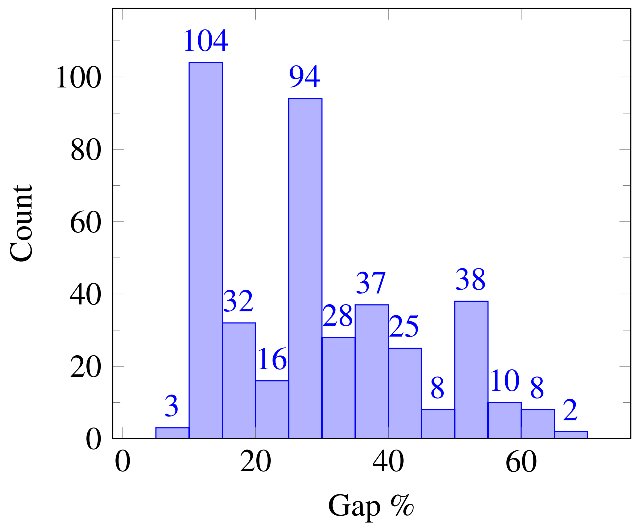

Figure 2.3: Histogram of Gap Values.

ing time limit for each of the 930 runs. If the solver could not find an optimal solution before the time limit was reached, the software reported the best objective function value at that time along with the *gap*. Given an optimal solution, the gap is a value which indicates that the current solution is off by no more than this percentage. Of the 930 runs, 404 reached the 12 hour time limit with an average gap of 28.4% and a range of $6.25-69.1\%$. This means that the solutions obtained at the end of the 12 hours, in the worst case, differed from an optimal solution by $6.25-69.1\%$. A visualization of the gap distribution for these runs is shown in Figure 2.3. Finally, there were 3 runs which ran out of memory on the high performance computer and failed to solve. The HPC has another partition which is designated *high-memory*. We ran the 3 runs with a 12 hour time limit on the *high-memory* partition and they all reached the 12 hour time limit. The combined solving results for all runs are shown in Table 2.7.

Each row of the table summarizes the outcome of the 31 runs for that set of factor levels, with the exception of the runs which ran out of memory and did not solve on the HPC. For the factor combinations which ran out of memory, those rows are marked with a number in the exponent which indicates the number of runs which ran out memory. The column *Mean 12 Hour Gap (%)* shows the averaged gap for each row. When the value in this column is zero, this means that all 31 runs resulted in an integer optimal solution. When the value in this column is not zero,

Table 2.7: Solving results for the integer optimal and time limit reached runs for the Core Case Study.

Factors			Using CPLEX 12.10				Objective Value		
Number of Tasks	Number of Servicers	Fuel Capacity	Integer Optimal Solution	Reached 12 Hours	Mean 12 Hour Gap (%)	Mean Solve Time (hours)	Min	Mean	Max
8	1	10	31	0	0.00	0.16	1	1.61	2
	1	15	31	0	0.00	0.12	2	2.48	3
	1	25	31	0	0.00	0.05	3	3.00	3
	2	10	13	18	28.89	8.19	2	3.06	4
	2	15	27	4	3.38	4.31	4	4.61	5
	2	25	24	7	5.00	4.16	5	5.00	5
	3	10	1	30	41.86	11.79	3	4.52	6
	3	15	5	26	16.83	10.95	6	6.32	7
	3	25	8	23	9.27	10.58	6	6.90	7
	4	10	1	30	28.23	11.63	4	5.71	7
	4	15	18	13	5.24	5.64	7	7.52	8
	4	25	29	2	0.81	1.14	7	7.90	8
	5	10	7	24	17.34	10.82	5	6.58	8
	5	15	31	0	0.00	0.70	7	7.97	8
	5	25	31	0	0.00	0.34	7	7.97	8
16	1	10	31	0	0.00	0.05	2	2.58	3
	1	15	31	0	0.00	0.04	3	3.52	4
	1	25	31	0	0.00	0.03	4	4.00	4
	2	10	16	13^2	16.60	8.02	4	4.84	6
	2	15	27	4	1.70	3.72	6	6.52	7
	2	25	30	1	0.36	0.72	8	8.00	8
	3	10	2	28^1	24.74	11.35	5	6.94	9
	3	15	25	6	4.56	4.35	9	9.39	10
	3	25	28	3	2.25	2.71	9	10.48	12
	4	10	3	28	40.98	11.74	6	7.94	10
	4	15	6	25	22.98	10.14	10	11.00	12
	4	25	5	26	16.98	10.60	10	12.45	14
	5	10	0	31	45.98	12.00	6	8.58	11
	5	15	0	31	26.65	12.00	9	11.71	13
	5	25	0	31	13.52	12.00	11	13.81	15

$^\#$ Indicates the number of runs which ran out of memory.

this means that at least one of of the 31 reached the 12 hour time limit when solving. Each row shows the counts of the runs which were integer optimal or reached the time limit of 12 hours. The *Mean 12 Hour Gap (%)* includes zero values in the average for those runs that reached integer optimality. This same logic follows for the *Mean Solve Time (hours)* column. For the rows which had runs which reached the 12 hour time limit, the solve time mean includes 12 hours in

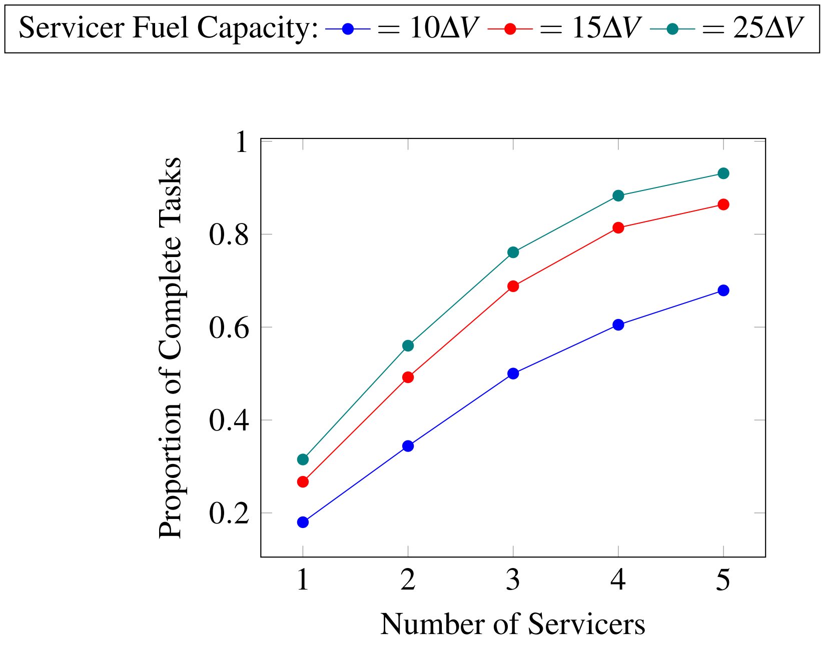

Figure 2.4: How the number and fuel capacity of servicers affected task completion.

the average.

In addition to solving insights, we can show the number and configuration of the robot servicers impacts the number of completed tasks using the Objective Value columns. As would be expected, as the number of servicers increases so does the number of completed tasks. We examined this further using Figure 2.4 and can see that the slopes of the lines decrease as the number of servicers increase indicating a diminishing return on the number of completed tasks. Figure 2.4 also illustrates the decreasing impact of increasing the servicer fuel capacity. The increase in the objective value when increasing the servicer fuel capacity from 10 to 15 is almost twice that of the increase when the fuel capacity increases from 15 to 25, illustrating the decreasing impact of increasing the servicer fuel capacity.

In addition to solving statistics and the impact of the factors on the response variable, we also examined robot servicer movements. In the 930 runs of the main test design, there were $2,790$ total servicers and $11,160$ tasks. On average, each servicer completed 2.3 tasks, refuelled 2.8 times, and used 13.9 ΔV. The number of tasks completed by servicers ranged from $0-4$. There were 3 servicers that completed no tasks and all of these cases occurred when there were 5 servicers available. These results are intuitive because when there are less servicers, the servicers available must complete more tasks. No servicer completed more than 4 tasks in this 24-hour time horizon indicating that it might be better to have more than one servicer when then are more than four tasks to be completed.

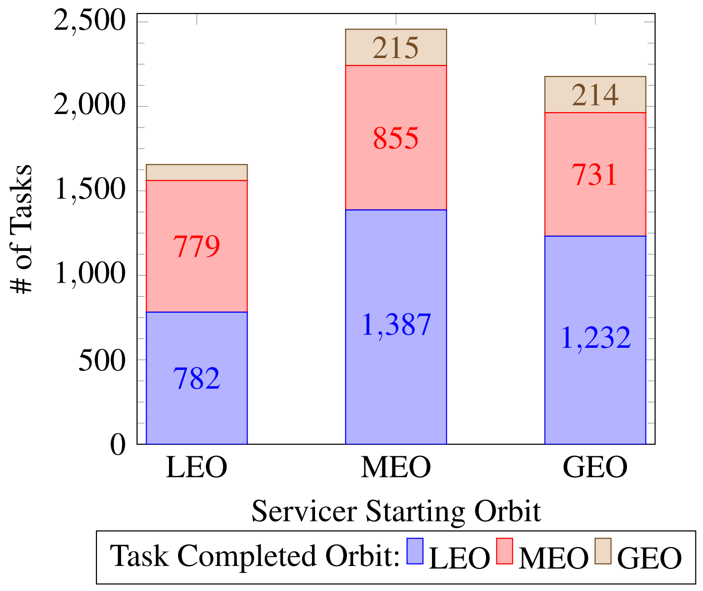

Figure 2.5: Where tasks were completed versus the starting orbit of the completing servicer.

Table 2.8: Where tasks were completed versus the starting orbit of the completing servicer.

	Servicer Starting Orbit		
Refuelling Orbit	LEO	MEO	GEO
LEO	782	1387	1232
MEO	779	855	731
GEO	95	215	214

Another interesting observation from this case study is that where servicers started did not influence where they completed tasks. We examined the orbits where tasks were completed versus where the completing servicer started. Figure 2.5 shows where tasks were completed as compared to where the completing servicer started and Table 2.8 shows a tabular representation. The plot shows the results for all completed tasks in the Core Case Study. The colors indicate the proportion of all tasks which were completed in that orbit. We know from Table 2.2 that the maneuvers to and within LEO use much less time allowing the servicers to move more and perhaps complete more tasks. All servicers completed the most tasks in LEO, followed by MEO and GEO and this was true across all servicer starting orbits indicating that no matter where a servicer starts, they proceed to lower orbits to accomplish tasks. The results here motivated the third case study which focused on the impact of inter orbit transfers.

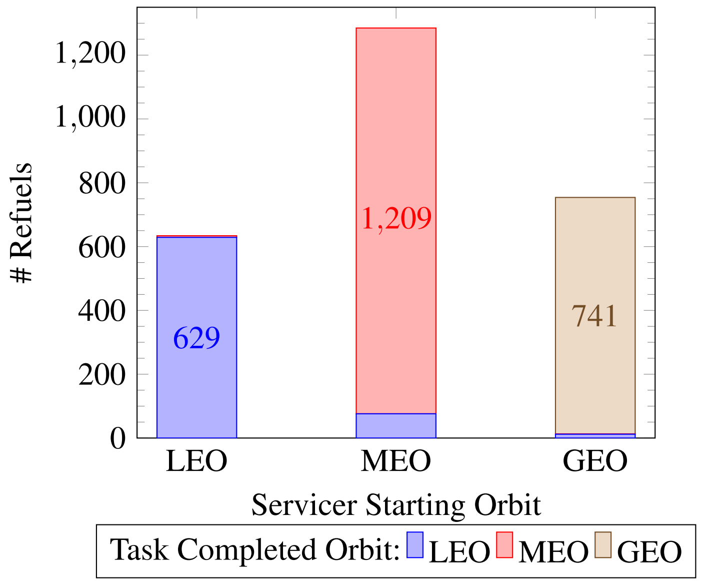

Figure 2.6: Where servicers refuelled versus their starting location (starting node).

Table 2.9: Where servicers refuelled versus their starting location (starting node).

	Servicer Starting Orbit		
Refuelling Orbit	LEO	MEO	GEO
LEO	629	76	12
MEO	5	1209	1
GEO	0	0	741

Although the servicers preferred to work in LEO, they refuelled most often in the orbits where they started. In the Core Case Study, there were 2673 refuelling events. A refuelling event occurred when the fuel level of a servicer increased from one time period to the next while travelling on a refuelling arc. As shown in Section 2.3, in our model, there is no penalty for refuelling, meaning that servicers can refuel as many times as they want. Servicers refuelled on average 1.93 times. Figure 2.6 shows where servicers started versus where they refuelled and Table 2.9 shows a tabular representation of the same data. The distribution of the number of refuelling events by the servicer fuel capacity is shown in Table 2.10. As would be expected, servicers with a smaller fuel capacity refuelled more often. These results motivated the second case study which focused on refuelling. In this section, we have shown a few examples of insights to be gained when using this model to solve a robot servicer scheduling and routing problem. Next, we present the details of the Refuelling Case Study.

Table 2.10: Distribution of refuelling events by servicer starting fuel.

Servicer Fuel Capacity (ΔV)	Number of Refuelling Events
25	498
15	883
10	1292
Total	2673

2.4.3 Refuelling Case Study

Following the Core Case Study, we completed a smaller, more focused test design to determine how refuelling affects the number of weighted tasks completed. We proceed in the section with the test design and then present the results.

The test design for the Refuelling Case Study was a full factorial design of 30 runs based on the factors and levels in Table 2.11. Half of the runs had 5 refuelling depots and the other half had none. We also examined a longer time horizon than the main test design, using 48 hours, with the hypothesis that refuelling becomes more influential in a longer time horizon. Finally, because a longer time horizon makes the problem more difficult to solve, we extended the solving time limit to 72 hours.

The results of the Refuelling Case study indicated that the presence of refuelling depots enabled the completion of more tasks, however, not as significantly as expected. Servicers, on average, completed on average $1-2$ more tasks when refuelling depots were available. It is likely these results would have been more significant if solving difficulties were not present. Even with the longer solving time, only 19 of the 30 runs solved to integer optimality. None of the runs with

Table 2.11: Factors and levels for the focused Refueling Case Study.

Factor	**Levels**
Number of Fuel Depots	0, 5[†]
Number of Robot Servicers	1
Robot Servicer Fuel Capacity	10, 15, 25
Servicer Starting Node	12-LEO, 32-MEO, 80-MEO, 125-GEO, 150-GEO
Number of Tasks	8
Time Horizon (*hours*)	48
Solving Time Limit (*hours*)	72

[†] No runs without any fuel depots were accomplished in the original data set.

Table 2.12: How refuelling depots affected task completion.

Servicer Starting Orbit	Servicer Fuel Capacity	Without Refuelling		With Refuelling	
		Complete	Incomplete	Complete	Incomplete
LEO	10	1	7	3	5
	15	3	5	5	3
	25	5	3	6	2
MEO	10	5	11	7	9
	15	6	10	9	7
	25	10	6	10	6
GEO	10	6	10	7	9
	15	6	10	10	6
	25	10	6	11	5

refuelling depots and the smallest fuel capacity of $10\Delta V$ solved to optimality within the 72 hour time limit. All of the runs without refuelling depots reached integer optimality within the 72 hour time period. Thus, the $1-2$ task improvement with refuelling depots is a lower bound because the true optimal solutions could be up to 39% improved in the runs which did not reach integer optimality. In Table 2.12, we show the results for all 30 grouped by the servicer fuel capacity and servicer starting orbit. As would be expected, the impact of refuelling depots on task completion is most evident when the servicer fuel capacity is lower. Although on orbit refuelling depots do not yet exist, these results show that refuelling is necessary to fully enable the long term operation of robot servicers. In the next case study, we examine the effect that the inter-orbit transfers have on the weighted number of tasks completed.

2.4.4 Single Versus Multi-Orbit Case Study

During the Core Case Study, we observed that when tasks were completed, they were most often completed by servicers which started in a different orbit (see Figure 2.5). This observation combined with the multi-orbit nature of our problem motivated the creation and execution of a smaller case study focused on determining the difference in the weighted number of completed tasks when inter-orbit maneuvers are and are not allowed.

This section summarizes the details and results of the Single Versus Multi-Orbit (SVM) Case Study. We begin with the test design and then present the results and insights.

We call the network which allows inter orbit transfers a *multi-orbit network* and the network without inter orbit transfers a *single orbit network*. We show the SVM Case Study factors and levels in Table 2.13. We completed 36 runs using these factors and levels. We did not include time horizons greater than 36 hours for the multi-orbit network configuration due to solving issues observed in the Core Case Study. In contrast to the Core Case Study, we examined multiple different time horizons and only included runs with 5 servicers and 5 refuelling depots. In the comparisons, we included tasks on orbits which had a servicer and excluded all tasks which would be unreachable without inter orbit transfers reducing the number of tasks to 4 or 8. To create the single orbit networks, we followed the arc creation algorithm as shown in Algorithm 1 for each time horizon, and then removed all arcs which allowed servicers to move between orbits or exceeded the time horizon.

Table 2.13: Factors and levels for the focused Single vs. Multi-Orbit Case Study.

Factor	**Levels**
Number of Robot Servicers	5
Robot Servicer Fuel Capacity	10, 15, 25
Number of Tasks	4,8
Network Configuration	Multi-Orbit, Single Orbit
Time Horizon (*hours*)	24, 36, 48, 72

The mean solving time for the runs with a single orbit network configuration was less than 13 minutes, while the mean solving time for runs with the multi-orbit network configuration was

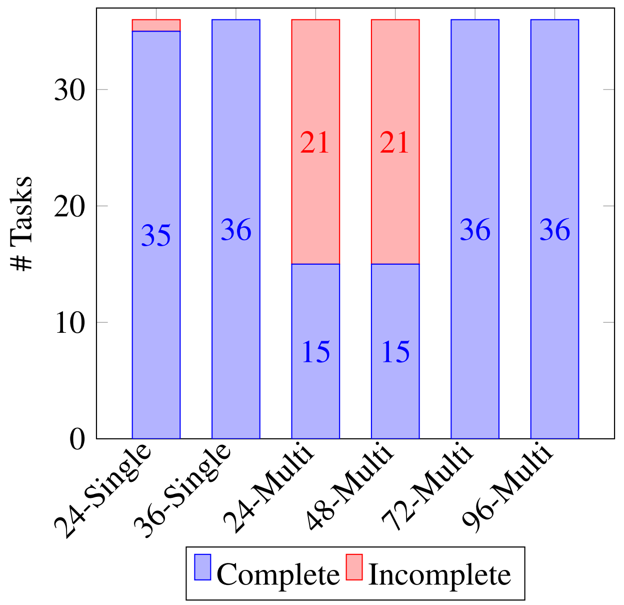

Figure 2.7: Task completion by network configuration and time horizon.

nearly 11 hours. The results of the runs are shown in Figure 2.7. Each bar shows the results from 36 possible tasks for that network configuration/time horizon combination.

For our network, the effect of the multi-orbit network is significant. Servicers operating on the multi-orbit network completed nearly as many tasks in 24 hours as those operating on the single orbit network in 72 hours. These results would likely translate to similar results with larger networks and longer time horizons which could justify using multi-orbit servicers at some point in the future as space technologies advance.

2.5 Conclusions

We are the first to consider the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots (MORSO) problem. We present a novel Mixed Integer Linear Program formulation seeking to maximize the weighted number of tasks completed within a given time horizon. We created new algorithms to model the movement of tasks and refuelling depots over time and to generate the network data based on time and ΔV costs which come from orbital mechanics calculations. We performed computational tests using data created based on satellites which are currently on-orbit in LEO, MEO, and GEO and provide managerial insights which inform decision makers on the number and configuration of robot servicers needed for a set of tasks. We also demonstrated the benefit of on orbit refuelling depots and the importance of inter-orbit transfers for task completion.

To validate MORSO, we completed three case studies using our network and were able to provide results concerning where robot servicers were completing tasks and how refuelling depots and servicer starting locations influence tasks completions. These results translate into policy insights for the numbers and locations of both robot servicers and refuelling depots. We also present analysis showing how network configuration impacts the types and numbers of movements that robot servicers make. Finally, we presented the significant impact of a multi versus single orbit network on task completions. These results are based on our network, however similar types of insights would be possible from larger network configurations, with more detailed maneuver costs. Incorporating additional maneuver types, precise rendezvous time and ΔV costs, and incorporating task times are definite possibilities for future research.

Another facet of this work is the initial evidence surrounding the solving difficulty of MORSO when constructed with larger networks, more servicers, more refuelling depots, and longer time horizons. The solving difficulties encountered with even 24 hour time horizon empirically demonstrate that the MORSO may be hard to solve. To prove the MORSO is hard to solve is the first direction for follow-on research related to this work.

Although the technology for the MORSO does not yet exist, there are many additional areas of research such as the development of a heuristic to obtain optimal or near optimal solutions to the MORSO more quickly. With a heuristic, much longer time horizons and much larger networks could be investigated. Further research could also include examining the MORSO as a multi-objective optimization maximizing the number of tasks completed while minimizing time and ΔV expenditures. This work has demonstrated that this model can be used to effectively optimize the scheduling and routing of robot servicers to accomplish on-orbit servicing tasks across many orbits and presents algorithms which make it robust to larger and more complex network considerations.

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Bibliography

[1] Alfriend, K. T., Lee, D. J., and Creamer, N. G. (2006). Optimal Servicing of Geosynchronous Satellites. *Journal of Guidance, Control, and Dynamics*, 29(1):203–206.

[2] Bourjolly, J., Gürtuna, Ö., and Lyngvi, A. (2006). On Orbit Servicing: A Time Dependent, Moving Target Traveling Salesman Problem. *International Transactions in Operational Research*, 13(5):461–481.

[3] British Broadcasting Corporation (2020). Space news: First Ever Space 'Petrol Station' to Be Built in UK. `https://www.bbc.co.uk/newsround/54551557`. Last Accessed: July 4, 2022.

[4] Corbin, B. A., Abdurezzak, A., Newell, L. P., Roesler, G. M., and Lal, B. (2020). Global Trends in On-Orbit Servicing, Assembly and Manufacturing (OSAM). `https://www.ida.org/research-and-publications/publications/all/g/gl/global-trends-in-on-orbit-servicing-assembly-and-manufacturing-osam`. IDA Document D-13161.

[5] Curtis, H. (2019). *Orbital Mechanics for Engineering Students*. Elsevier Science & Technology, San Diego.

[6] Daneshjou, K., Mohammadi-Dehabadi, A. A., and Bakhtiari, M. (2017). Mission Planning For On-Orbit Servicing Through Multiple Servicing Satellites: A New Approach. *Advances in Space Research*, 60(6):1148–1162.

[7] Du, B., Zhao, Y., Dutta, A., Yu, J., and Chen, X. (2015). Optimal Scheduling of Multispacecraft Refueling Based on Cooperative Maneuver. *Advances in Space Research*, 55(12):2808–2819.

[8] Dutta, A. and Tsiotras, P. (2007). A Greedy Random Adaptive Search Procedure for Optimal Scheduling of P2P Satellite Refueling. In *AAS/AIAA Space Flight Mechanics Meeting*, pages 07–150.

[9] Dutta, A. and Tsiotras, P. (2008). A Cooperative P2P Refueling Strategy for Circular Satellite Constellations. In *AIAA SPACE 2008 Conference & Exposition*, page 7643.

[10] Dutta, A. and Tsiotras, P. (2010). Network Flow Formulation for Cooperative Peer-to-Peer Refueling Strategies. *Journal of Guidance, Control, and Dynamics*, 33(5):1539–1549.

[11] ESA (2018). Removing Debris to Demonstrate Commercial In-Orbit Servicing. `https://blogs.esa.int/cleanspace/2018/09/06/removing-a-debris-to-demonstrate-commercial-in-orbit-servicing/`. Last Accessed: July 4, 2022.

[12] Gürtuna, Ö. and Trépanier, J. (2003). On-Orbit Satellite Servicing: A Space-Based Vehicle On-Orbit Servicing Routing Problem. In *Operations Research in Space and Air*, pages 123–141. Springer.

[13] Hudson, J. S. and Kolosa, D. (2020). Versatile On-Orbit Servicing Mission Design in Geosynchronous Earth Orbit. *Journal of Spacecraft and Rockets*, 57(4):844–850.

[14] Li, W. J., Cheng, D. Y., Liu, X. G., Wang, Y. B., Shi, W. H., Tang, Z. X., Gao, F., Zeng, F. M., Chai, H. Y., Luo, W.-B., Cong, Q., and Gao, Z. L. (2019). On-orbit Service (OOS) of Spacecraft: A Review of Engineering Developments. *Progress in Aerospace Sciences*, 108:32–120.

[15] NASA (2020). NASA Selects Proposals to Demonstrate In-Space Refueling and Propellant Depot Tech. https://www.nasa.gov/directorates/spacetech/solicitations/tipping_points/2020_selections. Last Accessed: July 4, 2022.

[16] NASA (2021a). About - Hubble Servicing Missions. https://www.nasa.gov/mission_pages/hubble/servicing/index.html. Last Accessed: July 4, 2022.

[17] NASA (2021b). On-orbit Servicing, Assembly, and Manufacturing 1 (OSAM-1). https://nexis.gsfc.nasa.gov/osam-1.html. Last Accessed: July 4, 2022.

[18] Northrup Grumman (2020). Mission Extension Vehicle. https://news.northropgrumman.com/news/releases/northrop-grumman-and-intelsat-make-history-with-docking-of-second-mission-extension-vehicle-to-extend-life-of-satellite. Last Accessed: July 4, 2022.

[19] Sarton du Jonchay, T., Chen, H., Gunasekara, O., and Ho, K. (2020). Rolling Horizon Optimization Framework For the Scheduling of On-Orbit Servicing Operations Under Servicing Demand Uncertainties. In *ASCEND 2020*, page 4131. American Institute of Aeronautics and Astronautics, Inc.

[20] Shen, H. and Tsiotras, P. (2005). Peer-to-Peer Refueling for Circular Satellite Constellations. *Journal of Guidance, Control, and Dynamics*, 28(6):1220–1230.

[21] Space-Track.org (2021). Space-Track.org. https://www.space-track.org. Last Accessed: July 4, 2022.

[22] Yu, J., Ouyang, Q., Chen, X. Q., and Chen, L. H. (2013). Optimal Peer-to-Peer Maneuvers for Refueling Satellites in Circular Constellations. *Applied Mechanics and Materials*, 290:41.

[23] Yu, J., Yu, Y. G., Huang, J. T., Chen, X. Q., and Liu, H. Y. (2017). Optimal Scheduling of GEO On-Orbit Refuelling with Uncertain Object Satellites. *MATEC Web of Conferences*, 114:3001.

[24] Zhang, J., Parks, G. T., Luo, Y. Z., and Tang, G. J. (2014). Multispacecraft Refueling Optimization Considering the J2 Perturbation and Window Constraints. *Journal of Guidance, Control, and Dynamics*, 37(1):111–122.

[25] Zhang, T. J., Yang, Y. K., Wang, B. H., Li, H. N., Li, Z., and Shen, H. X. (Oct 2019). Optimal Scheduling for Location Geosynchronous Satellites Refueling Problem. *Acta Astronautica*, 163:264–271.

[26] Zhao, Z., Zhang, J., Li, H. Y., and Zhou, J. Y. (2017). LEO Cooperative Multi-Spacecraft Refueling Mission Optimization Considering J2 Perturbation and Target's Surplus Propellant Constraint. *Advances in Space Research*, 59(1):252–262.

[27] Zhou, Y., Yan, Y., Huang, X., and Kong, L. (2015). Optimal Scheduling of Multiple Geosynchronous Satellites Refueling Based on a Hybrid Particle Swarm Optimizer. *Aerospace Science and Technology*, 47:125–134.

[28] Zhou, Y., Yan, Y., Huang, X., and Yang, Y. (2017). Multi-Objective Planning of a Multiple Geostationary Spacecraft Refuelling Mission. *Engineering Optimization*, 49(3):531–548.

Appendix

Appendix 2.A Appendix: Orbital Mechanics

2.A.1 Orbital Parameters and Constants

These orbital parameters and constants were used in the creation of the data sets in this work.

a or r: semi-major axis, distance from the center the Earth in km; orbital altitude

ν or θ: true anomaly, degrees, where on the "orbit" the node is located

i: inclination, degrees, the vertical "tilt" of the orbit

e: eccentricity, the "roundness" of the orbit

Ω: right ascension of ascending node (RAAN), degrees, the horizontal "tilt" of the orbit

ω: perigee (angle to periapsis, the location on the orbit that is closest to Earth)

μ: Earth's gravitational parameter in km^3/s^2 $398,600$

NOTE: Circular orbits have $e = 0$ and do not have ω, because all points on the orbit have the same distance to Earth.

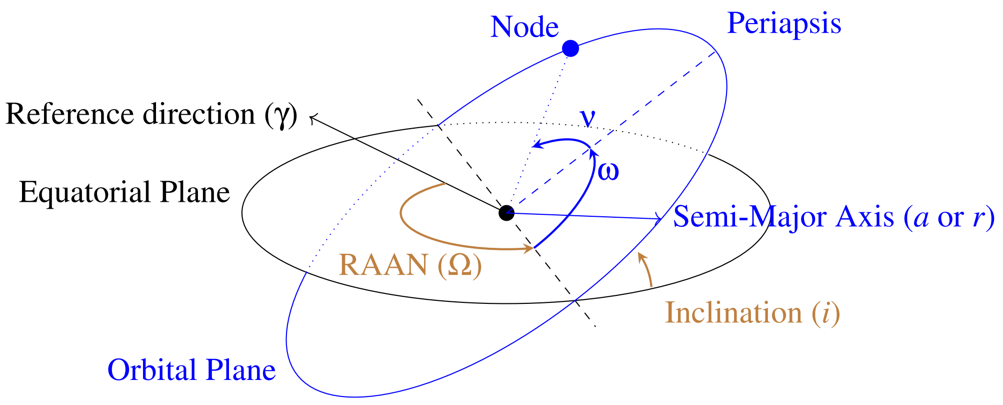

Figure 2.A.1: Orbital Parameters

The period (the time) of a circular orbit is calculated as follows: $T = \frac{2\pi r}{\sqrt{\frac{\mu}{r}}}$, in seconds, where r is the semi-major axis of the orbit.

2.A.2 Hohmann Transfer

A Hohmann transfer is used to make a change in orbital altitude, r on the same orbital plane. The arrival location on the destination orbit is at a position of 180 degrees forward or backward (ν), depending on where maneuver was initiated. The calculations for a change in altitude from circular orbit r_1 to the co-planar orbit r_2 are as follows [5]:
The angular momentum of the starting orbit is:

$$h_1 = \sqrt{2 * \mu}\sqrt{\frac{r_1 * r_1}{r_1 + r_1}}\ km^2/s \tag{2.9}$$

The angular momentum of the transfer orbit is:

$$h_2 = \sqrt{2 * \mu}\sqrt{\frac{r_1 * r_2}{r_1 + r_2}}\ km^2/s \tag{2.10}$$

The angular momentum of the destination orbit is:

$$h_3 = \sqrt{\mu r_2}\ km^2/s \tag{2.11}$$

The velocity at the starting orbit is:

$$V_1 = \frac{\sqrt{2 * \mu}\sqrt{\frac{r_1 * r_1}{r_1 + r_1}}}{r_1}\ km/s \tag{2.12}$$

The velocity at the transfer orbit is:

$$V_2 = \frac{\sqrt{2 * \mu}\sqrt{\frac{r_1 * r_2}{r_1 + r_2}}}{r_1}\ km/s \tag{2.13}$$

The velocity at the destination orbit is:

$$V_3 = \frac{\sqrt{\mu r_2}}{r_2}\ km/s \tag{2.14}$$

The ΔV for the maneuver is:

$$\begin{aligned}\Delta v_1 &= V_2 - V_1 \\ \Delta v_2 &= V_3 - V_2 \\ \Delta V &= |v_1| + |v_2|\end{aligned} \tag{2.15}$$

The time for the maneuver is:

$$\frac{T}{2} = \frac{2\pi r_2^{\frac{3}{2}}}{2\sqrt{\mu}} \tag{2.16}$$

2.A.3 Combined Hohmann Transfer with Inclination Change

This maneuver combines a change from r_1 to r_2 (altitude) and a change in i (inclination). Let the starting orbit (i) have parameters r_1 and i_1 and the destination orbit (j) have parameters r_2 and i_2. The calculations are as follows [5]:

The velocity at the starting orbit, i:

$$v_{i_1} = \sqrt{\frac{\mu}{r_1}}\ km/s \tag{2.17}$$

The angular momentum at the destination orbit, j:

$$h_j = \sqrt{2\mu}\sqrt{\frac{r_1 * r_2}{r_1 + r_2}}\ km/s \tag{2.18}$$

Intermediate calculations:

$$v_{i_2} = \frac{h_j}{r_1} \tag{2.19}$$

$$v_{j_2} = \frac{h_j}{r_2} \tag{2.20}$$

$$v_{j_3} = \sqrt{\frac{\mu}{r_2}} \tag{2.21}$$

The change in inclination can be done either at the starting orbit, the destination orbit or half and half. The lower the altitude, the more expensive the ΔV for the inclination change [5]. At the destination:

$$\begin{aligned}
\Delta v_j &= \sqrt{v_{j_2}^2 + v_{j_3}^2 - 2 * v_{j_2} * v_{j_3} * \cos\Delta i} && (2.22)\\
&= \sqrt{\left(\frac{h_j}{r_2}\right)^2 + \left(\frac{\mu}{r_2}\right) - 2 * \frac{h_j}{r_2} * \sqrt{\frac{\mu}{r_2}} * \cos\Delta i}\\
\Delta v_i &= v_{i_2} - v_{i_1} = \frac{h_j}{r_1} - \sqrt{\frac{\mu}{r_1}} && (2.23)\\
\Delta V &= \Delta v_i + \Delta v_j && (2.24)
\end{aligned}$$

At the origin:

$$\begin{aligned}
\Delta v_i &= \sqrt{v_{i_1}^2 + v_{i_2}^2 - 2 * v_{i_2} * v_{i_1} * \cos\Delta i} && (2.25)\\
&= \sqrt{\frac{\mu}{r_1} + \left(\frac{h_j}{r_1}\right)^2 - 2 * \sqrt{\frac{\mu}{r_1}} * \frac{h_j}{r_1} * \cos\Delta i}\\
\Delta v_j &= v_{j_3} - v_{j_2} = \sqrt{\frac{\mu}{r_2}} - \frac{h_j}{r_2} && (2.26)\\
\Delta V &= \Delta v_i + \Delta v_j && (2.27)
\end{aligned}$$

We calculated the costs for accomplishing the inclination change at both the starting orbit and the destination orbit and the result with the smaller ΔV.

2.A.4 Phasing Maneuver

A phasing maneuver is used for a change in the *true anomaly*. Let $\Delta\phi$ = the change in *true anomaly* angle between nodes i and j in radians. We can calculate the costs for a phasing maneuver as follows [5]:

$$\begin{cases} \theta_j - \theta_i < 0 & \Delta\phi = 360 + \theta_j - \theta_i \frac{\pi}{180} \\ \theta_j - \theta_i \geq 0 & \Delta\phi = \theta_j - \theta_i \frac{\pi}{180} \end{cases} \qquad (2.28)$$

The period of the starting orbit is:

$$T_s = \frac{2\pi r}{\sqrt{\frac{\mu}{r}}} \text{ seconds} \tag{2.29}$$

The mean motion of the satellite in its circular orbit is:

$$n = \sqrt{\frac{\mu}{r^3}} \tag{2.30}$$

k is the number of revolutions which the satellite performs in the phasing ellipse, here in this work we assume $k = 1$. The minimum k is 1. As k increases, the time increases, and the ΔV for the maneuver decreases. So then the phasing ellipse orbital period in seconds is [5]:

$$T_{\text{phasing ellipse}} = \frac{2k\pi + \Delta\phi}{kn} \tag{2.31}$$

And the semi major axis of the phasing ellipse is:

$$a(k,n,r)_{\text{phasing ellipse}} = \left(\frac{\mu \left(\frac{2k\pi+\Delta\phi}{kn} \right)^2}{4\pi^2} \right)^{\frac{1}{3}} \tag{2.32}$$

The ΔV to start and stop the maneuver is equal so the total ΔV is:

$$\Delta V = 2 * \left| \sqrt{\frac{2\mu}{r} - \frac{\mu}{a(k)_{\text{phasing ellipse}}}} - \sqrt{\frac{\mu}{r}} \right| \tag{2.33}$$

The time to complete the phasing maneuver is:

$$t_{\Delta\phi} = T_{\text{phasing ellipse}} * \frac{\Delta\phi}{2\pi} + k * T_{\text{phasing ellipse}} \tag{2.34}$$

Appendix 2.B Demonstration of Algorithms Presented in this Work

2.B.1 Arc Creation Algorithm Example

To aid the in the explanation of arc creation we provide a specific example as follows. Consider node $i = 3$ in orbit 1. Node 3 has a stationary true anomaly of 120°, a semi-major axis of $6652.55(km)$, and an inclination of 52.986. For node $i = 3$ there are 10 arcs possible. Table 2.B.1 shows the nodes that node $i = 3$ is connected to in column j. The third column indicates the type of maneuver. Node $i = 3$ is adjacent in the rotation direction of the orbit to node $j = 4$ on the same orbit therefore the arc $(3,4)$ between these nodes is an orbiting maneuver arc. Node $i = 3$ can make a phasing maneuver to any other node on the same orbit which is not adjacent in the rotation direction of the orbit, therefore node $i = 3$ is connected to nodes $j = 1,2,5,6$ by *phasing maneuver* arcs. Node $i = 3$ is connected to orbit 4 on the arc $(3,39)$, orbit 5 on the arc $(3,63)$, orbit 6 on the arc $(3,87)$, orbit 7 on arc $(3,111)$ and to orbit 8 on arc $(3,155)$ using *Hohmann transfer with inclination change* arcs.

Table 2.B.1: All Possible Arcs from Node $i = 3$ with Maneuver Types, Time and Fuel Costs

i	j	Maneuver Type	Time Cost τ_{ij}	ΔV (Fuel Cost) ϕ_{ij}	Fuel per Time Period Ψ_{ij}
3	1	Phasing	17	2.09	0.12
3	2	Phasing	20	2.39	0.12
3	4	Orbit	1	0.00	0.00
3	5	Phasing	11	1.30	0.12
3	6	Phasing	13	1.74	0.13
3	39	Hohmann+incline	24	8.24	0.34
3	63	Hohmann+incline	24	7.44	0.31
3	87	Hohmann+incline	24	7.44	0.31
3	111	Hohmann+incline	24	7.44	0.31
3	155	Hohmann+incline	48	7.02	0.15

2.B.2 Sub Task Creation Algorithm Example

An example of the subtasks for Task 4 is shown in Table 2.B.2 below.

Table 2.B.2: MORSO Core Case Study: Sub Tasks for Task 4

Task	Sub Task	Start Time	End Time	Weight
4	19	1	1	1
4	20	3	3	1
4	21	5	5	1
4	22	7	7	1
4	23	9	9	1
4	24	11	11	1
4	25	13	13	1
4	⋮	⋮	⋮	⋮
4	40	91	91	1
4	41	93	93	1
4	42	95	95	1

2.B.3 Refueling Arc Designation Algorithm Example

An example of the refuelling arcs for a depot starting at node 12 is shown in Table 2.B.3. The scenarios in this case study have 1 to 5 robot servicers available to accomplish tasks. The robots start the overall time period at one of the refuelling depots. The movements of the robot servicers are controlled with the decision variable y_{dijt} as and their fuel level is controlled with the decision variable f_{dt}. Like the tasks, the fuel depots are also in orbit and so for a particular arc in a refueling orbit, we have times when that depot is available for refueling. For all case studies, we have 162 orbiting maneuver arcs. Thus, for a given depot, we can determine if and when each of these arcs is available to refuel robot servicers. Table 2.B.3 shows on which arcs at which times that fuel depot 12 is available for refueling. Thus if a robot servicer needs to refuel, it would need to move on one of these arcs at one of the times listed as these are the $(i,j,t) \in A_t^R$ for depot 12.

Table 2.B.3: Example: Refuelling Arcs for Depot Starting at Node 12

From	To	
i	*j*	Refuelling Time Periods: *t*
12	7	0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90
7	8	1, 7, 13, 19, 25, 31, 37, 43, 49, 55, 61, 67, 73, 79, 85, 91
8	9	2, 8, 14, 20, 26, 32, 38, 44, 50, 56, 62, 68, 74, 80, 86, 92
9	10	3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63, 69, 75, 81, 87, 93
10	11	4, 10, 16, 22, 28, 34, 40, 46, 52, 58, 64, 70, 76, 82, 88,94
11	12	5, 11, 17, 23, 29, 35, 41, 47, 53, 59, 65, 71, 77, 83, 89,95

Bibliography

[1] Alfriend, K. T., Lee, D. J., and Creamer, N. G. (2006). Optimal Servicing of Geosynchronous Satellites. *Journal of Guidance, Control, and Dynamics*, 29(1):203–206.

[2] Bourjolly, J., Gürtuna, Ö., and Lyngvi, A. (2006). On Orbit Servicing: A Time Dependent, Moving Target Traveling Salesman Problem. *International Transactions in Operational Research*, 13(5):461–481.

[3] British Broadcasting Corporation (2020). Space news: First Ever Space 'Petrol Station' to Be Built in UK. `https://www.bbc.co.uk/newsround/54551557`. Last Accessed: July 4, 2022.

[4] Corbin, B. A., Abdurezzak, A., Newell, L. P., Roesler, G. M., and Lal, B. (2020). Global Trends in On-Orbit Servicing, Assembly and Manufacturing (OSAM). `https://www.ida.org/research-and-publications/publications/all/g/gl/global-trends-in-on-orbit-servicing-assembly-and-manufacturing-osam`. IDA Document D-13161.

[5] Curtis, H. (2019). *Orbital Mechanics for Engineering Students*. Elsevier Science & Technology, San Diego.

[6] Daneshjou, K., Mohammadi-Dehabadi, A. A., and Bakhtiari, M. (2017). Mission Planning For On-Orbit Servicing Through Multiple Servicing Satellites: A New Approach. *Advances in Space Research*, 60(6):1148–1162.

[7] Du, B., Zhao, Y., Dutta, A., Yu, J., and Chen, X. (2015). Optimal Scheduling of Multispacecraft Refueling Based on Cooperative Maneuver. *Advances in Space Research*, 55(12):2808–2819.

[8] Dutta, A. and Tsiotras, P. (2007). A Greedy Random Adaptive Search Procedure for Optimal Scheduling of P2P Satellite Refueling. In *AAS/AIAA Space Flight Mechanics Meeting*, pages 07–150.

[9] Dutta, A. and Tsiotras, P. (2008). A Cooperative P2P Refueling Strategy for Circular Satellite Constellations. In *AIAA SPACE 2008 Conference & Exposition*, page 7643.

[10] Dutta, A. and Tsiotras, P. (2010). Network Flow Formulation for Cooperative Peer-to-Peer Refueling Strategies. *Journal of Guidance, Control, and Dynamics*, 33(5):1539–1549.

[11] ESA (2018). Removing Debris to Demonstrate Commercial In-Orbit Servicing. `https://blogs.esa.int/cleanspace/2018/09/06/removing-a-debris-to-demonstrate-commercial-in-orbit-servicing/`. Last Accessed: July 4, 2022.

[12] Gürtuna, Ö. and Trépanier, J. (2003). On-Orbit Satellite Servicing: A Space-Based Vehicle On-Orbit Servicing Routing Problem. In *Operations Research in Space and Air*, pages 123–141. Springer.

[13] Hudson, J. S. and Kolosa, D. (2020). Versatile On-Orbit Servicing Mission Design in Geosynchronous Earth Orbit. *Journal of Spacecraft and Rockets*, 57(4):844–850.

[14] Li, W. J., Cheng, D. Y., Liu, X. G., Wang, Y. B., Shi, W. H., Tang, Z. X., Gao, F., Zeng, F. M., Chai, H. Y., Luo, W.-B., Cong, Q., and Gao, Z. L. (2019). On-orbit Service (OOS) of Spacecraft: A Review of Engineering Developments. *Progress in Aerospace Sciences*, 108:32–120.

[15] NASA (2020). NASA Selects Proposals to Demonstrate In-Space Refueling and Propellant Depot Tech. `https://www.nasa.gov/directorates/spacetech/solicitations/tipping_points/2020_selections`. Last Accessed: July 4, 2022.

[16] NASA (2021a). About - Hubble Servicing Missions. `https://www.nasa.gov/mission_pages/hubble/servicing/index.html`. Last Accessed: July 4, 2022.

[17] NASA (2021b). On-orbit Servicing, Assembly, and Manufacturing 1 (OSAM-1). `https://nexis.gsfc.nasa.gov/osam-1.html`. Last Accessed: July 4, 2022.

[18] Northrup Grumman (2020). Mission Extension Vehicle. `https://news.northropgrumman.com/news/releases/northrop-grumman-and-intelsat-make-history-with-docking-of-second-mission-extension-vehicle-to-extend-life-of-satellite`. Last Accessed: July 4, 2022.

[19] Sarton du Jonchay, T., Chen, H., Gunasekara, O., and Ho, K. (2020). Rolling Horizon Optimization Framework For the Scheduling of On-Orbit Servicing Operations Under Servicing Demand Uncertainties. In *ASCEND 2020*, page 4131. American Institute of Aeronautics and Astronautics, Inc.

[20] Shen, H. and Tsiotras, P. (2005). Peer-to-Peer Refueling for Circular Satellite Constellations. *Journal of Guidance, Control, and Dynamics*, 28(6):1220–1230.

[21] Space-Track.org (2021). Space-Track.org. `https://www.space-track.org`. Last Accessed: July 4, 2022.

[22] Yu, J., Ouyang, Q., Chen, X. Q., and Chen, L. H. (2013). Optimal Peer-to-Peer Maneuvers for Refueling Satellites in Circular Constellations. *Applied Mechanics and Materials*, 290:41.

[23] Yu, J., Yu, Y. G., Huang, J. T., Chen, X. Q., and Liu, H. Y. (2017). Optimal Scheduling of GEO On-Orbit Refuelling with Uncertain Object Satellites. *MATEC Web of Conferences*, 114:3001.

[24] Zhang, J., Parks, G. T., Luo, Y. Z., and Tang, G. J. (2014). Multispacecraft Refueling Optimization Considering the J2 Perturbation and Window Constraints. *Journal of Guidance, Control, and Dynamics*, 37(1):111–122.

[25] Zhang, T. J., Yang, Y. K., Wang, B. H., Li, H. N., Li, Z., and Shen, H. X. (Oct 2019). Optimal Scheduling for Location Geosynchronous Satellites Refueling Problem. *Acta Astronautica*, 163:264–271.

[26] Zhao, Z., Zhang, J., Li, H. Y., and Zhou, J. Y. (2017). LEO Cooperative Multi-Spacecraft Refueling Mission Optimization Considering J2 Perturbation and Target's Surplus Propellant Constraint. *Advances in Space Research*, 59(1):252–262.

[27] Zhou, Y., Yan, Y., Huang, X., and Kong, L. (2015). Optimal Scheduling of Multiple Geosynchronous Satellites Refueling Based on a Hybrid Particle Swarm Optimizer. *Aerospace Science and Technology*, 47:125–134.

[28] Zhou, Y., Yan, Y., Huang, X., and Yang, Y. (2017). Multi-Objective Planning of a Multiple Geostationary Spacecraft Refuelling Mission. *Engineering Optimization*, 49(3):531–548.

3. Complexity and Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots

Susan E. Sorenson Sarah G. Nurre Pinkley

Abstract: The Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing optimization problem determines how to best route and schedule a fleet of highly maneuverable and refuellable space robot servicers to complete a set of tasks orbiting in space. We prove that this problem with objective seeking to maximize the weighted number of completed tasks within a set time horizon is $NP-$Hard. In spite of these results, we present and demonstrate two heuristics which utilize a node labeling algorithm. The first heuristic assigns servicers to tasks greedily and the second heuristic assigns tasks using a clustering algorithm. Using a realistic case study, with a network spanning the Low, Mid, and Geosynchronous Earth Orbits, and tasks based on satellites that are currently on orbit, we compare our heuristics with CPLEX and present encouraging results.

3.1 Introduction

On-orbit servicing (OOS) is a field that has experienced tremendous growth over the past decade and covers a wide range of activities spanning fixing, improving, removing, or reviving satellites and refers to any work to refuel, repair, replace, or augment an existing asset in space [16]. In this work, we examine the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem which determines how to schedule and route a fleet of highly maneuverable refuellable space robots to complete a set of OOS tasks spanning multiple orbits in space. Although we use the term OOS, we are referring to any space-based activity that a servicer might do which could be inspection, refuelling, repair, replace, augment, or upgrade.

Although the technology is not yet widely available, OOS with refuellable servicers will soon be the norm rather than a novel occurrence. Recent successes in OOS have occurred by

Northrup Grumman [17] and in 2020, NASA granted contracts to several companies to develop and demonstrate space and lunar based based fuel depots [15]. One company, Thales Alenia Space is building a chemical refuelling depot with an expected launch in 2027 [3]. These projects are only a few examples of the many proposed or already launched systems. The reader should investigate [14, 5] for comprehensive reviews of engineering developments and planned OOS missions or projects.

The purpose of this work is to establish the computational complexity of the MORSO problem and to present two fast accurate constructive heuristics for solving this problem. We focus on the optimization goal of maximizing the weighted number of tasks completed within a set time horizon. With this focus, we present the complexity results for MORSO with a reduction from the known $NP-$Hard non-preemptive single machine scheduling problem seeking to minimize the weighted number of late tasks when all tasks have the same due date [13]. We follow this with two constructive heuristics for solving the MORSO problem. We empirically show that the heuristics can find near optimal solutions in significantly less time than CPLEX.

Main Contributions. The main contributions of this work are as follows: (i) We prove the MORSO problem with objective seeking to maximize the weighted number of completed tasks is $NP-$Hard; (ii) We develop and demonstrate two constructive heuristics for the solving the MORSO problem; (iii) We perform extensive computational experiments on a realistic data set based on active satellites (iv) We summarize the results of the experiments thereby demonstrating the speed, accuracy, and scalability of our proposed heuristic methods.

The remainder of this work is as follows. In Section 3.2, we summarize literature related to the optimization of different types of OOS missions. In Section 3.3, we provide a detailed overview of the MORSO problem. In Section 3.4, we provide the complexity results. In Section 3.5, we present our two new constructive heuristics and in Section 3.6, we demonstrate the speed, accuracy, and scalability using test cases based on the orbital parameters of current satellites. In Section 3.7, we present conclusions and areas for further study.

3.2 Literature

In this section, we focus our literature review on the relevant works related to the computational complexity and optimization of the routing and scheduling of space robots for on-orbit servicing (OOS). We call the highly maneuverable space robots which do the tasks “servicers” and any work that a space vehicle is to receive a “task.” In our work, there is always a set of designated servicers and a set of tasks to be accomplished on other satellites. The servicers actively move to rendezvous with the satellite where their next task is located. In our work, OOS involves at least two participants, one who does the servicing and another that has the task and receives the service. We use the term OOS to refer to any activity that a servicer might do as a task which could be inspection, refuelling, repair, replace, augment, or upgrade.

The optimization of the routing and scheduling of space robots is relatively new, and thus works which examine the computational complexity of OOS are few. In Gürtuna and Trépanier [10] OOS is modelled as a Vehicle Routing Problem (VRP) and thus, stated to be $NP-$Hard. In Coene et al. [4] the OOS problem is that of refuelling and both the servicer and task move to a different location to share fuel while seeking to minimize the total fuel expended during maneuvers. It is shown to be $NP-$Hard using a reduction from a three dimensional assignment problem with decomposable costs. We will now summarize other works which use heuristic methods for solving problems related to OOS.

The number of works which examine the complexity of OOS are few when compared to the number of works that use constructive and meta heuristics for solving OOS problems. In the next few paragraphs we discuss works which use heuristics for solving different types of OOS problems. For many works which address the scheduling of OOS, the meta heuristic “particle swarm optimization” is used for solving. qian Chen and Yu [18] use a combination of exhaustive search and particle swarm optimization to minimize financial costs while solving a single orbit refuelling problem. A multi-objective particle swarm optimization was used to examine the mission planning for a large scale OOS which sought to simultaneously minimize time and fuel consumption in [6]. Zhou et al. [24] formulated a single orbit time-fixed refuelling problem with multiple refuellable servicers and used a hybrid particle swarm optimizer to minimize the fuel consumed during orbital transfers. Zhang and Zhang [21] used a hybrid discrete particle swarm optimization to minimize lost spacecraft, fuel, and time while simultaneously maximizing the tasks completed.

In addition to particle swarm optimization, other heuristics have also been used for optimizing the routing and scheduling of robot servicers for OOS. Bourjolly et al. [2] consider a single servicer and a set of tasks as a time dependant moving target Travelling Salesman Problem (TSP). For instances with a small number of tasks ($n \leq 10$), they solve the problem with an exhaustive search and use a Tabu search algorithm for instances with a large amount of tasks ($n > 10$). Gürtuna and Trépanier [10] model OOS as a time static Vehicle Routing Problem (VRP) and use the Clarke and Wright Algorithm to determine the best task sequence. Hudson and Kolosa [11] addressed a single orbit multiple servicer OOS problem in GEO as a moving task TSP looking to maximize lifetime profit with simulations run on a well known space simulation software. Zhang et al. [22] examined on-orbit refuelling (OOR) as a location routing problem with multiple refuelling depots and multiple servicers and used the ant colony optimization heuristic for solving. Zhang et al. [20] considered a multi-objective optimization model and used a hybrid encoding genetic algorithm to simultaneously minimize the fuel and time while routing a single servicer to tasks located in closely located but different orbits.

Other OOS problem formulations include a Mixed Integer Non-linear Programming (MINLP) formulation for a refuelling problem, with non-linear fuel cost calculations, seeking to minimize the fuel consumed and solved the problem using a Multi-island Genetic Algorithm Du et al. [7]. Dutta and Tsiotras [8] used a greedy random adaptive search procedure to schedule movements that minimize orbital maneuver fuel expenditure for on orbit refuelling without designated servicers. Called cooperative refuelling, space vehicles with surplus fuel are assigned to transfer fuel to fuel deficient space vehicles. Zhao et al. [23] used a two level approach to solve a MINLP which modelled cooperative refuelling while considering Earth's gravitational effects.

They used a hybrid-encoding genetic algorithm to find near optimal solutions for the up-level optimization and then a linear relative dynamic equation in the low level optimization for the gravitational effects. Zhou et al. [25] also implemented a two-level optimization for a single orbit OOR in which the up-level optimization determined the refuelling order with a non-dominated sorting genetic algorithm and an exact method for the low level optimization to minimize fuel consumption.

With a review of some of the literature related to the complexity and solving heuristics for OOS and OOR problems, we next present the problem statement.

3.3 Problem Statement

In this section, we provide an overview of the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem, presenting our solving goal and nomenclature for the network, tasks, refuelling, robot servicers, and time horizon. In the MORSO problem examined here, we seek to maximize the weighted number of tasks completed within a set time horizon. The discussion that follows in the section is heavily based on Sorenson and Nurre Pinkley [19] in which the MORSO problem and a Mixed Integer Linear Program for solving are initially introduced. This information is included out of necessity to demonstrate the true contributions of this work – problem complexity and heuristic solution methods.

In the MORSO problem, the robot servicers, the satellites with tasks, and the refuelling depots are all constantly moving though a network over time. The network, G, consists of a set of nodes, $N, i \in N$, which are "fixed" locations in space, and a set of directed arcs, $A, (i, j) \in A$, which have associated time: τ_{ij}, and fuel: ϕ_{ij} costs. The nodes and arcs span multiple orbital altitudes and the time and fuel costs are calculated using orbital mechanics.

The problem has a a finite time horizon: T, which is broken down into smaller intervals from $t = 0, \ldots, |T|$. Within the network, the robot servicers, the satellites with tasks, and the refuelling depots all have starting locations which are known and are located at one of the network nodes at $t = 0$. Within the set of tasks, $B, v \in B$, each task is broken down into a set of time discretized subtasks, $k \in B_v$. Each k has an associated time and node and the subtasks are how task movement is expressed over time. Tasks also have a weight, w_v, which corresponds to the importance of task completion. Task completion is tracked with the decision variable:

$$\beta_{vk} = \begin{cases} 1, \text{ if sub task } k \text{ of task } v \text{ is completed, for } v \in B \\ 0, \text{ otherwise} \end{cases} \tag{3.1}$$

Another important aspect of MORSO includes the refuelling of the robot servicers using on-orbit refuelling depots. The set of refuelling depots is expressed as $\Gamma, r \in \Gamma$. Refuelling depot movement is expressed by designating specific arcs as refuelling arcs depending on the time. Arcs which are available for refuelling at time t are contained in the set $A_t^R \subset A$. In this work, when we consider refuelling, we assume that if a servicer moves along an arc denoted a refuelling arc, the servicer completely refuels to the maximum capacity by the end of the refuelling arc.

Finally, the robot servicers, $d \in D$, have a known starting location withing the network and a known maximum fuel capacity, F. The fuel level of a servicer is tracked over time with the decision variable f_{dt}, for $d \in D$ and $t \in T$. We also track the movement of the servicers through the network with a time and a node number for each servicer.

With an understanding of the MORSO problem notation, we next proceed to demonstrating the complexity of MORSO.

3.4 Complexity

We now prove that the MORSO problem with objective seeking to maximize the weighted number of completed tasks is *NP*-Hard. For our proof, we reduce the known $NP-Complete$ problem $1 \mid d_j = d \mid w_jU_j$ to an instance of the MORSO problem. $1 \mid d_j = d \mid w_jU_j$ which is a single machine scheduling problem that seeks to minimize the weighted number of late tasks when all tasks have the same due date [13]. We proceed by presenting the instance of the MORSO prob-

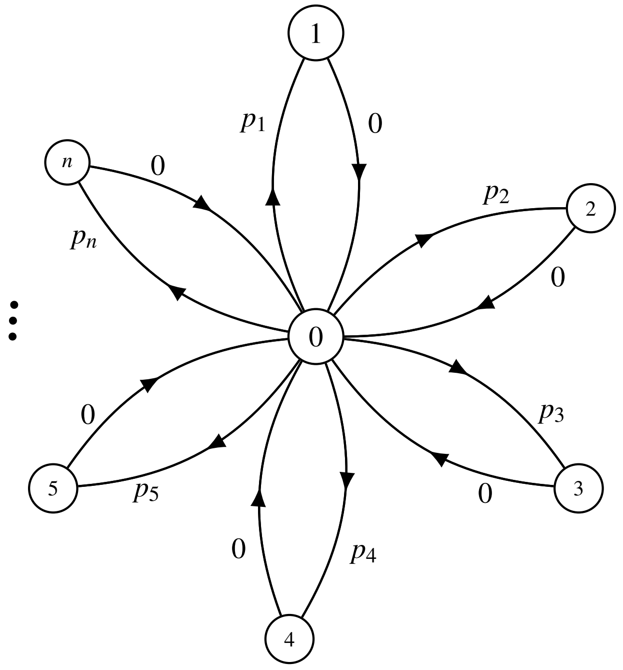

Figure 3.4.1: Graphical representation of a MORSO network with n tasks. The arc cost is equal to the processing time of the task p_j, for $j = 1, \ldots, n = |J|$

lem and the formal proof. We point the reader to Garey and Johnson [9] for an introduction to complexity.

Theorem 3.4.1. *The simplified MORSO problem, excluding multiple orbits, moving tasks, and servicer refuelling, with an objective to maximize the weighted number of completed tasks is $NP-$Hard*

Proof. We reduce the $NP-$Hard problem, $1 \mid d_j = d \mid w_j U_j$, to an instance of our MORSO problem that seeks to maximize the weighted number of completed tasks. We note, in the forthcoming MORSO instance we ignore orbits, refuelling, and moving tasks, thus creating a simplified version of the MORSO problem. As we will show, this simplified version of the MORSO problem is $NP-$Hard, thus, we can then claim the more complex MORSO problem with multiple orbits and moving tasks is also $NP-$Hard.

From the input $1 \mid d_j = d \mid w_j U_j$ problem with 1 machine and n jobs we transform the problem and create an instance of our simplified MORSO problem. We set D = 1 thereby equating one servicer in the MORSO problem with the single machine. Next, we consider a network representation as indicated in Figure 3.4.1. In this network, there is a single central node 0, surrounded by n additional nodes. The n additional nodes correspond to the n jobs in the scheduling problem. Each of these n nodes has a single associated task, where the weight for each task equals w_j from the associated scheduling job. Note, as we are not considering task movement, there are no subtasks needed. Specifically, in our MORSO problem instance, we consider tasks $B = 1,\dots,n$. In the network, the nodes are connected through directed arcs $(0, j)$ and $(j,0)$ for $j = 1,\dots,n$, with traversal times equal to p_j and 0, respectively. In the MORSO instance, we start the single servicer at node 0 and set the time horizon of the problem $T = d$.

With this transformation, we now equate a solution to this MORSO instance to a solution of the input $1 \mid d_j = d \mid w_jU_j$ scheduling problem and vice versa. In the MORSO instance, $\beta_j = 1$ if the task is completed within the time horizon. We seek to equate this to information about whether a schedule job is tardy, $U_j = 1$, or not, $U_j = 0$.

When $\beta_j = 1$, this equates to $U_j = 0$ because task j in the MORSO instance was able to be completed within the time horizon ($T = d$) thereby meaning scheduling job j is not tardy. Likewise, when $\beta_j = 0$, this equates to $U_j = 1$, because task j was not completed within the time horizon of the MORSO instance, thus the associated scheduling job is tardy. Thus, by solving this instance of MORSO, we also get a solution to $1 \mid d_j = d \mid w_jU_j$.

We can translate a solution to $1 \mid d_j = d \mid w_jU_j$ into a solution to MORSO as follows. In a solution to $1 \mid d_j = d \mid w_jU_j$, there are tasks $j = 1,\dots,n$ and each task has a completion time, C_j. If $C_j \leq d$, then $U_j = 0$, otherwise $U_j = 1$. If we let $d = T$, the time horizon in MORSO, then $U_j = 0$ is equivalent to $\beta_j = 1$ and $U_j = 1$ is equivalent to $\beta_j = 0$. Therefore a solution to $1 \mid d_j = d \mid w_jU_j$ provides a solution to MORSO.

□

3.5 Solution Methodology

With the complexity of the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem established, we proceed in this section to present two solving methodologies and a node labelling algorithm that they both use. Because both heuristics use the node labelling algorithm, we begin with node labelling and then proceed with the details of the heuristics. We begin with the assumptions and inputs needed when using the methods in this section.

In this section, we solve MORSO seeking to maximize the weighted number of tasks completed within a given time horizon. The model inputs here are based on the inputs to the mixed integer linear program (MILP) model for MORSO presented in Sorenson and Nurre Pinkley [19]. Our inputs include a network $G = (N, A)$ with nodes $i \in N$ and arcs $(i, j) \in A$. The arcs have associated time and fuel costs: τ_{ij} and ϕ_{ij} respectively. The problem has a a finite time horizon: T, which is broken down into smaller intervals from $t = 0, \dots, |T|$. We have a set of robot servicers D with $d \in D$ that have a known starting location at some $i \in N$ and a known maximum fuel capacity, F. Servicers also have a fuel level which changes as servicers move through the network, f_{dt}, for $d \in D$ and $t \in T$. We have a set of tasks, $B, v \in B$, and each task is broken down into a set of time discretized subtasks, $k \in B_v$. Each subtask k has an associated time and node and task completion occurs instantaneously when a servicer is at the subtask node at the correct time. Finally, the set of refuelling depots is expressed as $\Gamma, r \in \Gamma$, and as the depots move, some of the arcs in A are designated as refuelling arcs, so we have the set $A^R_t \subset A$ which contains the arcs $(i, j) \in A$ which are available for refuelling at time t. The node labelling algorithm centers around the robot servicers which we present next.

3.5.1 Label Making Algorthim

In this section we present the Label Making Algorthim which is used in both of the constructive heuristics that follow. The node labelling algorithm is based on ideas which come from Dijkstra's algorithm and an aircraft shortest path routing with refuelling algorithm [1, 12]. Dijkstra's algorithm makes labels in order to find the shortest path through a network. We extend this idea and use the labels to find efficient paths to tasks while also considering refuelling. The labels provide information about the nodes that the servicer can reach within the time horizon and contain data concerning tasks, fuel levels, and the time to reach the nodes from the servicer starting location. The information in the labels is then used in the heuristics that follow in order to determine how to best assign servicer movements to complete tasks.

Our labels consist of 10 fields and are unique when indexed on the first three fields. The first field is the servicer name, $d \in D$ which pertains to one servicer. The next field is the node number, $i \in N$, to which this label belongs. The label number is the next field, which is a number that begins at 1 for each node, for each servicer, and increments by one, each time a new label is added for this servicer at this node. The next field is the "cost to here" which as labels are made, we add the time cost, τ_{ij} for all of the arcs taken from the servicer starting location to the current node. This value has an upper bound of the time horizon, T. The "fuel remaining" field contains the current fuel level for the servicer after reaching this node which is $f_{dt} - \phi_{ij}$ if the arc ending at the node is not a refuelling arc. This field has a lower bound of 0. If the arc is a refuelling arc, we assume the servicer takes on fuel from the depot on that arc and then the "fuel remaining" field is the maximum fuel capacity of the servicer, F. The sixth field is an indicator which is 0 if there is not a task at this node and the task number, if there is a task at the node. The seventh and eighth fields track the nodes and the labels that the servicer passed en route to this node. The final two fields track the status of the labels. The dominated field is initially set to 0 and later is set to 1, if the label has been superseded by another label at the same node with an earlier "cost to here" and more "fuel remaining". As is done in Djikstra's, we mark the node label permanent, if all nodes connected to this node have been labelled, or in other words, if this node has been examined. We can see a summary of these labels in Table 3.5.1. Next we will present how to make, update, and examine the labels and then present the associated pseudocode for labelmaking.

When the heuristics, and thus the label making algorithm are initialized, no tasks are complete and all servicers' current location is their starting node and their time is $t = 0$. For the discussion here, we assume there are no tasks at any of the servicer starting nodes. It is important to note that the Label Making Algorthim is run for each servicer independently, creating all possible labels for that servicer. To begin, We consider a single servicer, d, and create a label for its

Table 3.5.1: Components of labels used in both heuristics.

Field	Description
Servicer	$d \in D$
Node	$i \in N$
Name	Label Number
Cost to Here	$\sum \tau_{ij}$ from servicer current node
Fuel Remaining	$\begin{cases} f_{dt} - \phi_{ij}, \text{if not a refuelling arc} \\ \text{F, if the node ends a refuelling arc} \end{cases}$
Task	$\begin{cases} v \in B, \text{if there is a task at this node} \\ 0, \text{ otherwise} \end{cases}$
Predecessors	Previous nodes
Label numbers of predecessors	Previous nodes' label numbers
Dominated?	$\begin{cases} 1, \text{if ``bested'' by another label} \\ 0, \text{ otherwise} \end{cases}$
Permanent?	$\begin{cases} 1, \text{if label has been examined} \\ 0, \text{ otherwise} \end{cases}$

starting node, i, also marking the label permanent. The first label for servicer d, is as follows: $(d, i, 1, 0, F, 0, [i], [1], 0, 1)$. We next examine each of the nodes connected to node i and create a label for each in the same manner, updating the values for "cost to here" and "fuel remaining" as required. A second label for servicer d at node j connected to node i, without a task, and without refuelling would be as follows: $(d, j, 1, 0+\tau_{ij}, F-\phi_{ij}, 0, [i, j], [1, 1], 0, 0)$. A label for servicer d at node k connected to node i, with a task v, would be as follows: $(d, k, 1, 0+\tau_{ik}, F-\phi_{ik}, v, [i, k], [1, 1], 0, 0)$. A label for servicer d at node ℓ connected to node i, without a task, but with refuelling on arc (i, ℓ) at time $t = 0$ would be as follows: $(d, \ell, 1, 0+\tau_{i\ell}, F, 0, [i, \ell], [1, 1], 0, 0)$. When labels have been created for all nodes connected to node i, we examine the non-permanent labels, and choose the node label with the earliest "cost to here" as our next label. We mark the next node label permanent and make labels for all of the connected nodes.

Some additional considerations when making labels follow. If when marking a node permanent, the label has an incomplete task, we track a potential completion time for that task and remove it from the list of incomplete tasks for this servicer for this round of label making. If there is a potential completion time for all incomplete tasks, we stop label making for this servicer. If when considering a connected node the “cost to here” would be greater than the time horizon, we do not make the label for that node. We also do make the label if the “fuel remaining” would be less than or equal to 0 as we do not want to consider sending a servicer to location where they would be stranded. If we examine a node and a label exists, we check to see if it can be dominated by examining the “cost to here” and “fuel remaining” fields. If the potential new label has an earlier “cost to here” and more “fuel remaining” then the existing label is marked dominated and the new label is created, otherwise no new label is created for this node. We continue through the network, making labels for this servicer until all nodes have been marked permanent, or no more labels can be created because of time or fuel, or because we have a potential completion time for all incomplete tasks. The set of created labels for each servicer are then used in each of the MORSO solving heuristics that follow. We present high level label making psuedocode in Algorithm 4 Label Making.

Algorithm 4 Label Making

```
1: Input: Network $G = (N,A)$ with nodes $i \in N$ and arcs $(i,j) \in A$ have time $\tau_{ij}$ and fuel $\phi_{ij}$
   costs.
2: Input: Servicer $d$, with time $t \in T$, current location $i \in N$, and fuel level $f_{dt}$
3: Input: Set of incomplete tasks $B$ with $v \in B$
4: Create a label for node $i$, make $label_{permanent} = 0$, and $cost\_to\_here = t$
5: while $\exists$ a task without a potential completion time AND $\exists$ time remaining for servicer $d$ do
6:    for All $j : (i,j) \in A$ do
7:       Set $MakeLabel$ = TRUE
8:       if Labels exist at this node then
9:          for Each existing $label$ do
10:            if $label_{dominated} = 0$ then
11:               if $label_{cost\_to\_here} \leq$ potential $cost\_to\_here$ AND
                  $label_{Fuel\_Remaining} \geq$ potential $Fuel\_Remaining$ then
12:                  Set $MakeLabel$ = FALSE
13:               else
14:                  Make $label_{dominated} = 1$ and $label_{permanent} = 1$
15:               end if
16:            end if
17:         end for
18:      end if
19:      if $MakeLabel$ = TRUE then
20:         Create a label for this node
21:      end if
22:   end for
23:   From the non-permanent labels, choose the one with the earliest $cost\_to\_here$ and mark it
      $label_{permanent} = 1$
24:   Let $i =$ the node associated with this label
25: end while
26: Output: Set of labels for servicer $d$
```

3.5.2 Heuristic I: Greedy Task Assignment

The first heuristic we present for solving MORSO is called Greedy Task Assignment, because the servicers are assigned to tasks “greedily” based on the “cost to here” field in the available labels. The Label Making Algorithm is run for all servicers and all of the labels with incomplete tasks are examined. The label with the earliest “cost to here” is chosen from all of the labels that have tasks. The servicer in that label is assigned to the task in the label and the task is marked permanently complete. When the servicer is assigned to the task, the servicer is advanced in time and position to the location and time in the label. Additionally, the servicer fuel is updated to the "fuel remaining" in the label. All of the labels for the just assigned servicer are removed from the set of labels. The just matched task and its associated subtasks are removed from all of the remaining labels. The Label Making Algorithm is run again for the just matched servicer beginning with the servicer's updated location, time, and fuel level and the just completed task removed from the incomplete task list. We continue in this manner until all tasks have been completed, or all servicers are advanced as far as possible within the time horizon. The high level psuedocode for this heuristic is presented in Algorithm 5 Greedy Task Assignment.

Algorithm 5 Greedy Task Assignment

```
 1: Input: Network $G = (N,A)$ where nodes $i \in N$ have an associated time $t_i \in T$ and arcs $(i,j) \in A$ have
    time $\tau_{ij}$ and fuel $\phi_{ij}$ costs.
 2: Input: Set of servicers $D$ with their associated starting location, time, current location, and fuel capac-
    ity
 3: Input: Set of incomplete tasks $B$ and subtasks $B_v$ for $k \in B_v$ and $v \in B$
 4: for each servicer $d \in D$ do
 5:     Label Making Algorithm
 6: end for
 7: Examine all node labels which have tasks; choose the label with the smallest cost_to_here, call this the
    chosen label.
 8: while Any task is incomplete or while any servicer time is less than time horizon do
 9:     Assign the servicer to the task in the chosen label
10:     Advance the servicer to the node and time in the chosen label, and update the servicer fuel level
        based on the fuel remaining in the chosen label.
11:     Mark the task in the chosen label complete
12:     Remove the task from the set of incomplete tasks
13:     Remove all labels from the set of labels for the servicer in the chosen label
14:     Remove the task and subtask in the chosen label from all the remaining labels
15:     Do Label Making Algorithm for the just assigned servicer.
16:     if Node labels with tasks exist then
17:         Examine all node labels which have tasks; choose the label with the smallest cost_to_here, call
            this the chosen label.
18:     else
19:         Advance all servicer current time values to end of the time horizon.
20:     end if
21: end while
22: Output: Completed tasks and associated labels indicating when and by which servicer
```

3.5.3 Heuristic II: Greedy Clustering Task Assignment

The next heuristic we present for solving MORSO is “Greedy Clustering”. Tasks are examined in clusters to assign a servicer to an area of the network where more tasks are potentially available. To speed up the active processing, we preprocess the input data, specifically the tasks and the network to create a lookup table of “clustered tasks”. We use Dijkstra’s Algorithm on a time expanded version of the network to find the shortest path, based on time, from every subtask to every other task to create the lookup table. With the lookup table created, we proceed as described in Algorithm 5 Greedy Task Assignment, but when the initial set of labels are created, we use the lookup table to match each label that contains a task, with the next nearest incomplete task. We then examine each “clustered” pair of tasks and choose the cluster with the earliest combined time. We call this combined time the “clustered time”. We choose the label with the earliest “clustered time” and assign the servicer to the task in this chosen label. As in the Greedy Task Assignment algorithm, the task in the chosen label is marked permanently complete and the servicer is advanced in time and position to the location and time in the label. Additionally, the servicer fuel is updated to the “fuel remaining” in the chosen label. All of the labels for the just assigned servicer are removed from the set of labels. The just matched task and its associated subtasks are removed from all of the remaining labels. The Label Making Algorithm is run again for the just matched servicer beginning with the servicer’s updated location, time, and fuel level and the just completed task removed from the incomplete task list. We continue in this manner until all tasks have been completed, or all servicers are advanced as far as possible within the time horizon. The high level psuedocode for this heuristic is presented in Algorithm 6 Greedy Clustering.

Algorithm 6 Greedy Clustering

```
1: Input: Network $G = (N,A)$ where nodes $i \in N$ have an associated time $t_i \in T$ and arcs $(i,j) \in A$ have time
   $\tau_{ij}$ and fuel $\phi_{ij}$ costs.
2: Input: Set of servicers $D$ with their associated starting location, time, current location, and fuel capacity
3: Input: "Lookup Table" with the shortest path from each subtask to all other tasks
4: for Each servicer $d \in D$ do
5:     Label Making Algorithm
6: end for
7: for Every label obtained in Label Making Algorithm which has a task do
8:     Check "Lookup Table" to find the next closest task to the current task and add the time in the current label to
       the time to the next closest task from the table; call this the "clustered time"
9: end for
10: From the list of labels, choose the label with the earliest "clustered time", call this the chosen label
11: while Any task is incomplete or while any servicer time is less than time horizon do
12:     Assign the servicer to the task in the chosen label
13:     Advance the servicer to the node and time in the chosen label, and update the servicer fuel level based on the
        fuel remaining in the chosen label.
14:     Mark the task in the chosen label complete
15:     Remove the task from the set of incomplete tasks
16:     Remove all labels from the set of labels for the servicer in the chosen label
17:     Remove the task and subtask in the chosen label from all the remaining labels
18:     Do Label Making Algorithm for the just assigned servicer.
19:     if Node labels with tasks exist then
20:         for Every label obtained in Label Making Algorithm which has a task do
21:             Check "Lookup Table" to find the next closest task to the current task and add the time in the current
                label to the time to the next closest task from the table; call this the "clustered time"
22:         end for
23:         From the list of labels, choose the label with the earliest "clustered time", call this the chosen label
24:     else
25:         Advance all servicer current time values to end of the time horizon.
26:     end if
27: end while
28: Output: Completed tasks and the associated labels indicating when and by which servicer
```

3.6 Computational Results

We used each of the solving algorithms in the previous section and CPLEX to solve the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) problem, and in this section will compare the performance of each of the two solving heuristics with CPLEX. Convention when comparing the performance of constructive heuristics with a commercial solver is to examine optimality gaps. In MORSO we are seeking to maximize the weighted number of completed tasks, thus the objective values are always some multiple of the number of tasks, which is 8 or 16 in this work. For this reason, we examined the objective values as well as the optimality gaps when comparing the solving methods. We proceed with descriptions of the computing resources, test design, and then present the comparisons of each method with CPLEX.

When solving MORSO for this work, each of the 1,860 runs from the test design in Table 3.6.1 was set to solve with CPLEX and each of the heuristics. The CPLEX instances were run using resources from a high performance computing center public standard nodes with two Xeon Gold 6130 processors, 32 cores, and 192 GB of memory. For the CPLEX instances we set a solving time limit of 12 hours for the runs with a 24 hour time horizon and a solving time limit of 24 hours for the 48 hour time horizon runs. All heuristic instances were run on a single Apple Mac Mini with an M1 processor and 16 GB of RAM. Next we cover the network and composition of the 1,860 instances which were solved with each of the solving methods.

The test design, network and tasks used in these computational tests are based on the case studies presented in Sorenson and Nurre Pinkley [19]. Their case studies and therefore the network used in this work, are based on satellites currently on orbit, but the methods in this work can be used on any network with or without moving tasks and refuelling. All that is needed are the time and fuel costs for the arcs in the network and the starting locations of the servicers, tasks, and refuelling depots. We created a test design in which we varied the number of servicers, the number of refuelling depots, the starting locations of servicers and refuelling depots, the fuel capacity of the servicers, and the time horizon. The factors and levels of the test design can be seen in Table 3.6.1.

Table 3.6.1: Factors and levels for the test cases. The combination of these factors and levels resulted in $1,860$ test cases which were solved with CPLEX and each of the heuristics.

Factor	**Levels**
Number of Robot Servicers	1, 2, 3, 4, 5
Number of Refuelling Depots	1, 2, 3, 4, 5
Refuelling Depot Starting Locations (node numbers)	12, 32, 80, 125, 150
Robot Servicer Fuel Capacity	10, 15, 25
Number of Tasks	8, 16
Time Horizon	24, 48

For these runs, we used both time expanded and non time expanded networks. A time expanded network has all the same information as a non time expanded network, with the exception that the node time and node name are combined. Our time increment is 15 minutes, thus a 24 hour time horizon has 96 time steps and the 48 hour time horizon has 192 steps. If we consider a non time expanded network with 162 nodes and 1,627 arcs and a time horizon of 24 hours this becomes a time expanded network with 15,552 nodes and 65,188 directed arcs. Likewise the 48 hour time expanded network has 31,104 nodes and 311,132 directed arcs. The benefit of the time expanded network is that one less piece of data is tracked for each servicer and task because the node time and node name are combined.

3.6.1 Greedy Task Assignment

We compare the performance of CPLEX and Greedy Task Assignment for the 24 hour time horizon and the 48 hour time horizon separately. We begin with the results for the 930 runs with a 24 hour time horizon. A numeric summary of the results organized by the number of tasks, number of servicers, and the servicer fuel capacities can be seen in Table 3.6.2. For the CPLEX solutions, each case was given a 12 hour solving time limit to obtain a solution. CPLEX found an integer optimal solution within 12 hours in 523 of 930 cases. In 407 of 930 cases, CPLEX presented the best solution obtained at the 12 hour time limit. The distribution of the non optimal optimality gaps for these cases can be seen on the left in Figure 3.6.1.

The total time to complete the CPLEX runs for the 24 hour time horizon was almost 233 days. The median time to complete a run with a 24-hour time horizon with CPLEX is 3.6 hours and the mean is 6 hours. Greedy Task Assignment optimality gaps can be seen on the right in Figure 3.6.1. The total time to complete the same 930 runs with Greedy Task Assignment for the 24 hour time horizon was less than two days. The median time to complete a run with a 24-hour time horizon with Greedy Task Assignment was 2.1 minutes and the mean was 2.8 minutes. A more in depth breakdown of the time differences grouped by the number of tasks, the number of servicers, and the servicer starting fuel level can be seen in Table 3.6.2. On average, in every

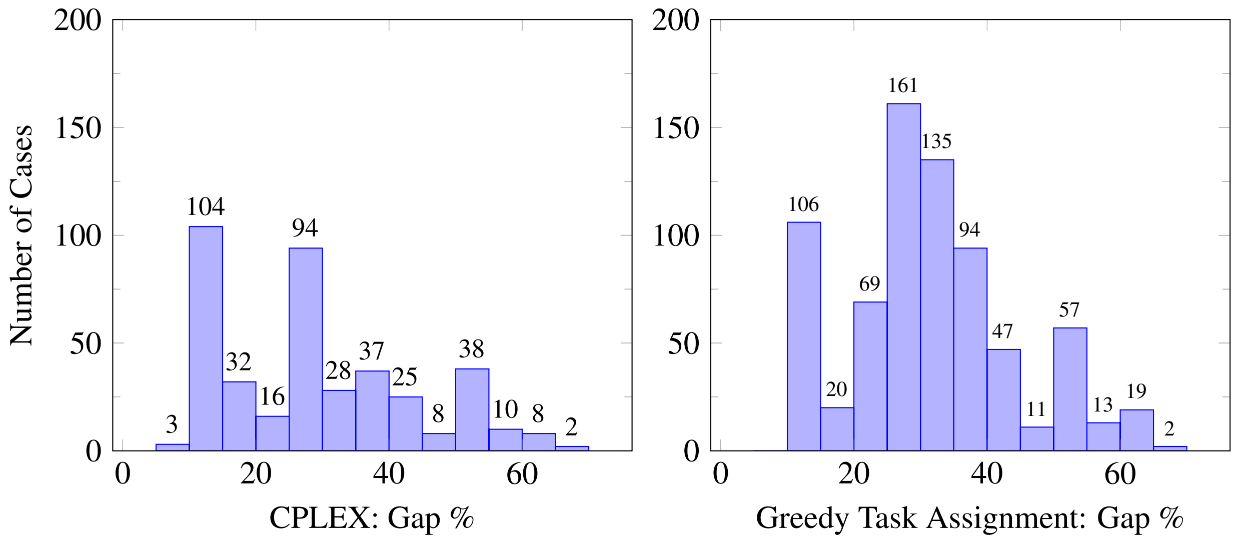

Figure 3.6.1: Comparison of the distributions of the non-zero optimality gap values for Greedy Task Assignment (right) and CPLEX (left) with a 24 hour time horizon.

case, Greedy Task Assignment was faster than CPLEX when solving the 24 hour time horizon runs in our case study.

Now we consider the objective values, we see a visual comparison in Figure 3.6.2. For each run, we subtracted the heuristic objective value from the CPLEX objective value, or best upper bound if no solution was reported. We can see here that Greedy Task Assignment matched or exceeded the objective value for CPLEX in just over 38% of the cases, and was within one task of the CPLEX objective value for nearly 77% of the cases. This translates to a 99.2% reduction in the median solving time to be within one task of CPLEX 77% of the time. Next we present the results for the 48 hour time horizon cases.

We next consider the same 930 runs with a 48 hour time horizon to compare Greedy Task Assignment and CPLEX. For the CPLEX solutions, the runs were each given a 24 hour solving time limit to obtain a solution. CPLEX found an integer optimal solution within 24 hours in 133 of 930 cases. In 730 of 930 cases, CPLEX presented the best solution obtained at the 24 hour time limit and in 67 cases CPLEX failed to find any solution within the 24 hour solving time. A numeric summary of the results organized by the number of tasks, number of servicers, and the servicer fuel capacities can be seen in Table 3.6.3. The 67 runs which did not achieve a solution

Table 3.6.2: A comparison of the optimality gaps and solving times for Greedy Task Assignment and CPLEX for runs with a 24 hour time horizon. In the last two columns, negative values indicate that, on average, the heuristic outperformed CPLEX for those instances.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	0.00	587	6.45	25	6.45	-561
		15	31	0.00	428	16.13	46	16.13	-382
		25	31	0.00	194	30.10	49	30.10	-145
	2	10	31	28.89	29499	39.33	72	10.44	-29426
		15	31	3.38	15530	15.64	92	12.26	-15438
		25	31	5.00	14987	23.35	88	18.36	-14898
	3	10	31	41.86	42449	49.56	99	7.70	-42350
		15	31	16.83	39441	20.87	224	4.03	-39217
		25	31	9.27	38080	20.85	200	11.58	-37879
	4	10	31	28.23	41877	38.08	128	9.85	-41749
		15	31	5.24	20313	11.69	231	6.45	-20081
		25	31	0.81	4113	12.10	183	11.29	-3929
	5	10	31	17.34	38944	27.94	153	10.60	-38791
		15	31	0.00	2498	1.21	225	1.21	-2273
		25	31	0.00	1233	1.21	226	1.21	-1006
16	1	10	31	0.00	182	5.38	33	5.38	-148
		15	31	0.00	137	21.50	64	21.50	-73
		25	31	0.00	112	12.90	71	12.90	-41
	2	10	31	16.60	28863	25.04	66	8.44	-28796
		15	31	1.70	13401	18.93	127	17.23	-13273
		25	31	0.36	2596	20.03	177	19.67	-2419
	3	10	31	24.74	40861	34.61	162	9.87	-40699
		15	31	4.56	15681	22.98	271	18.43	-15409
		25	31	2.25	9765	20.68	216	18.44	-9549
	4	10	31	40.98	42274	47.18	136	6.20	-42138
		15	31	22.98	36504	34.63	354	11.65	-36149
		25	31	16.98	38155	33.95	257	16.97	-37897
	5	10	31	45.98	43215	50.07	169	4.09	-43045
		15	31	26.65	43212	35.13	446	8.48	-42766
		25	31	13.52	43210	30.70	469	17.18	-42741

are annotated in the "Number of Runs" column with the exponent indicating the number of runs that failed to get any solution within the solving time limit.

The distribution of the optimality gaps for the cases which did not reach integer optimal-

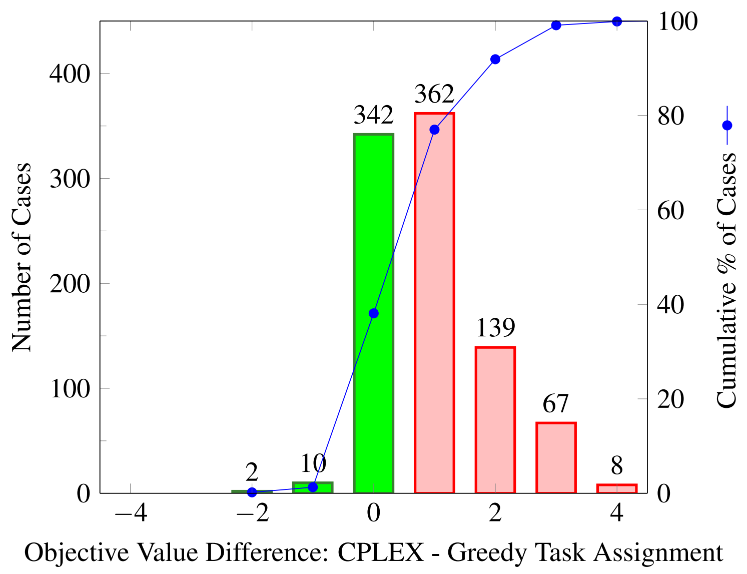

Figure 3.6.2: A comparison of the objective values for Greedy Task Assignment and CPLEX for runs with 24 Hour time horizon. Greedy Task Assignment matched or outperformed CPLEX in 354/930 cases. (38%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

ity can be seen on the left in Figure 3.6.3. The total time to complete the CPLEX runs for the 48 hour time horizon was almost 873 days. The median solving time for the runs with a 48 hour time horizon with CPLEX was 24 hours, the mean 22.5 hours, and the minimum 2.5 hours. The total time to complete the same 930 runs for the 48 hour time horizon with Greedy Task Assignment was just over 9 days. The median solving time for runs with a 48 hour time horizon using Greedy Task Assignment was 4.3 minutes, the mean 14.3 minutes, and the minimum 8.0 seconds. For a breakdown of the solving times and optimality gaps organized by the number of servicers, the number of tasks, and the servicer starting fuel, see Table 3.6.3. For the 48 hour time horizon, Greedy Task Assignment performed better than CPLEX on average for all of the factor combinations we examined. As the time horizon grows beyond 48 hours, the solving time difference between CPLEX, if it can reach a solution, and Greedy Task Assignment will continue to grow.

A histogram of the Greedy Task Assignment optimality gaps can be seen on the right in Figure 3.6.3 and a visual comparison of the objective values can be seen in Figure 3.6.4. We can

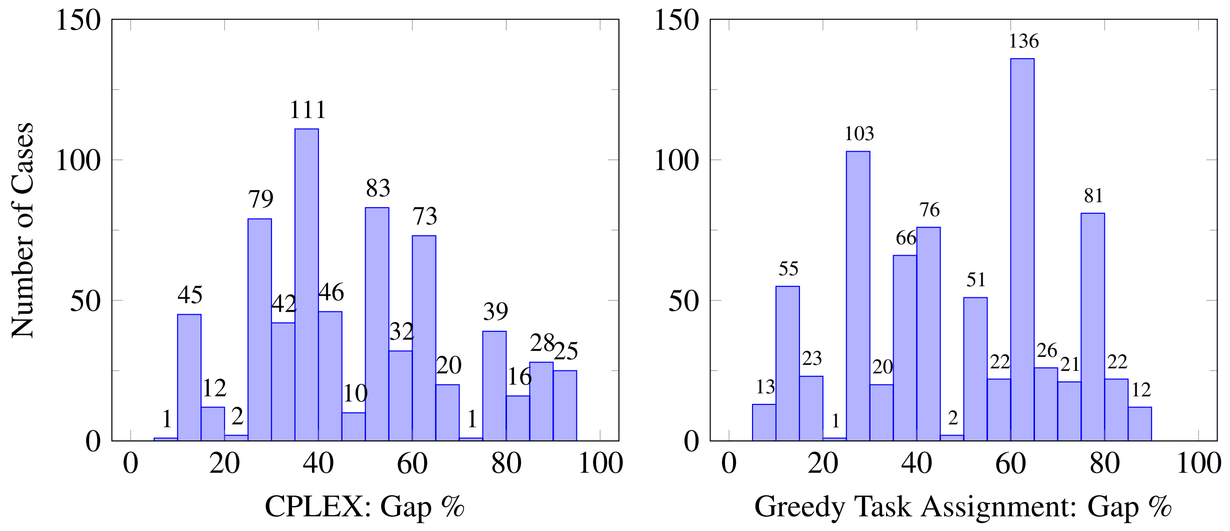

Figure 3.6.3: Distributions of non-zero gap values for Greedy Task Assignment (right) and CPLEX (left) with a 48 hour time horizon

see that Greedy Task Assignment matched or exceeded the objective value for CPLEX in just over 58% of the cases, and was within one task of the CPLEX objective value for nearly 70% of the cases. This equates to a 99.7% reduction in the median solving time to be on average within one task of CPLEX 75% of the time. Overall, when compared to the solving results obtained by CPLEX, Greedy Task Assignment provides results within one task of CPLEX over 70% of the time with a 99% reduction in the solving time. Next we will look at the "Greedy Clustering" heuristic and how it compares with CPLEX.

Table 3.6.3: A comparison of the optimality gaps and solving times for Greedy Task Assignment and CPLEX for runs with a 48 hour time horizon. In the last two columns, negative values indicate that, on average, the heuristic outperformed CPLEX for those instances.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	60.09	84329	79.81	3415	19.72	-80914
		15	31	45.56	86253	68.50	152	22.93	-86100
		25	31	27.42	84374	58.47	173	31.05	-84200
	2	10	31	50.81	86443	69.35	1153	18.55	-85289
		15	31	26.21	86448	44.35	350	18.15	-86098
		25	31	2.82	45294	23.79	40	20.97	-45253
	3	10	31^{3}	62.50	86411	56.45	1633	2.82	-85126
		15	31	27.82	80709	18.55	368	-9.27	-80341
		25	31	0.00	37098	0.00	36	0.00	-37062
	4	10	31^{5}	73.79	86418	35.08	1371	-16.94	-85627
		15	31	29.84	85424	0.81	404	-29.03	-85020
		25	31	3.63	57625	0.00	37	-3.63	-57588
	5	10	31^{7}	83.06	86409	18.55	1698	-28.23	-85524
		15	31^{4}	58.47	85144	0.00	440	-32.66	-85169
		25	31^{3}	22.18	72622	0.00	43	-2.82	-72926
16	1	10	31	61.25	86430	77.73	4065	16.48	-82364
		15	31	46.60	86431	74.62	262	28.02	-86169
		25	31	13.92	63166	48.27	180	34.36	-62985
	2	10	31	66.33	86420	72.38	1092	6.05	-85328
		15	31	46.98	86428	62.90	565	15.93	-85862
		25	31	33.27	86431	44.15	102	10.89	-86328
	3	10	31^{5}	83.06	86408	60.48	1830	-9.27	-85158
		15	31^{1}	50.20	86413	45.97	656	-0.60	-87075
		25	31	27.82	86414	25.00	76	-2.82	-86338
	4	10	31^{8}	91.73	86412	50.60	1558	-13.31	-85783
		15	31^{1}	67.14	86415	27.22	623	-34.68	-86914
		25	31^{2}	44.56	86497	7.46	74	-25.00	-86654
	5	10	31^{18}	95.56	86404	40.12	2650	20.36	-85844
		15	31^{6}	73.59	86410	14.52	663	-23.59	-86443
		25	31^{4}	62.30	86411	0.00	88	-36.49	-86786

$\#^{i}$ where *i* indicates the number of cases which failed to achieve any CPLEX solution within the 24 hour solving period.

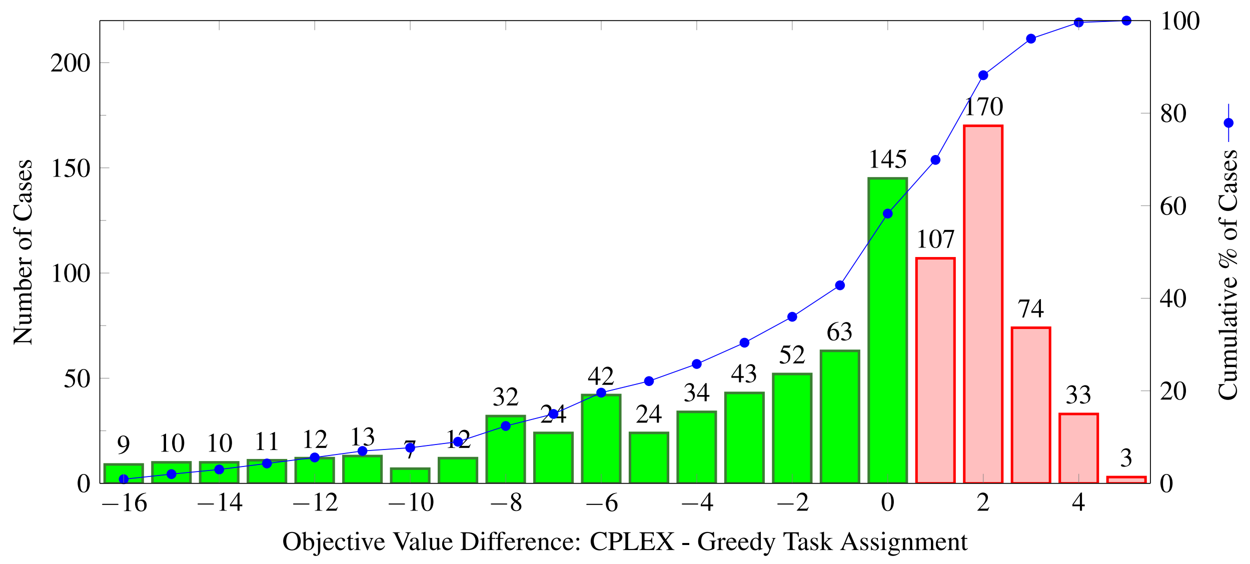

Figure 3.6.4: A comparison of the objective values for Greedy Task Assignment and CPLEX for runs with 48 hour time horizon. The heuristic matched or outperformed CPLEX in 543/930 cases. (58%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

3.6.2 Greedy Clustering

We next compare the results when solving the test cases using Greedy Clustering and CPLEX. A major difference between Greedy Task Assignment and Greedy Clustering is the pre-processing required to obtain a table of lookup values with the shortest path from each subtask to every other task. We add in the pre-processing time when we compare the solving times to ensure that all computational time is included. For the 24 hour time horizon, the lookup table creation took approximately 8 hours on a single Apple Mac Mini with an M1 processor and 16 GB of RAM. For the 48 hour time horizon, the lookup table took approximately 24 hours on two Xeon E5-2680 processors with 32 cores and 192 GB of memory. We consider the same 930 cases for the 24 and 48 hour time horizons in this section. We begin with the 24 hour time horizon.

A numeric summary of the optimality gap and solving time comparisons between Greedy Clustering and CPLEX are organized by the number of tasks, number of servicers, and the servicer starting fuel in Table 3.6.4. We can see a visual comparison of the optimality gaps in Figure 3.6.5 and of the objective function values in Figure 3.6.6. Greedy Clustering matched or exceeded CPLEX in just over 45% of the 930 cases (420/930) and was within one task of CPLEX in 85% of the cases (791/930).

The Greedy Clustering runs took just over 42 hours in total, and if we add in the lookup table preprocessing time, this equates to just over 50 hours, which is still a significant reduction from the CPLEX total processing time. The total time to complete the CPLEX runs for the 24 hour time horizon was almost 233 days. The median time to complete a run with a 24-hour time horizon with CPLEX is 3.6 hours and the same is 2.3 minutes using Greedy Clustering. The mean processing time for a run with a 24 hour time horizon with CPLEX is 6 hours and 2.7 minutes with Greedy Clustering. This is a 99% reduction in processing time to obtain a solution within one task of the CPLEX solution 85% of the time. In all of the runs with a 24 hour time horizon, Greedy Clustering performed no worse than 3 tasks different than CPLEX and solved faster than CPLEX in all but three (3/930) cases.

The Greedy Clustering runs with a 48 hour time horizon took much longer than the runs with a 24 hour time horizon. The total processing time for the runs with a 48 hour time horizon, was just over 1,188 hours and just over 1,212 hours with the pre-processing included. The median Greedy Clustering processing time for a run with a 48 hour time horizon was 1.2 hours, the mean 1.3 hours, and the minimum 8.3 seconds. The total time to complete the CPLEX runs for the 48 hour time horizon was almost 873 days. As stated earlier, the median CPLEX solving time for the runs with a 48 hour time horizon was 24 hours, the mean 22.5 hours, and the minimum 2.5 hours. A breakdown of the solving times and optimality gaps grouped by the number of tasks, number of servicers, and the servicer fuel capacity is shown in Table 3.6.5. On average, Greedy Clustering results in a 94% reduction over the CPLEX solving time for runs with a 48 hour time horizon.

In 67 of the 930 runs with a 48 hour time horizon, CPLEX failed to achieve any solution within the 24 hour solving period. In Table 3.6.5, in column, "Number of Runs", the exponent indicates how many of the runs failed to get any solution in the 24 hour solving time. A comparison of the optimality gaps for the runs which failed to reach an integer optimal solution are shown

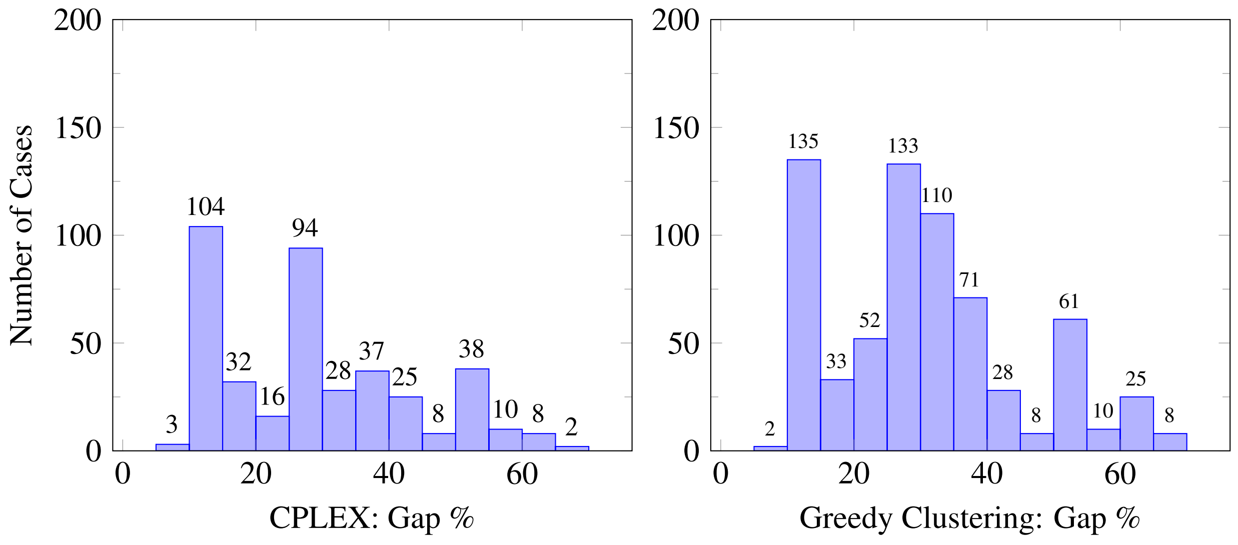

Figure 3.6.5: Distributions of non-zero gap values for Greedy Clustering (right) 254 and CPLEX (left) 523 with a 24 hour time horizon

in Figure 3.6.7 and a comparison of the objective values is shown in Figure 3.6.8. Greedy Clustering performed as well as or better than CPLEX in nearly 59% of the cases (506/930) and was within one task of CPLEX in nearly 73% of the cases (628/930).

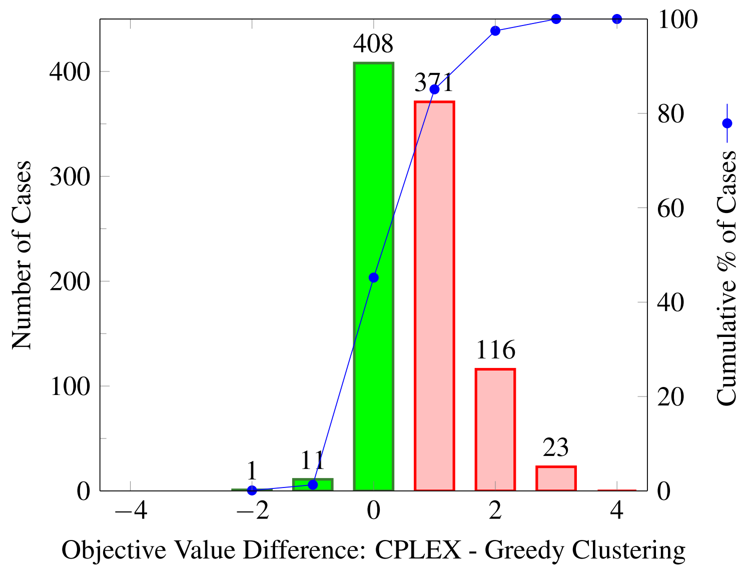

Figure 3.6.6: A comparison of the objective values for Greedy Clustering and CPLEX for runs with 24 Hour time horizon. Greedy Clustering matched or outperformed CPLEX in 420/930 cases. (45%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

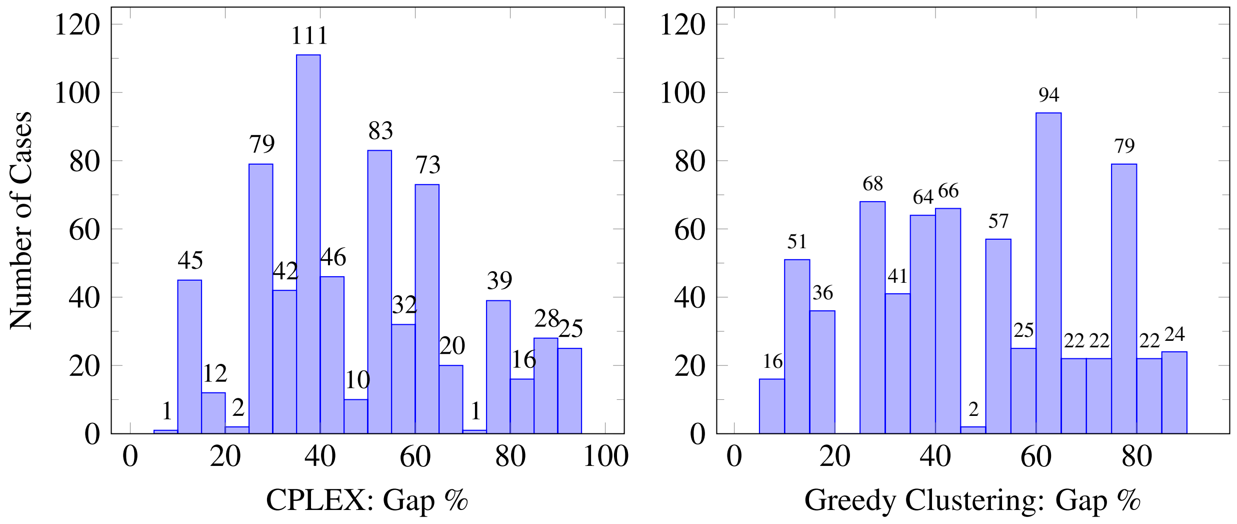

Figure 3.6.7: Distributions of non-zero gap values for Greedy Clustering (right) and CPLEX (left) for the cases/runs with a 48 hour time horizon

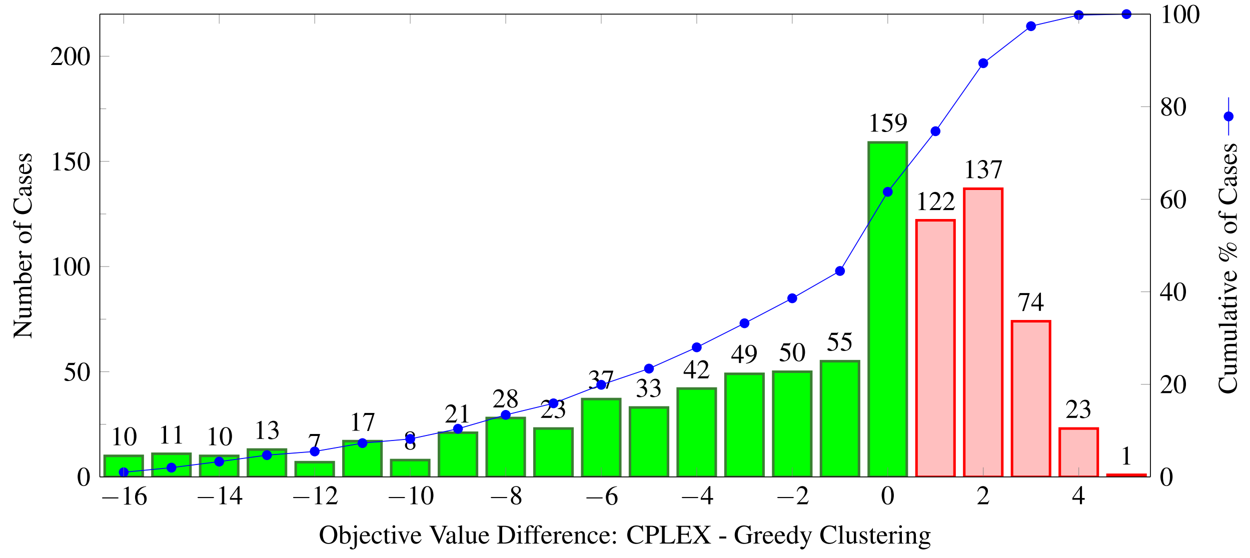

Figure 3.6.8: A comparison of the objective values for Greedy Clustering and CPLEX for runs with 48 hour time horizon. Greedy Clustering matched or outperformed CPLEX in 543/930 cases. (61.6%) Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

Table 3.6.4: A comparison of the optimality gaps and solving times for Greedy Clustering and CPLEX for runs with a 24 hour time horizon. In the last two columns, negative values indicate average amount that the heuristic outperformed (or not) CPLEX for that combination of factors.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	0.00	587	22.58	25	22.58	-562
		15	31	0.00	428	13.98	45	13.98	-383
		25	31	0.00	194	13.98	89	13.98	-105
	2	10	31	28.89	29499	38.82	52	9.93	-29446
		15	31	3.38	15530	11.12	94	7.74	-15435
		25	31	5.00	14987	8.76	108	3.76	-14878
	3	10	31	41.86	42449	48.31	89	6.45	-42360
		15	31	16.83	39441	19.66	144	2.82	-39296
		25	31	9.27	38080	18.78	202	9.51	-37877
	4	10	31	28.23	41877	37.10	120	8.87	-41756
		15	31	5.24	20313	11.69	187	6.45	-20126
		25	31	0.81	4113	12.10	241	11.29	-3871
	5	10	31	17.34	38944	25.81	150	8.47	-38794
		15	31	0.00	2498	0.00	234	0.00	-2263
		25	31	0.00	1233	0.00	311	0.00	-921
16	1	10	31	0.00	182	5.38	36	5.38	-146
		15	31	0.00	137	26.61	63	26.61	-73
		25	31	0.00	112	0.00	67	0.00	-44
	2	10	31	16.60	28863	23.86	73	7.26	-28790
		15	31	1.70	13401	23.47	161	21.77	-13239
		25	31	0.36	2596	12.41	148	12.05	-2448
	3	10	31	24.74	40861	31.58	107	6.84	-40753
		15	31	4.56	15681	20.01	201	15.45	-15480
		25	31	2.25	9765	14.40	225	12.15	-9540
	4	10	31	40.98	42274	44.60	183	3.62	-42091
		15	31	22.98	36504	31.96	324	8.99	-36180
		25	31	16.98	38155	29.69	300	12.71	-37855
	5	10	31	45.98	43215	50.47	179	4.49	-43035
		15	31	26.65	43212	30.69	354	4.03	-42858
		25	31	13.52	43210	26.65	384	13.13	-42825

Table 3.6.5: A comparison of the optimality gaps and solving times for Greedy Clustering and CPLEX for runs with a 48 hour time horizon. In the last two columns, negative values indicate average amount that the heuristic outperformed (or not) CPLEX.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	60.09	84329	82.5	2089	22.41	-82240
		15	31	45.56	86253	72.77	1430	27.21	-84823
		25	31	27.42	84374	50.83	1957	23.41	-82416
	2	10	31	50.81	86443	70.56	1808	19.76	-84635
		15	31	26.21	86448	45.56	2666	19.35	-83781
		25	31	2.82	45294	9.68	3293	6.85	-42000
	3	10	31^{3}	58.48	86412	53.57	2849	-4.91	-83563
		15	31	27.82	80709	15.73	3908	-12.1	-76800
		25	31	0	37098	0	4776	0	-32322
	4	10	31^{5}	68.75	86421	33.17	3871	-35.58	-82550
		15	31	29.84	85424	0	5351	-29.84	-80073
		25	31	3.63	57625	0	6114	-3.63	-51510
	5	10	31^{7}	78.13	86412	15.63	4940	-62.5	-81471
		15	31^{4}	52.31	84958	0	6646	-52.31	-78311
		25	31^{3}	13.84	71145	0	7555	-13.84	-63590
16	1	10	31	61.25	86430	77.73	3391	16.48	-83039
		15	31	46.6	86431	74.62	1729	28.02	-84702
		25	31	13.92	63166	43.54	2372	29.62	-60793
	2	10	31	66.33	86420	72.18	2734	5.85	-83685
		15	31	46.98	86428	62.5	3514	15.52	-82914
		25	31	33.27	86431	39.11	4262	5.85	-82168
	3	10	31^{5}	79.81	86409	59.62	4332	-20.19	-82076
		15	31^{1}	48.54	87656	42.92	5147	-5.63	-82508
		25	31	27.82	86414	16.94	6250	-10.89	-80164
	4	10	31^{8}	88.86	86416	47.01	5307	-41.85	-81109
		15	31^{1}	66.04	87455	24.38	6663	-41.67	-80791
		25	31^{2}	40.73	86504	1.08	8161	-39.66	-78343
	5	10	31^{18}	89.42	86411	42.31	4617	-47.12	-81794
		15	31^{6}	67.25	86413	16	8269	-51.25	-78143
		25	31^{4}	56.71	86412	0	9919	-56.71	-76493

$\#^{i}$ where i indicates the number of cases which failed to achieve any CPLEX solution within the 24 hour solving period.

3.6.3 Greedy vs. Greedy Clustering vs. CPLEX Results

In previous sections of this work, we compared each solving heuristic with CPLEX individually. Now we present a comparison of all three solving methods simultaneously. We begin with the solving time. When we compared the solving time for all three methods across all $1,860$ runs, we observed that CPLEX was never the fastest solving method and that Greedy Task Assignment was fastest method most often. Figure 3.6.9 shows the breakdown of the fastest methods.

We also compared the solutions by method and in many cases there was a tie in methods for the best objective value. In the 1,860 cases we examined, 397 resulted in a three-way tie and 490 resulted in a two-way tie for the best solution. Greedy Clustering tied with CPLEX for the best solution most often. For the 973 runs in which there was a single method with the best objective value, 793 of these were obtained with CPLEX, 138 with Greedy Clustering, and 42 with Greedy Task Assignment. Figure 3.6.10 shows the counts for the number of times a solution methodology achieved the best solution value for that run; this shows more than $1,860$ because of the many ties for best solution. We also examined a combination of speed and accuracy. There were 757 cases (of $1,860$) in which the method that had the fastest solve time also achieved at least a tie for the best solution. The distribution of these is shown in Figure 3.6.11.

Finally in Table 3.6.6 we examined each of the combinations of factors and counted how many of the runs a heuristic or CPLEX performed best. We did not break out the heuristic by type because in most cases Greedy Clustering obtained a better solution. We can see that the heuristics performed better than CPLEX with the longer time horizon and that with the shorter time horizon, the heuristics perform as well as CPLEX in many cases. This is a testament to the scalability of the heuristics for a longer time horizon with a smaller time increment. In Table 3.6.7 we present a summary of the solving times for each of the solving methods. In the best case, a heuristic reduced the solving time by 99% and the worst solving time improvement was a 94% reduction in processing time.

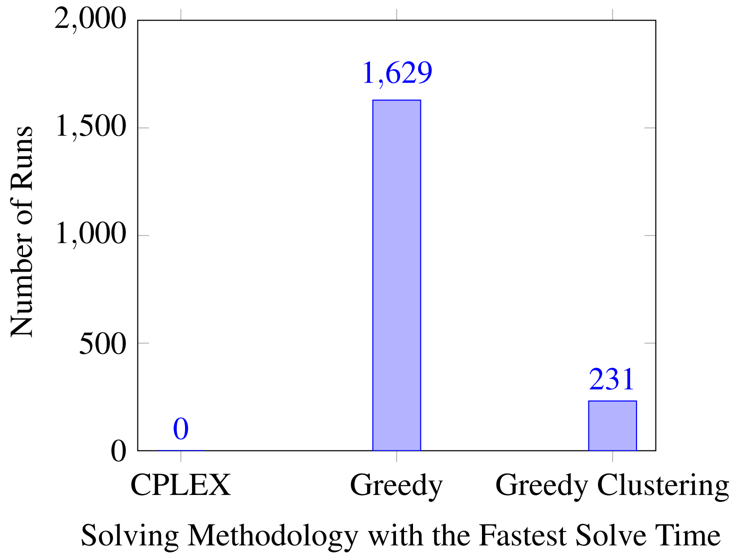

Figure 3.6.9: Histogram showing the number of times each methodology achieved the fastest solve time.

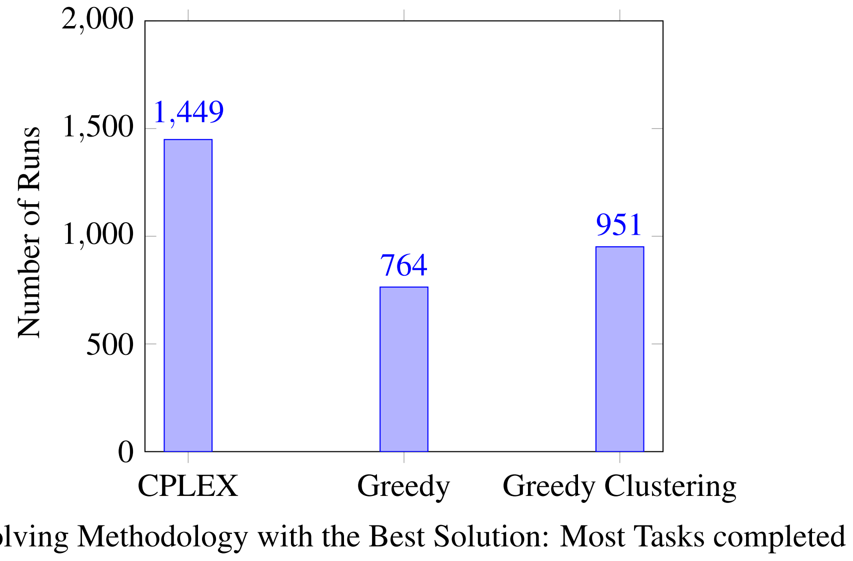

Figure 3.6.10: Histogram showing the number of times each methodology achieved the best solution, alone or in a tie. This has more than 1,860 because there were ties for the best solution on several runs. There were 397 three-way ties and 490 two-way ties for the best solution.

Table 3.6.6: Table which counts the number of instances in which CPLEX or a heuristic obtained a better solution and also if there was a tie. We did not break out the heuristic by type because in most cases Greedy Clustering obtained the better solution. We can see that the heuristics performed better than CPLEX with a longer time horizon and that with the shorter time horizon, the heuristics perform as well as CPLEX in many cases.

Factors			Number of Cases	Time Horizon					
				24			48		
Number of Tasks	Number of Servicers	Starting Fuel		CPLEX	CPLEX and Heuristic Tie	Heuristic	CPLEX	CPLEX and Heuristic Tie	Heuristic
8	1	10	62	1	30	0	22	9	0
		15	62	13	18	0	29	2	0
		25	62	13	18	0	29	2	0
	2	10	62	3	28	0	27	4	0
		15	62	12	19	0	27	3	1
		25	62	6	25	0	16	13	2
	3	10	62	6	25	0	7	8	16
		15	62	7	24	0	2	10	19
		25	62	23	8	0	0	31	0
	4	10	62	17	14	0	0	4	27
		15	62	16	15	0	0	5	26
		25	62	28	3	0	0	28	3
	5	10	62	16	12	3	0	0	31
		15	62	0	31	0	0	2	29
		25	62	0	31	0	0	17	14
16	1	10	62	5	26	0	30	1	0
		15	62	24	7	0	31	0	0
		25	62	0	31	0	31	0	0
	2	10	62	10	21	0	20	4	7
		15	62	23	8	0	29	2	0
		25	62	30	1	0	23	7	1
	3	10	62	16	15	0	4	1	26
		15	62	27	4	0	6	11	14
		25	62	30	1	0	4	10	17
	4	10	62	13	14	4	1	0	30
		15	62	24	7	0	1	0	30
		25	62	30	1	0	0	0	31
	5	10	62	15	11	5	0	0	31
		15	62	17	12	2	0	0	31
		25	62	29	2	0	0	0	31

Table 3.6.7: Table with solving times for each methodology. The Greedy Clustering Task Assignment times do not include the pre-processing times.

Time Horizon	Solving Method	Solving Time (hours unless indicated)			
		Minimum	Median	Maximum	Total
24 Hours	CPLEX	0.02	3.62	12.00	5582.52
	Greedy Task Assignment	9.94(sec)	0.03	1.20	43.4
	Greedy Clustering	12.13(sec)	0.04	0.36	42.36
48 Hours	CPLEX	2.50	24.01	24.02	20940.55
	Greedy Task Assignment	7.96 (sec)	0.07	11.75	222.27
	Greedy Clustering	0.02	1.15	8.31	1188.43

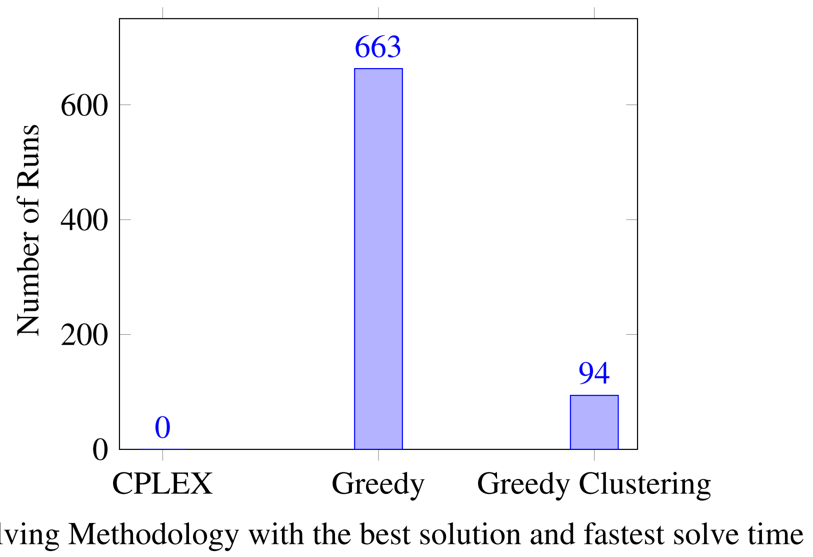

Figure 3.6.11: There were 757 cases in which the method that achieved the fastest solution time also achieved at least a tie for the best objective value solution. Here we see the distribution of those that led in both categories.

3.7 Conclusions

In this work we examined the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing (MORSO) optimization problem which seeks to maximize the weighted number of tasks which can be completed in a set time horizon. First we proved the complexity of the problem with a reduction from the known $NP-Complete$ problem $1 \mid d_j = d \mid w_jU_j$ to an instance of the MORSO problem. Next we presented our two new heuristics for solving the MORSO problem and a node labelling algorithm used in both. The heuristics resulted in a significant reduction in processing times while providing near optimal solutions to the problem.

We also conducted a comparison of the solution paths that the heuristics and CPLEX provided and were not able to discern an appreciable pattern attributable to any of the factors examined in this study. One area for future study could be to use machine learning or other more intensive data mining techniques to determine why the heuristics perform as they do. We also recommend incorporating task processing times into the problem and into the heuristics which would also require modification to the existing mixed integer linear programming model. Another direction for this work would be to examine a larger network with smaller time intervals. The smaller time intervals and larger network would make it more difficult for CPLEX to solve, so ideally much longer solving time wall would be needed for CPLEX to find the best solution possible for comparison with the heuristics.

Bibliography

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J.

[2] Bourjolly, J., Gürtuna, Ö., and Lyngvi, A. (2006). On Orbit Servicing: A Time Dependent, Moving Target Traveling Salesman Problem. *International Transactions in Operational Research*, 13(5):461–481.

[3] British Broadcasting Corporation (2020). Space news: First Ever Space 'Petrol Station' to Be Built in UK. `https://www.bbc.co.uk/newsround/54551557`. Last Accessed: July 4, 2022.

[4] Coene, S., Spieksma, F. C. R., Dutta, A., and Tsiotras, P. (2012). On the Computational Complexity of Peer-to-Peer Satellite Refueling strategies. *INFOR: Information Systems and Operational Research*, 50:88–94.

[5] Corbin, B. A., Abdurezzak, A., Newell, L. P., Roesler, G. M., and Lal, B. (2020). Global Trends in On-Orbit Servicing, Assembly and Manufacturing (OSAM). `https://www.ida.org/research-and-publications/publications/all/g/gl/global-trends-in-on-orbit-servicing-assembly-and-manufacturing-osam`. IDA Document D-13161.

[6] Daneshjou, K., Mohammadi-Dehabadi, A. A., and Bakhtiari, M. (2017). Mission Planning For On-Orbit Servicing Through Multiple Servicing Satellites: A New Approach. *Advances in Space Research*, 60(6):1148–1162.

[7] Du, B., Zhao, Y., Dutta, A., Yu, J., and Chen, X. (2015). Optimal Scheduling of Multispacecraft Refueling Based on Cooperative Maneuver. *Advances in Space Research*, 55(12):2808–2819.

[8] Dutta, A. and Tsiotras, P. (2007). A Greedy Random Adaptive Search Procedure for Optimal Scheduling of P2P Satellite Refueling. In *AAS/AIAA Space Flight Mechanics Meeting*, pages 07–150.

[9] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*, volume 174. freeman San Francisco.

[10] Gürtuna, Ö. and Trépanier, J. (2003). On-Orbit Satellite Servicing: A Space-Based Vehicle On-Orbit Servicing Routing Problem. In *Operations Research in Space and Air*, pages 123–141. Springer.

[11] Hudson, J. S. and Kolosa, D. (2020). Versatile On-Orbit Servicing Mission Design in Geosynchronous Earth Orbit. *Journal of Spacecraft and Rockets*, 57(4):844–850.

[12] Kannon, T. E., Nurre, S. G., Lunday, B. J., and Hill, R. R. (2015). The Aircraft Routing Problem with Refueling. *Optimization Letters*, 9(8):1609–1624.

[13] Lenstra, J. K., Kan, A. R., and Brucker, P. (1977). Complexity of Machine Scheduling Problems. In *Annals of Discrete Mathematics*, volume 1, pages 343–362. Elsevier.

[14] Li, W. J., Cheng, D. Y., Liu, X. G., Wang, Y. B., Shi, W. H., Tang, Z. X., Gao, F., Zeng, F. M., Chai, H. Y., Luo, W.-B., Cong, Q., and Gao, Z. L. (2019). On-orbit Service (OOS) of Spacecraft: A Review of Engineering Developments. *Progress in Aerospace Sciences*, 108:32–120.

[15] NASA (2020). NASA Selects Proposals to Demonstrate In-Space Refueling and Propellant Depot Tech. `https://www.nasa.gov/directorates/spacetech/solicitations/tipping_points/2020_selections`. Last Accessed: July 4, 2022.

[16] NASA (2021). On-orbit Servicing, Assembly, and Manufacturing 1 (OSAM-1). `https://nexis.gsfc.nasa.gov/osam-1.html`. Last Accessed: July 4, 2022.

[17] Northrup Grumman (2020). Mission Extension Vehicle. `https://news.northropgrumman.com/news/releases/northrop-grumman-and-intelsat-make-history-with-docking-of-second-mission-extension-vehicle-to-extend-life-of-satellite`. Last Accessed: July 4, 2022.

[18] qian Chen, X. and Yu, J. (2017). Optimal Mission Planning of GEO On-Orbit Refueling in Mixed Strategy. *Acta Astronautica*, 133:63–72.

[19] Sorenson, S. E. and Nurre Pinkley, S. G. (2022). Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots. Manuscript submitted for publication.

[20] Zhang, J., Parks, G. T., Luo, Y. Z., and Tang, G. J. (2014). Multispacecraft Refueling Optimization Considering the J2 Perturbation and Window Constraints. *Journal of Guidance, Control, and Dynamics*, 37(1):111–122.

[21] Zhang, Q. and Zhang, Y. (2014). On-orbit servicing task allocation for multi-spacecrafts using hdpso. *Applied Mechanics and Materials*, 538(Mechanical, Electronic and Engineering Technologies (ICMEET 2014)):150–153.

[22] Zhang, T. J., Yang, Y. K., Wang, B. H., Li, H. N., Li, Z., and Shen, H. X. (Oct 2019). Optimal Scheduling for Location Geosynchronous Satellites Refueling Problem. *Acta Astronautica*, 163:264–271.

[23] Zhao, Z., Zhang, J., Li, H. Y., and Zhou, J. Y. (2017). LEO Cooperative Multi-Spacecraft Refueling Mission Optimization Considering J2 Perturbation and Target's Surplus Propellant Constraint. *Advances in Space Research*, 59(1):252–262.

[24] Zhou, Y., Yan, Y., Huang, X., and Kong, L. (2015). Optimal Scheduling of Multiple Geosynchronous Satellites Refueling Based on a Hybrid Particle Swarm Optimizer. *Aerospace Science and Technology*, 47:125–134.

[25] Zhou, Y., Yan, Y., Huang, X., and Yang, Y. (2017). Multi-Objective Planning of a Multiple Geostationary Spacecraft Refuelling Mission. *Engineering Optimization*, 49(3):531–548.

4. Solution Methods for the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots with Known Task Times

Susan E. Sorenson Sarah G. Nurre Pinkley

Abstract: The Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times optimization problem determines how to best route and schedule a fleet of highly maneuverable and refuellable space robot servicers to complete a set of tasks with known task times on satellites orbiting in space. We formulate the problem as a mixed integer linear program seeking to maximize the weighted number of completed tasks within a set time horizon. The model has constraints related to the the movements of the robot servicers, the refuelling depots, and the tasks. We also present two constructive heuristics for solving the problem and compare the results with CPLEX.

4.1 Introduction

The Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times (MORSO-KTT) determines how to best schedule and route a set of space based robotic servicers to complete a set of tasks with known task times on other satellites spanning multiple orbits in space. When servicing satellites that are already on orbit, it is called On-Orbit Servicing (OOS). Servicing events can include activities like refuelling, refitting, refurbishing, repairing, inspecting, augmenting, or upgrading. OOS is not a common occurrence yet, but many countries and commercial enterprises have accomplished or are planning to accomplish OOS in the future [4, 12].

NASA completed it's first OOS to repair the Hubble telescope in 1993 [13] and is developing technologies like the the On-orbit Servicing, Assembly and Manufacturing-1, a robotic spacecraft designed to extend the life of satellites, even if they were not designed for OOS [14]. A commercial entity, Orbit Fab placed a refuelling depot in Low Earth Orbit (LEO) in 2021 and plans to place another in Geostationary Earth Orbit (GEO) in cooperation with SpaceX in the near future [20]. As these technologies continue to grow and robots which can travel across multiple orbital bands to accomplish OOS become commonplace, the optimization of the scheduling and routing of these robot servicers to accomplish multiple tasks spanning multiple orbits will grow.

The purpose of this work is to present a new model for MORSO-KTT along with two algorithms for solving. The MORSO problem was previously studied in [19] but in their work, the tasks had instantaneous processing times. For a servicer to accomplish a task, they must negotiate and rendezvous with the satellite that has the task, and continue to orbit with them while the task is processed. In reality, the actual task processing times would be unknown until the servicer arrives at the task location and has completed the negotiation and rendezvous portion of the mission. Task processing times are an integral part of this problem and are for the most part uncertain, thus in this work we take the first step towards the problem with stochastic processing times by examining the problem with deterministic task processing times. We present a new mixed integer linear programming (MILP) model for solving the MORSO-KTT problem. The model accounts for the movement of the servicers, tasks, and refuelling depots. The model also uses sets of arcs to represent the time the servicer must stay with the task for processing and completion.

After we establish the model, we then present two constructive heuristics for solving the MORSO-KTT problem. We use a realistic case study with a network and tasks based on satellites currently on orbit to compare the the solving times and solution accuracy of the heuristics with CPLEX.

Main Contributions. The main contributions of this work are as follows: (i) We present a new MILP formulation for the MORSO-KTT problem; (ii) We develop and demonstrate two constructive heuristics for the solving the MORSO-KTT problem; (iii) We perform extensive computational experiments on a realistic data set based on active satellites demonstrating the speed and accuracy of our algorithms.

The remainder of this work is as follows. In Section 4.2, we summarize literature related to the modeling and optimization of different types of OOS missions. In Section 4.3, we provide a detailed overview of the MORSO-KTT problem and the MILP. In Section 4.4, we present two heuristic solution methods and in Section 4.5 we demonstrate the speed and accuracy of our methods. In Section 4.6, we present conclusions and areas for further study.

4.2 Literature

In this section, we focus our literature review on works related to the modeling of on-orbit servicing (OOS) that address the scheduling of tasks with known or unknown processing times and methodologies for solving these same OOS problems. We begin with a definition of OOS and defining terminology used in this work.

In a work which reviewed OOS technologies defined it as follows: *[O]n-orbit activities conducted by a space vehicle that performs up-close inspection of, or results in intentional and beneficial changes to, another resident space object. These activities include non-contact support, orbit modification (relocation) and maintenance, refueling and commodities replenishment, upgrade, repair, assembly, and debris mitigation [6].* This definition considers on-orbit refuelling (OOR) as a type of OOS, and with this as motivation, when we refer to OOS in this work, it may or may not include OOR activities, but if we use the term OOR then we are specifically referring to the specific OOS task of OOR. Additionally, when we use the term "refuelling" we are referring to the refuelling of the robot servicers that we are routing and scheduling throughout this work.

We begin with some works that examine OOS or OOR missions and assume known and fixed task processing times. Daneshjou et al. [5] use Particle Swarm Optimization to minimize the mission time and fuel consumed for a set of OOS tasks. Their problem has three service types with different task times for each type of service. Zhang et al. [22] also sought to minimize the mission time and fuel consumed for a single servicer to accomplish multiple OOS tasks in multiple orbits with known rendezvous and service times. They use a hybrid encoding genetic algorithm to optimize the sequence of tasks. Gürtuna and Trépanier [10] modeled a single servicer accomplishing multiple tasks, with identical known processing times, as a time dependant Vehicle Routing Problem (VRP) and used the Clarke and Wright algorithm to minimize the fuel expended. [16] propose an optimization framework for a multi-servicer OOS mission with fixed servicing times using a MILP model, but their framework and model is focused at the strategic level with decisions about servicer types and servicing tools.

Some authors who examine OOS or OOR do not make a distinction between those that do the servicing and those that receive the servicing. These works address a class of problems which involve peer-to-peer servicing or refuelling. Peer-to-peer OOR with a fixed servicing time was examined by Dutta and Tsiotras [8] and the authors sought to minimize the overall fuel expended using a greedy random adaptive search procedure. Later in [9], they used a network flow model with a fixed servicing time to determine the minimum fuel required to conduct a set of peer-to-peer OOR operations. Shen and Tsiotras [17] treat the scheduling of a peer-to-peer OOR as a matching problem looking to minimize mission time and all refuelling events occur simultaneously within a set time horizon.

Other works do not specifically address the service times or assume service is accomplished instantaneously. Bourjolly et al. [3] models a single servicer to many OOS tasks as a time dependant Travelling Salesman Problem (TSP) assuming instantaneous task completion upon rendezvous. They solve it with an exhaustive search when there are less than 10 tasks and use a Tabu search when there are more than 10 tasks. qian Chen and Yu [15] examine OOS, with both a single servicer and multiple servicers, looking to minimize the overall cost and fuel used, without specifically addressing the servicing times. They solve the problem with a particle swarm optimization optimizing decisions concerning the fuel and path. Yu et al. [21] examined a single servicer OOS scheduling problem on a single orbit. They used a multi-objective optimization model, minimizing the fuel used while maximizing the weighted number of refuelling tasks for a single servicer and did not consider servicing time, only the servicing order. In the initial presentation of Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing as a mixed integer linear program (MILP) in [19], tasks were accomplished instantaneously.

Some works make the servicing time the decision variable. Alfriend et al. [2] treats a single orbit OOR problem as a TSP, looking to minimize the total fuel expended, by deciding the maneuvers and the service time. In Du et al. [7] look to minimize the expended fuel costs for a OOR mission by deciding the task sequence, rendezvous strategy, and the refuelling time and locations using an algorithm which combines a Multi island Genetic Algorithm and Sequential Quadratic Programming. Their mission format requires the satellites to leave their orbital location to rendezvous with the servicer.

With an overview of some of the existing literature, we proceed by formalizing the MORSO-KTT problem notation and mathematical model.

4.3 Problem Statement

In this section we provide an overview of the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times (MORSO-KTT) problem. We present the nomenclature for the time horizon, network, robot servicers, tasks, and refuelling. We also present an algorithm for creating the subtasks and subtask sets which are used for tracking task movement and completion. We finish with the mixed integer linear program (MILP) formulation. Much of the discussion that follows in this section is heavily based on Sorenson and Nurre Pinkley [19] in which the MORSO problem and a MILP are first introduced. We begin with the problem nomenclature.

MORSO-KTT has a a finite set of time intervals: $t = 0, \dots, |T|$. We also have a network $G = (N, A)$ with nodes $i \in N$ and arcs $(i, j) \in A$. The nodes are at “fixed” locations in space. Arcs exist between nodes if there exists an orbital maneuver such that a robot servicer would be able to get from i to j via some route within the given time horizon. The time and fuel costs, τ_{ij} and ϕ_{ij} respectively, for each arc, are the actual time and fuel costs for the arc, calculated using orbital mechanics. We allow the robot servicers to finish at any node in the network with the designation of a super sink node, E. The super sink node E is connected to every node $i \in N$ with a directed arc (i, E) with unit traversal time, $\tau_{iE} = 1$ and no fuel cost, $\phi_{iE} = 0$. Equivalently, we allow a servicer to end at any node in the network at time $|T|-1$.

The problem also has a set of identical robot servicers D, with $d \in D$. The robot servicers move on the arcs in the network over time. Each servicer has a starting location at a node, $s_d \in N$, and a known maximum fuel capacity, F. The fuel level of each robot servicer, f_{dt}, for $d \in D$ and $t \in T$, changes over time as the servicers move through the network accomplishing tasks. A robot servicer, d, can only move on arc, $(i,j) \in A$ if the fuel level of the servicer f_{dt} exceeds the fuel cost of the arc ϕ_{ij}. Refuelling is allowed only if the arc is a refuelling arc. Refuelling arcs are determined by the starting location of a refuelling depot. As time passes, refuelling depots continually move through the network on arcs. If a refuelling depot is on an arc $(i,j) \in A$ at time $t = t$ this arc then becomes a refuelling arc. Over time the refuelling depot transits many arcs and we create a set of arcs for each time period t: A_t^R where A_t^R is the set of refuelling arcs at time $t \in T, A_t^R \in A$.

The goal when solving the MORSO-KTT problem is to maximize the weighted number of tasks completed within the time horizon. The set of tasks to be completed is called B, and each task $v \in B$ has a weight w_v which is the task priority. Tasks also have a known discrete processing time, p_v and a starting node which is some $i \in N$ at time $t = 0$. As with the refuelling depots, the tasks are also always moving. We break the tasks into subtasks adding a subtask for each node the task passes. A servicer could start processing a task at any one of these subtask nodes. Therefore each subtask $k \in B_v$ has a starting node represented by a node and time pair: (n_{vk}, t_{vk}). We first create all the subtasks by determining their starting node and time pairs. Task servicing can begin at any one of these subtask starting node and time pairs, but for servicing to be completed, the servicer must stay with the subtask through the full processing time. To account for this, we create a set of arcs for each subtask which begins at the subtask starting node and time pair, (n_{vk}, t_{vk}). We call this set of arcs Ω_{vk}. We construct Ω_{vk} by adding consecutive orbiting arc beginning at (n_{vk}, t_{vk}) until the time cost of all the included arcs is greater than or equal to the task processing time p_v. In other words, each arc has an associated time cost, τ_{ij}, and the sum of the time costs for the arcs in each Ω_{vk} must be equal to or exceed the processing time, p_v for the task.

For the detailed steps for creating the subtasks and associated sets, see Algorithm 7. For a subtask and therefore a task to be completed, the same servicer must consecutively travel each arc in Ω_{vk} at the associated times.

Algorithm 7 Subtask Set Creation Algorithm

```
1: Input $|T|$ as the time horizon and network $G = (N,A)$.
2: Input task set $B$ where each task $v \in B$ has a known starting node and processing time $p_v$
3: for each task $v \in B$ do
4:     Create subtask set $B_v = \emptyset$.
5:     Set $current\_node$ = starting node.
6:     Set $current\_time = 0$.
7:     while $current\_time \leq |T|$ do
8:         Add subtask to $B_v$ associated with the ($current\_node$, $current\_time$).
9:         Set $next\_node$ = to the next node on the orbit in the rotational direction of the orbit.
10:        Set $current\_time = current\_time + \tau_{current\_node\ next\_node}$.
11:        Set $current\_node = next\_node$.
12:    end while
13:    for each subtask $k \in B_v$ do
14:        for each $(n_{vk}, t_{vk}) \in B_v$ do
15:            Create $\Omega_{vk} = \emptyset$
16:            Let $current\_node = n_{vk}$ and let $current\_time = t_{vk}$
17:            Set $next\_node$ = to the next node on the orbit in the rotational direction of the orbit.
18:            Let $arc\_count$ = $\lceil \frac{p_v}{\tau_{current\_node\ next\_node}} \rceil$
19:            while $|\Omega_{vk}| < arc\_count$ do
20:                Add arc $(current\_node, next\_node), current\_time$ to $\Omega_{vk}$
21:                Set $current\_node = next\_node$
22:                Set $current\_time = current\_time + \tau_{current\_node\ next\_node}$
23:                Set $next\_node$ = to the next node on the orbit in the rotational direction of the orbit.
24:            end while
25:        end for
26:    end for
27: end for
28: Return $B_v,\ \Omega_{vk}\ \forall k \in B_v\ \forall v \in B$
```

The last input to MORSO-KTT is a set of refuelling depots expressed as $\Gamma, r \in \Gamma$, and as the depots move, some of the arcs in A are designated as refuelling arcs, so we have the set $A_t^R \subset A$ which contains the arcs $(i, j) \in A$ which are available for refuelling at time t.

4.3.1 Model

With the overview of the notation presented we next present the MORSO-KTT MILP formulation which includes the formal definition of the decision variables, parameters and sets, and the constraints. The model comes from [18] with † indicating new or modified entries.

Parameters/Sets

$$
\begin{aligned}
T &: \text{Set of time periods, where } |T| \text{ is the last time period}\\
N &: \text{Set of nodes }, i \in N\\
E &: \text{Super sink node, } E \in N\\
B &: \text{Set of tasks}, v \in B\\
\Gamma &: \text{Set of refuelling depots}, r \in \Gamma\\
B_v &: \text{Set of subtasks associated with } v \in B, k \in B_v\\
A &: \text{Set of directed arcs}, (i,j) \in A\\
A_t^R &: \text{Set of refuelling arcs at time } t \in T, A_t^R \subset A\\
\Omega_{vk}{}^\dagger &: \text{Set of arcs}, (i,j) \in A, \text{ and times}, t \in T,\\
&\qquad\qquad\qquad \text{associated with subtask } k \in B_v, \text{ for task } v \in B\\
D &: \text{Set of robot servicers}, d \in D\\
s_d &: \text{Starting node of servicer } d \in D\\
F_d &: \text{Maximum fuel capacity of servicer } d \in D, \text{ in } \Delta V\\
w_v &: \text{Weight of task } v \in B\\
p_v{}^\dagger &: \text{Processing time of task } v \in B, \text{ in time periods}\\
n_{vk} &: \text{The node location of subtask } k \in B_v \text{ of task } v \in B\\
t_{vk} &: \text{First time period of subtask } k \in B_v \text{ of task } v \in B\\
\tau_{ij} &: \text{Time periods to traverse arc } (i,j) \in A\\
\phi_{ij} &: \text{Fuel to traverse arc } (i,j) \in A, \text{ in } \Delta V\\
\Psi_{ij} &: \text{Per time period fuel needed to traverse arc } (i,j) \in A, \text{ in } \Delta V
\end{aligned}
$$

Decision Variables

$$
\beta_{vkd}{}^\dagger = \begin{cases} 1, & \text{if subtask } k \text{ of task } v \text{ is completed by servicer } d \in D, \text{ for } v \in B\\ 0, & \text{otherwise} \end{cases}
$$

$$
f_{dt} = \text{fuel level of servicer } d \text{ at time } t, \text{ for } d \in D \text{ and } t \in T
$$

$$
y_{dijt} = \begin{cases} 1, & \text{if robot servicer } d \text{ initiates move on arc } (i,j) \in A, \text{ at time } t \in T\\ 0, & \text{otherwise} \end{cases}
$$

Decision Variables

$$^{\dagger}\max \sum_{v\in B} w_v \sum_{k\in B_v}\sum_{d\in D}\beta_{vkd} \tag{4.1}$$

$$\text{s.t. } ^{\dagger}\sum_{k\in B_v}\sum_{d\in D}\beta_{vkd}\leq 1, \qquad \text{for } v\in B \tag{4.2}$$

$$^{\dagger}\sum_{\substack{i:(i,n_{vk})\in A\\ t_{vk}-\tau_{in_{vk}}\geq 0}} y_{din_{vk}t_{vk}-\tau_{in_{vk}}} \geq \beta_{vkd}, \qquad \text{for } v\in B, k\in B_v, d\in D \tag{4.3}$$

$$^{\dagger}\sum_{(i,j,t)\in\Omega_{vk}} \frac{\tau_{ij} * y_{dijt}}{p_v} \geq \beta_{vkd}, \qquad \text{for } v\in B, k\in B_v, d\in D \tag{4.4}$$

$$\sum_{\substack{j:(j,i)\in A\\ t-\tau_{ji}\geq 0}} y_{djit-\tau_{ji}} - \sum_{j:(i,j)\in A} y_{dijt} = \begin{cases} -1, & \text{if } i=s_d \text{ and } t=0\\ 1, & \text{if } j=E \text{ and } t=|T|\\ 0, & \text{otherwise}\end{cases}$$

$$\text{for } i\in N, d\in D, t\in T \tag{4.5}$$

$$f_{dt}\leq f_{dt-1} - \left(\sum_{(i,j)\in A}\sum_{s=\max\{t-\tau_{ij},0\}}^{t}\left(y_{dijs}*\Psi_{ij}\right) + \sum_{\substack{(i,j)\in A_t^R\\ t-\tau_{ij}\geq 0}}\left(y_{dijt-\tau_{ij}}*F_d\right)\right)$$

$$\text{for } t\in T\setminus|T|, d\in D \tag{4.6}$$

$$f_{d0}=F_d, \qquad \text{for } d\in D \tag{4.7}$$

$$0\leq f_{dt}\leq F_d, \qquad \text{for } d\in D, t\in T \tag{4.8}$$

$$y_{dijt},\ ^{\dagger}\beta_{vkd}\in\{0,1\}, \qquad \text{for } d\in D, (i,j)\in A, t\in T, v\in B, k\in B_v \tag{4.9}$$

In Equation, (4.1) we present the objective function seeking to maximize the weighted number of tasks completed. In Constraints (4.2), we ensure that across all servicers at most one subtask is completed for each task. In Constraints (4.3), we ensure the robot servicers travel all arcs in a subtask set for the task to be completed. In Constraints (4.5), we balance the flow of robot servicers through the network by ensuring that each robot servicer must start at their designated start node, only leave a node they are at, and must finish at the super sink E. In Constraints (4.6), we update the fuel on-board of each robot servicer at each time based on the prior time period fuel level, movement, and refuelling decisions. In Constraints (4.7), we set the starting fuel level of all robot servicers to the maximum amount. In Constraints (4.8), we force the fuel level of each servicer over time to be between 0 and the maximum capacity. In Constraints (4.9), we place the binary restriction on some decision variables.

With the model presented we next present two new algorithms for solving MORSO-KTT.

4.4 Solution Methodology

In this section, we present two algorithms for solving MORSO-KTT. We begin by expanding on the label contents for the labels created in the "Label Making Algorithm" presented in Sorenson and Nurre Pinkley [18]. The label making scheme builds upon ideas from Dijkstra's algorithm and an aircraft routing with refuelling algorithm [1, 11] to find the shortest paths to tasks that are in motion on a network. We expand the label content to include information about the subtasks and task processing times for the nodes that have tasks. The complete label contents are shown in Table 4.4.1. With the exception of the contents of the labels, the "Label Making Algorithm" from Sorenson and Nurre Pinkley [18] should be used in the solution methodology algorithms in this work.

4.4.1 Heuristic I: Fast Task Assignment

The Fast Task Assignment algorithm is very similar to the "Greedy Task Assignment" algorithm in [18] with a change to the steps for assigning a servicer to a task and marking the task complete. In the "Greedy Task Assignment" algorithm [18], the servicer completed a task by advancing to the location of the matched task. In this work, tasks are not completed instantaneously and have an associated processing time. The modifications to the "Greedy Task Assignment" algorithm in [18] are marked in Algorithm 8. Next we provide an overview of the "Fast Task Assignment" algorithm.

The "Fast Task Assignment" algorithm begins with the network, the servicers, the tasks, and the associated subtasks and subtask sets for the tasks. Labels are created for each of the servicers as described in "Label Making Algorithm" from [18]. Next we examine the labels for all servicers and focus on those which have incomplete tasks indicated in the label. We choose the label with a task that has the earliest "cost to here" value. This represents the task that can be reached the fastest based on the position and time of all of the available servicers. We assign the

Table 4.4.1: Components of labels used in both heuristics. Fields marked with † indicate new fields added in this work. The other fields existed in [18].

Field	Description
Servicer	$d \in D$
Node	$i \in N$
Name	Label Number
Cost to Here	$\sum \tau_{ij}$ from servicer current start
Fuel Remaining	$\begin{cases} f_{dt} - \phi_{ij}, \text{if not a refuelling arc} \\ \text{F, if the node ends a refuelling arc} \end{cases}$
Task	$\begin{cases} v \in B, \text{if there is a task at this node} \\ 0, \text{ otherwise} \end{cases}$
† Subtask	$\begin{cases} k \in B_v, \text{if a subtask begins at this node} \\ 0, \text{ otherwise} \end{cases}$
† Task Processing Time	$\begin{cases} p_v, v \in B, \text{if there is a task at this node} \\ 0, \text{ otherwise} \end{cases}$
Predecessors	Previous nodes
Label numbers of predecessors	Previous nodes' label numbers
Dominated?	$\begin{cases} 1, \text{if “bested” by another label} \\ 0, \text{ otherwise} \end{cases}$
Permanent?	$\begin{cases} 1, \text{if label has been examined} \\ 0, \text{ otherwise} \end{cases}$

servicer to the subtask/task in the label with the earliest “cost to here” value and mark the task complete. When we assign the servicer to the task, we must move the servicer along the arcs in the associated subtask set. We then remove all of the labels from the set of labels associated with the just assigned servicer and remove the just completed task and its subtasks from each of the labels that contained them. We do the “Label Making Algorithm” from [18] again for the just assigned servicer. We again find the label with the earliest “cost to here” value and continue through the steps until all tasks have been marked complete or until the servicers cannot reach any more tasks within the time horizon. The details and pseudocode for the algorithm can be seen in Algorithm 8.

Algorithm 8 Fast Task Assignment (Lines marked with † indicate new or changed steps in the algorithm.)

```
1: Input: Network $G = (N,A)$ where nodes $i \in N$ have an associated time $t \in T$ and arcs $(i,j) \in A$ have time $\tau_{ij}$ and fuel $\phi_{ij}$ costs.
2: Input: Set of servicers $D$ with their associated starting location, current time, current location, and fuel capacity
3: † Input: Set of subtask sets $\Omega_{vk}$, for $k \in B_v$ and $v \in B$
4: for each servicer $\in D$ do
5:     Label Making Algorithm from [18]
6: end for
7: † Examine all node labels which have tasks; choose the label with the smallest cost_to_here, call this the chosen label.
8: † Let chosen_task be the task number in the chosen label and let chosen_subtask be the subtask number in the chosen label.
9: while Any task is incomplete or while any servicer time is less than time horizon do
10:     †Obtain the set of arcs and times for the chosen label: $\Omega_{chosen\_task,chosen\_subtask}$
11:     †Advance the servicer position to the final node in $\Omega_{chosen\_task,chosen\_subtask}$
12:     †Advance the servicer time to the time corresponding to the final node in $\Omega_{chosen\_task,chosen\_subtask}$
13:     Mark the task in the chosen label complete
14:     Do Label Making Algorithm from [18] for the just assigned servicer.
15:     if Node labels with tasks exist then
16:         Examine all node labels which have tasks; choose the label with the smallest cost_to_here, call this the chosen label.
17:     else
18:         Advance all servicer current time values to end of the time horizon.
19:     end if
20: end while
21: Output: Completed tasks and the associated labels indicating when and by which servicer
```

4.4.2 Heuristic II: Task Plus One Algorithm

The next constructive heuristic we present is called the “Task Plus One Algorithm” because it examines the next task, plus one additional task into the future. This algorithm is similar to the “Greedy Clustering Task Assignment” algorithm in [18], but because it incorporates the task processing times, there are some differences. The Task Plus One algorithm has an additional preprocessing step much like the “Greedy Clustering Task Assignment” in [18] in which we create a lookup table which contains the minimum time cost from the end of each subtask to every other incomplete task that can be reached within the time horizon. Creating the lookup table can take some time, especially for problems with a longer time horizon, but the pre-processing increases the efficiency of finding close pairs of tasks. The steps to create a “Task Plus One” lookup table are very similar to the well known steps of Dijkstra’s Algorithm [1], but we present the pseudocode to illustrate how the task processing times are incorporated. The pseudocode is shown in Algorithm 9. We use this lookup table in the Task Plus One Algorithm when examining labels which we introduce next.

Algorithm 9 Steps to create the lookup table for the “Task Plus One” Algorithm

```
1: Input: Network $G = (N, A)$ with nodes $i \in N$ and a time $t \in T$ and arcs $(i, j) \in A$ have time
   $\tau_{ij}$ and fuel $\phi_{ij}$ costs.
2: Input: Set of tasks $B$, with $v \in B$ and the set of associated subtasks for each tasks $B_v$ with
   $k \in B_v$ and the associated sets for each subtask $\Omega_{vk}$ for $v \in B$, for $k \in B_v$.
3: for $v \in B$ do
4:   for $k \in B_v$ do
5:     Let $tasks\_todo = B \setminus v$
6:     for $i \in N$ do
7:       Create a label for each $i$ with $Permanent = 0$ and $cost\_to\_here = \infty$
8:     end for
9:     Obtain the $final\_node\_vk$ and $final\_time\_vk$ from the set $\Omega_{vk}$
10:    For $final\_node\_vk$, set $cost\_to\_here = final\_time\_vk$
11:    Set $curr\_node$ = the node with $Permanent = 0$ and the minimum $cost\_to\_here$
12:    while $tasks\_todo \neq \emptyset$ AND $\exists$ a $Permanent = 0$ node do
13:      for All $j : (curr\_node, j) \in A$ do
14:        Let $next\_cost = curr\_node_{cost\_to\_here} + \tau_{curr\_node j}$
15:        if $j_{cost\_to\_here} > next\_cost$ then
16:          $j_{cost\_to\_here} = next\_cost$
17:        end if
18:      end for
19:      Set $curr\_node$ with $Permanent = 1$
20:      if $\exists$ a subtask at $curr\_node$ for a task which is $\in tasks\_todo$ then
21:        Remove the task from $tasks\_todo$ and retain this subtask $k$, task $v$, and the associ-
           ated $final\_node\_vk$ and $final\_time\_vk$ from the set $\Omega_{vk}$
22:      end if
23:      Set $curr\_node$ = to be the node with $Permanent = 0$ and the smallest $cost\_to\_here$
24:    end while
25:  end for
26: end for
27: Output: “Lookup Table”: A list containing $\forall k \in B_v$ $\forall v \in B$ the time cost of the path from $k$ to
    $v$ if it exists.
```

Like the “Fast Task Assignment” algorithm, the “Task Plus One” algorithm begins with the network, the servicers, the tasks, and the associated subtasks and subtask sets for the tasks. Labels are created for each of the servicers as described in “Label Making Algorithm” from [18]. We examine the labels for all servicers and focus on those which have incomplete tasks indicated in the label. For each of the labels with an incomplete task, we use the lookup table to obtain the time cost to the next closest incomplete task and this value becomes the “Task Plus One” time for that task. From all of the labels with tasks, we choose the label with the earliest “Task Plus One” time and assign the servicer to accomplish the task in the label ensuring we move the servicer along the arcs in the associated subtask set. We then remove all of the labels from the set of labels associated with the just assigned servicer and remove the just completed task and its subtasks from each of the labels that contained them. We do the “Label Making Algorithm” from [18] again for the just assigned servicer. We continue in this manner until all the tasks have been assigned to servicer or until all of the servicers run out of time. The details and associated pseudocode can be seen in Algorithm 10.

Algorithm 10 Task Plus One Assignment (Lines marked with † indicate new or changed steps in the algorithm.)

```
1: Input: Network $G = (N, A)$ where nodes $i \in N$ have an associated time $t \in T$ and arcs $(i, j) \in A$ have
   time $\tau_{ij}$ and fuel $\phi_{ij}$ costs.
2: Input: Set of servicers $D$ with their associated starting location, current time, current location, and fuel
   capacity
3: Input: "Lookup Table" created with Algorithm 9
4: $^\dagger$ Input: Set of subtask sets $\Omega_{vk}$, for $k \in B_v$ and $v \in B$
5: **for** each servicer $\in D$ **do**
6:    **Label Making Algorithm** from [18]
7: **end for**
8: **for** Every label obtained in **Node Labeling Algorithm** from [18] which has a task **do**
9:    $^\dagger$ Check "Lookup Table" to obtain the time cost to the next closest incomplete task and this value
      becomes the *task_plus_one* time for that task
10: **end for**
11: $^\dagger$ From the list of labels, choose the label with the earliest *task_plus_one* time, call this the chosen
    label.
12: $^\dagger$ Let *chosen_task* be the task number in the chosen label and let *chosen_subtask* be the subtask num-
    ber in the chosen label.
13: **while** Any task is incomplete or while any servicer time is less than time horizon **do**
14:    $^\dagger$Obtain the set of arcs and times for the chosen label: $\Omega_{chosen\_task,chosen\_subtask}$
15:    $^\dagger$Advance the servicer position to the final node in $\Omega_{chosen\_task,chosen\_subtask}$
16:    $^\dagger$Advance the servicer time to the time corresponding to the final node in $\Omega_{chosen\_task,chosen\_subtask}$
17:    Mark the task in the chosen label complete
18:    Do **Label Making Algorithm** from [18] for the just assigned servicer.
19:    **if** Node labels with tasks exist **then**
20:       **for** Every label obtained in **Node Labeling Algorithm** from [18] which has a task **do**
21:          $^\dagger$ Check "Lookup Table" to obtain the time cost to the next closest incomplete task and this
             value becomes the *task_plus_one* time for that task
22:       **end for**
23:       $^\dagger$ From the list of labels, choose the label with the earliest *task_plus_one* time, call this the cho-
          sen label.
24:       $^\dagger$ Let *chosen_task* be the task number in the chosen label and let *chosen_subtask* be the subtask
          number in the chosen label.
25:    **else**
26:       Advance all servicer current time values to end of the time horizon.
27:    **end if**
28: **end while**
29: Output: Completed tasks and the associated labels indicating when and by which servicer
```

4.5 Computational Results

In the previous section, we presented two algorithms for solving the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times

(MORSO-KTT) problem. In this section we performed extensive computation testing using the mixed integer linear program model presented in Section 4.3 first solving with CPLEX and then solving the same instances with both the “Fast Task” algorithm and the “Task Plus One” algorithm. The computational tests used in this work come from a case study first presented in [19]. The case study uses a network with 162 nodes spread across three different orbital altitudes. The arcs between the nodes were constructed based on considerations from orbital mechanics resulting in 1624 arcs. The case study in [19] uses a test design based on factors similar to the factors in Table 4.5.1. The factor and level combinations in Table 4.5.1 result in 1860 test cases. We ran all cases three times once with CPLEX and then once with each algorithm.

The CPLEX runs were accomplished in the high performance computing center on public standard nodes with two Xeon Gold 6130 processors, 32 cores, and 192 GB of memory. For the CPLEX instances with a 24 hour time horizon we set a solving time limit of 12 hours for the instances with a 24 hour time horizon we set a solving time limit of 24 hours. All heuristic instances were run on a single Apple Mac Mini with an M1 processor and 16 GB of RAM.

In the sections that follow we will first present a comparison of the optimality gaps achieved with each algorithm as compared to CPLEX. Each instance has a set of 8 or 16 tasks for the servicers to complete therefore the maximum value of the objective function is 8 or 16, depending on the number of tasks available. For this reason, we also present a comparison of the objective values achieved by each algorithm as compared to CPLEX. For each comparison we will also present a table comparing the numerical results for the optimality gaps and the solving times. We begin with the ‘Fast Task” algorithm.

Table 4.5.1: Factors and levels for the test cases. The combination of these factors and levels resulted in 1860 test cases which were solved with CPLEX and each of the algorithms.

Factor	**Levels**
Number of Robot Servicers	1, 2, 3, 4, 5
Number of Refuelling Depots	1, 2, 3, 4, 5
Refuelling Depot Starting Locations (node numbers)	12, 32, 80, 125, 150
Robot Servicer Fuel Capacity	10, 15, 25
Number of Tasks	8, 16
Time Horizon	24, 48

4.5.1 Fast Task Algorithm

In this section we present the results for the Fast Task Algorithm (FTA) as compared to CPLEX for solving MORSO-KTT. First we present the results for the cases with a 24 hour time horizon. Then we present the results for cases with a 48 hour time horizon.

In Table 4.5.2 we group the responses by the Number of Factors, Number of Servicers, and the robot servicer Starting fuel, which is also the robot servicer maximum fuel capacity. These three factors were the most influential on the difference in the optimality gap and solving time when comparing CPLEX with FTA. Each of these combinations has 31 cases because the number and starting locations of the refuelling depots, and the starting location of the robot servicers were not statistically significant.

For the cases with a 24 hour time horizon, 3 of 930 cases failed to achieve any solution in CPLEX in the 12 hour solving time. Within the 12 hour solving time, 786/930 cases reached integer optimality and 141/930 cases presented the best solution when the time limit was reached. The cases that failed to reach any solution are marked with an exponent on the Number of Runs column. When examining the two furthest right columns, we can see that FTA provided the biggest combined improvement over CPLEX for the resource constrained problem with 8 tasks, 5 services and a fuel capacity of 15. Next we take a deeper look at the optimality gap and an alternative to evaluating the solution qualities.

Convention when comparing solution methodologies is to compare the demonstrated optimality gaps achieved within the same solving time period. We show a histogram to compare the optimality gaps achieved in 12 hours in Figure 4.5.1. For the runs with 16 tasks, a gap of 6.25% indicates that the FTA solution was off by one task from the CPLEX Best Upper Bound (BUB). Likewise for the 8 task runs a gap of 12.5% indicates that the FTA was off from the CPLEX BUB by 1 task. Because the objective values are limited to 8 or 16, we also present a comparison of the objective values in Figure 4.5.2. In this figure we see that FTA was within CPLEX in nearly 78% of cases.

In addition to the objective values, we also considered the time savings. For the 930 runs considered the solving time for CPLEX ranged from 12 seconds to 12 hours. The FTA solving times ranged from 5 seconds to 35 minutes. The total solving time for CPLEX was just over 2110 hours while the total solving time for FTA was just under 39 hours which is a 98% reduction in solving time.

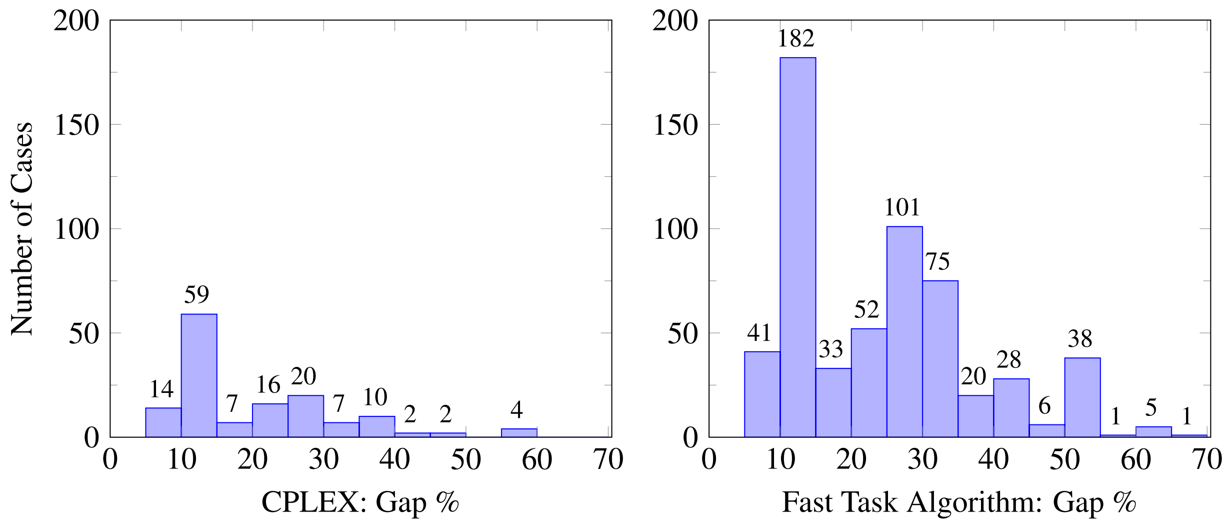

Figure 4.5.1: Distributions of non-zero optimality gap values for the Fast Task Algorithm (right) and CPLEX (left) with a 24 hour time horizon. In the Fast Task Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound.

Next we examine the 48 hour time horizon and compare the performance of CPLEX and the FTA. As we did with the 24 hour time horizon we have a table comparing the results in Table 4.5.3. We can see that the FTA performed better than CPLEX in the resource constrained cases, which were generally more difficult for CPLEX to solve. As in the 24 hour time horizon cases, the one task optimality gap equivalent in the 16 task cases is 6.25% and in 8 task cases is 12.5%. We can see that in the cases with 16 tasks, 5 servicers and a starting fuel of 10, that the FTA performed on average 5.5 tasks better than the CPLEX BUB and nearly half of the 31 runs failed to get any solution within the 24 hour solving time. The distributions of the optimality gaps can be seen in Figure 4.5.3 and the differences in the objective values (number of tasks completed) can be seen in Figure 4.5.4. For the cases with the 48 hour time horizon, the FTA matched or exceeded the CPLEX performance in 53% of the runs and was within 1 task of the CPLEX per-

Table 4.5.2: Results for the Fast Task Algorithm compared with CPLEX for the 24 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. Three runs failed to achieve any solution within the 12 hour solving time for CPLEX. 786/930 cases reached integer optimality and 141/930 cases hit the solving time limit for CPLEX.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	0.00	824	34.41	12	34.41	-811
		15	31	0.00	200	19.89	53	19.89	-146
		25	31	0.00	19	12.90	32	12.90	13
	2	10	31^1	10.96	20363	35.64	56	24.68	-20307
		15	31	0.65	1734	17.40	117	16.75	-1617
		25	31	0.00	36	7.84	118	7.84	82
	3	10	31	9.67	24161	31.44	44	21.77	-24117
		15	31	1.33	9635	11.41	65	10.08	-9569
		25	31	0.00	84	12.10	143	12.10	59
	4	10	31^1	13.79	34563	23.14	172	9.34	-34391
		15	31	6.85	16855	6.85	119	0.00	-16735
		25	31	0.00	137	6.85	190	6.85	52
	5	10	31	10.96	24166	17.07	154	6.11	-24012
		15	31	5.65	12755	0.40	321	-5.24	-12434
		25	31	0.00	177	0.40	241	0.40	64
16	1	10	31	0.00	208	32.26	22	32.26	-186
		15	31	0.00	195	9.68	72	9.68	-122
		25	31	0.00	87	0.00	78	0.00	-8
	2	10	31	4.76	9650	31.49	109	26.73	-9541
		15	31	0.00	92	14.09	202	14.09	110
		25	31	0.00	51	3.87	158	3.87	107
	3	10	31^1	9.24	10468	28.42	126	19.18	-10341
		15	31	0.00	457	10.00	226	10.00	-230
		25	31	0.00	84	2.42	344	2.42	260
	4	10	31	7.66	18167	25.05	239	17.39	-17927
		15	31	2.03	4804	9.69	154	7.66	-4649
		25	31	0.00	359	0.69	249	0.69	-110
	5	10	31	13.94	31983	26.19	123	12.25	-31860
		15	31	4.32	18570	10.78	253	6.46	-18316
		25	31	0.40	4160	4.30	328	3.90	-3831

$\#^i$ where i indicates the number of cases which failed to achieve any CPLEX solution within the 12 hour solving period.

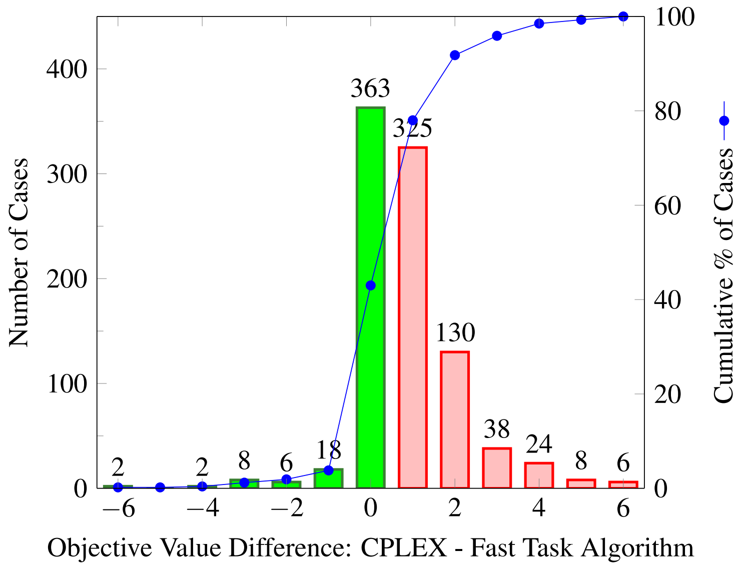

Figure 4.5.2: Fast Task Algorithm with a 24-Hour time horizon compared with CPLEX. The Fast Task Algorithm matched or outperformed CPLEX in 399/930 cases (43%) and was within 1 task of CPLEX in 724/930 (77.8%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

formance in 60% of the runs.

The FTA time savings over CPLEX is significant. For the runs with a 48 hour time horizon and a 24 hour solving time limit, the CPLEX runs took 657 days, while the FTA runs took 24 days. CPLEX failed to get any solution, other than a trivial BUB, within 24 hours in 48 of the runs; FTA provided a solution in all 930 runs. In Table 4.5.3 the runs which CPLEX failed to get any solution are marked in the Number of Runs column with the number of failed runs in the exponent. A deeper examination of the runs for which CPLEX failed to obtain a solution are shown in Table 4.5.4.

4.5.2 Task Plus One Algorithm

In this section we present the results for the Task Plus One algorithm (TPO) as compared to CPLEX for solving MORSO-KTT. First we present he results for the cases with a 24 hour time horizon and then we present the cases with a 48 hour time horizon. Table 4.5.5 shows the numeric summary of the results for the TPO with a 24 hour time horizon. Figure 4.5.5 shows the

Table 4.5.3: Results for the Fast Task Algorithm compared with CPLEX for the 48 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. 48 runs failed to achieve any solution within the 24 hour solving time for CPLEX. 417/930 cases reached integer optimality and 465/930 cases hit the solving time limit for CPLEX.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	19.16	61868	69.07	153	49.91	-61714
		15	31	6.08	28685	58.39	380	52.32	-28304
		25	31	0.00	1285	40.63	499	40.63	-785
	2	10	31	30.24	85782	61.69	801	31.45	-84981
		15	31	4.84	50068	29.44	1477	24.60	-48591
		25	31	0.00	12209	6.45	1126	6.45	-11083
	3	10	31^{2}	35.48	85530	42.74	1054	7.26	-84475
		15	31	4.84	48470	4.03	1293	-0.81	-47176
		25	31	0.00	18416	0.00	1584	0.00	-16832
	4	10	31^{7}	57.26	85223	21.37	881	-35.89	-84342
		15	31^{2}	17.74	65592	0.00	1532	-17.74	-64060
		25	31	2.82	30901	0.00	1617	-2.82	-29283
	5	10	31^{10}	76.61	85463	7.66	1069	-68.95	-84394
		15	31	11.29	63673	0.00	2307	-11.29	-61365
		25	31	7.26	43702	0.00	2234	-7.26	-41468
16	1	10	31	41.00	86427	71.69	508	30.68	-85919
		15	31	2.31	30146	47.19	1111	44.88	-29035
		25	31	0.00	2701	33.08	1302	33.08	-1398
	2	10	31	33.27	86426	64.11	1275	30.85	-85151
		15	31	15.12	86433	40.73	2367	25.60	-84065
		25	31	8.67	86414	21.17	2926	12.50	-83487
	3	10	31	37.10	86413	48.79	1955	11.69	-84457
		15	31	8.47	75368	15.73	3230	7.26	-72138
		25	31	0.00	40601	2.82	3667	2.82	-36933
	4	10	31^{9}	75.40	86407	34.07	1875	-41.33	-84531
		15	31	43.95	83220	3.63	4039	-40.32	-79180
		25	31	25.81	68844	0.40	5211	-25.40	-63632
	5	10	31^{15}	83.27	86406	17.14	2458	-66.13	-83947
		15	31^{3}	55.24	83205	0.81	4723	-54.44	-78482
		25	31	39.92	76658	0.00	4422	-39.92	-72236

$\#^{i}$ where *i* indicates the number of cases which failed to achieve any CPLEX solution within the 24 hour solving period.

Table 4.5.4: Runs for which CPLEX did not find a solution in 24 hours. The optimality gap is that of the Fast Task Algorithm as compared to the reported CPLEX Best Upper Bound.

Number of Tasks	Number of Servicers	Starting Fuel	Number of CPLEX Failed Runs	Mean Gap (%)
8	3	10	2	37.5
	4	10	7	23.2
		15	2	0.0
	5	10	10	7.5
16	4	10	9	31.9
	5	10	15	16.3
		15	3	0.0

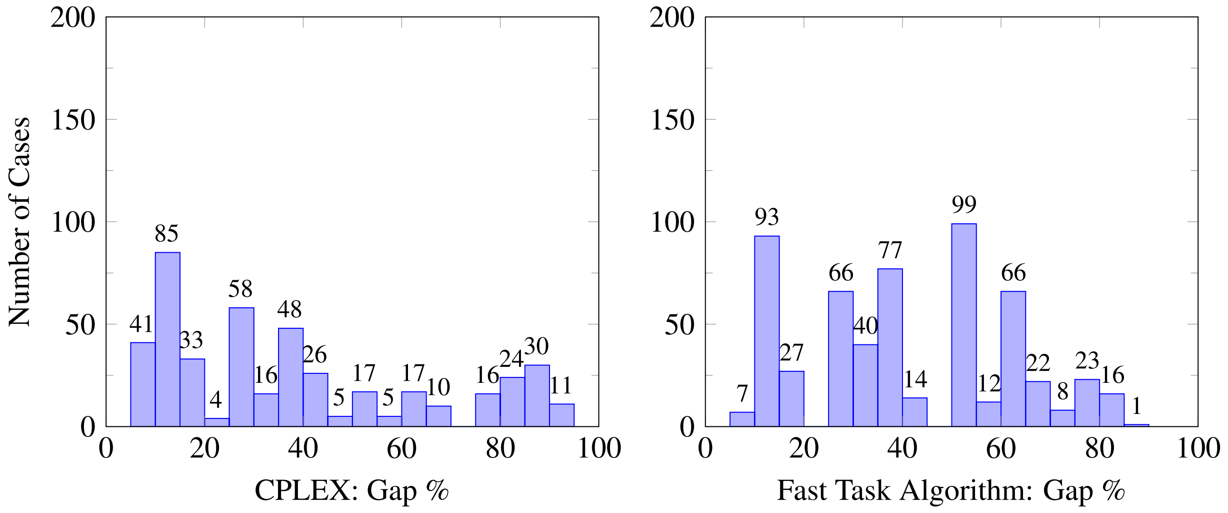

Figure 4.5.3: Distributions of non-zero optimality gap values for the Fast Task Algorithm (right) and CPLEX (left) with a 48 hour time horizon. In the Fast Task Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound.

distribution of the optimality gaps and Figure 4.5.6 shows the differences in the objective values. For the 25 hour time horizon, TPO exceeded or matched the CPLEX performance 58% of the time and was within one task of nearly CPLEX 80% of the time.

This methodology requires the use of a lookup table. The time to complete the lookup table for the 24 hour time horizon was approximately 8 hours. If we add the lookup table creation to the total processing time used to solve the 930 cases for the 24 hour time horizon, the total time for TPO is approximately 53 hours. This is a 97% time savings from the CPLEX total solving

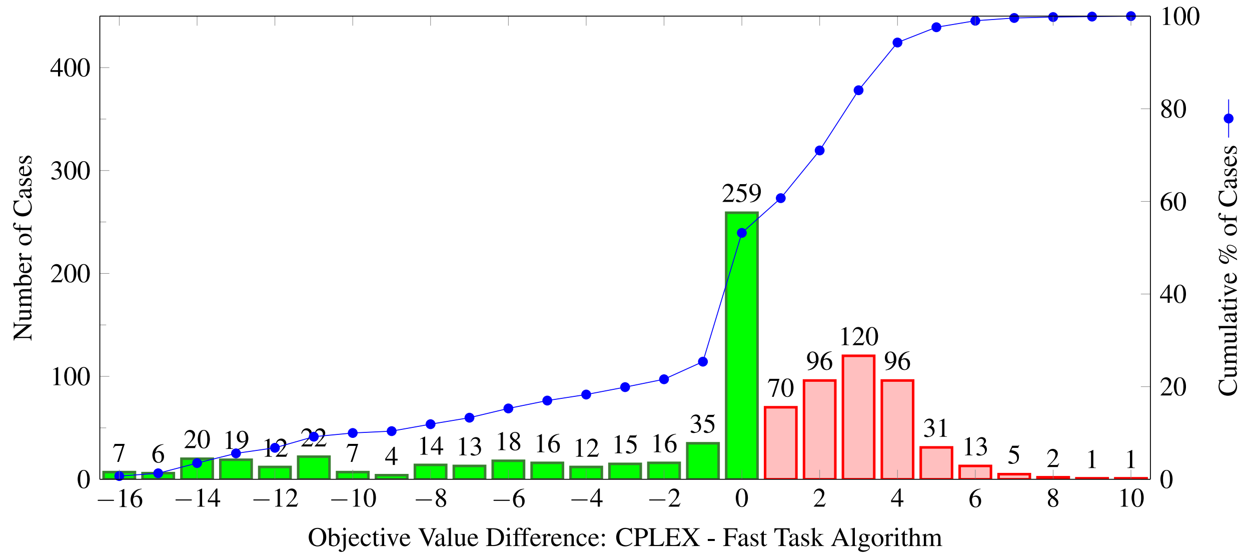

Figure 4.5.4: Fast Task Algorithm with a 48-Hour time horizon compared with CPLEX. The Fast Task Algorithm matched or outperformed CPLEX in 495/930 cases (53%) and was within 1 task of CPLEX in 565/930 (60.7%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

time of 2110 hours.

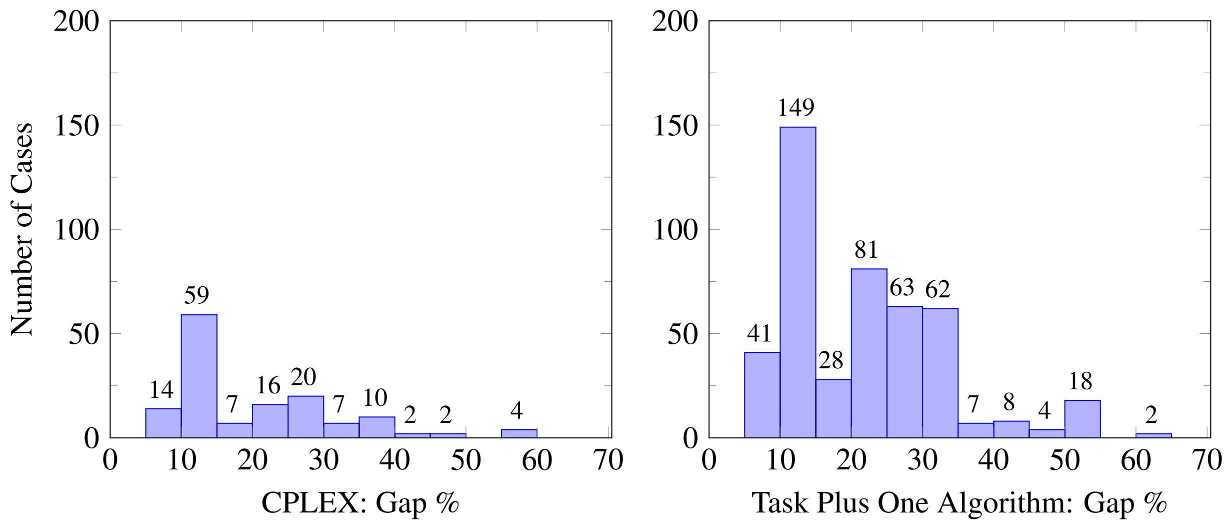

Figure 4.5.5: Distributions of non-zero optimality gap values for the Task Plus One Algorithm (right) and CPLEX (left) with a 24 hour time horizon. In the Task Plus One Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound.

Next we present the results for the cases with a 48 hour time horizon. Table 4.5.7 shows the numeric summary of the results for the TPO with a 48 hour time horizon as compared with

Table 4.5.5: Results for the Task Plus One Algorithm compared with CPLEX for the 24 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. Three runs failed to achieve any solution within the 12 hour solving time for CPLEX. 786/930 cases reached integer optimality and 141/930 cases hit the solving time limit for CPLEX.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	0.00	824	13.98	29	13.98	-795
		15	31	0.00	200	2.42	34	2.42	-165
		25	31	0.00	19	0.00	58	0.00	39
	2	10	31^1	10.96	20363	17.30	64	6.34	-20299
		15	31	0.65	1734	5.56	80	4.92	-1654
		25	31	0.00	36	0.00	111	0.00	75
	3	10	31	9.67	24161	16.09	96	6.42	-24065
		15	31	1.33	9635	2.54	116	1.21	-9518
		25	31	0.00	84	3.23	156	3.23	71
	4	10	31^1	13.79	34563	15.00	122	1.21	-34441
		15	31	6.85	16855	1.21	198	-5.65	-16656
		25	31	0.00	137	1.21	206	1.21	68
	5	10	31	10.96	24166	12.57	156	1.61	-24010
		15	31	5.65	12755	0.40	247	-5.24	-12508
		25	31	0.00	177	0.40	270	0.40	93
16	1	10	31	0.00	208	32.26	40	32.26	-168
		15	31	0.00	195	27.74	86	27.74	-109
		25	31	0.00	87	11.22	107	11.22	20
	2	10	31	4.76	9650	29.97	87	25.21	-9562
		15	31	0.00	92	18.92	173	18.92	80
		25	31	0.00	51	8.39	205	8.39	153
	3	10	31^1	9.24	10468	21.00	134	11.76	-10333
		15	31	0.00	457	3.23	230	3.23	-226
		25	31	0.00	84	0.00	312	0.00	228
	4	10	31	7.66	18167	20.15	177	12.49	-17990
		15	31	2.03	4804	3.84	272	1.81	-4531
		25	31	0.00	359	0.00	394	0.00	34
	5	10	31	13.94	31983	26.19	210	12.25	-31773
		15	31	4.32	18570	10.30	401	5.98	-18169
		25	31	0.40	4160	10.95	461	10.55	-3699

$\#^i$ where *i* indicates the number of cases which failed to achieve any CPLEX solution within the 12 hour solving period.

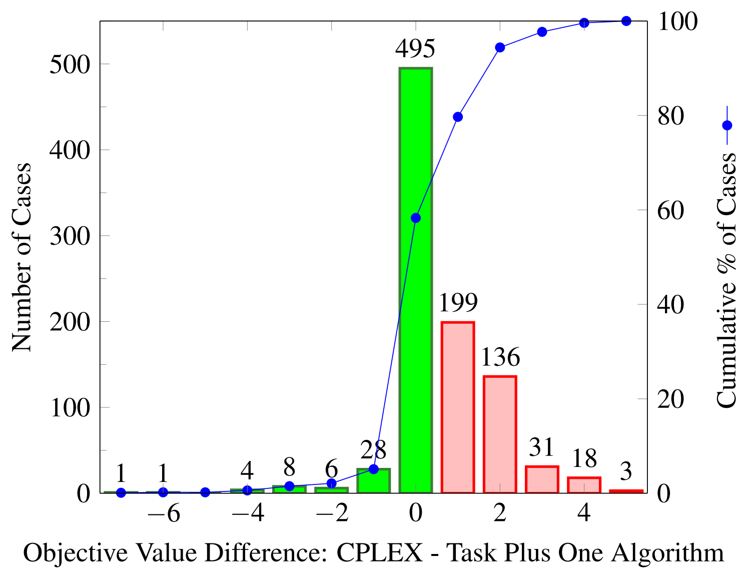

Figure 4.5.6: Task Plus One Algorithm with a 24-Hour time horizon compared with CPLEX. The Task Plus One Algorithm matched or outperformed CPLEX in 543/930 cases (58.3%) and was within 1 task of CPLEX in 742/930 (79.7%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

CPLEX. Figure 4.5.7 shows the distribution of the optimality gaps and Figure 4.5.8 shows the differences in the objective values. For the 48 hour time horizon, TPO exceeded or matched the CPLEX performance 59% of the time and was within one task of nearly CPLEX 69.2% of the time. The time to complete the lookup table for the 48 hour time horizon was approximately 22 hours. If we add the lookup table creation to the total processing time used to solve the 930 cases for the 48 hour time horizon, the total time for TPO is approximately $561+22=583$ hours. This is a 96% time savings from the CPLEX total solving time of 15779 hours. Table 4.5.6 shows the breakdown of the 48 hour time horizon runs for which CPLEX did not obtain a solution. The optimlaity gap for the TPO uses the CPLEX reported BUB.

We have presented the computational comparison results for each of two constructive heuristics as compared to CPLEX for solving the MORSO-KTT problem. Both heuristics present an accurate and timely alternative to CPLEX, especially in the cases that are resource constrained and therefore more difficult to solve. In this brief section we want to provide some results when we compared all three methods.

Table 4.5.6: Runs for which CPLEX did not find a solution in 24 hours. The optimality gap is that of the Task Plus One Algorithm as compared to the reported CPLEX Best Upper Bound.

Number of Tasks	Number of Servicers	Starting Fuel	Number of CPLEX Failed Runs	Mean Gap (%)
8	3	10	2	25.0
	4	10	7	1.8
		15	2	0.0
	5	10	10	0.0
16	4	10	9	30.6
	5	10	15	15.0
		15	3	0.0

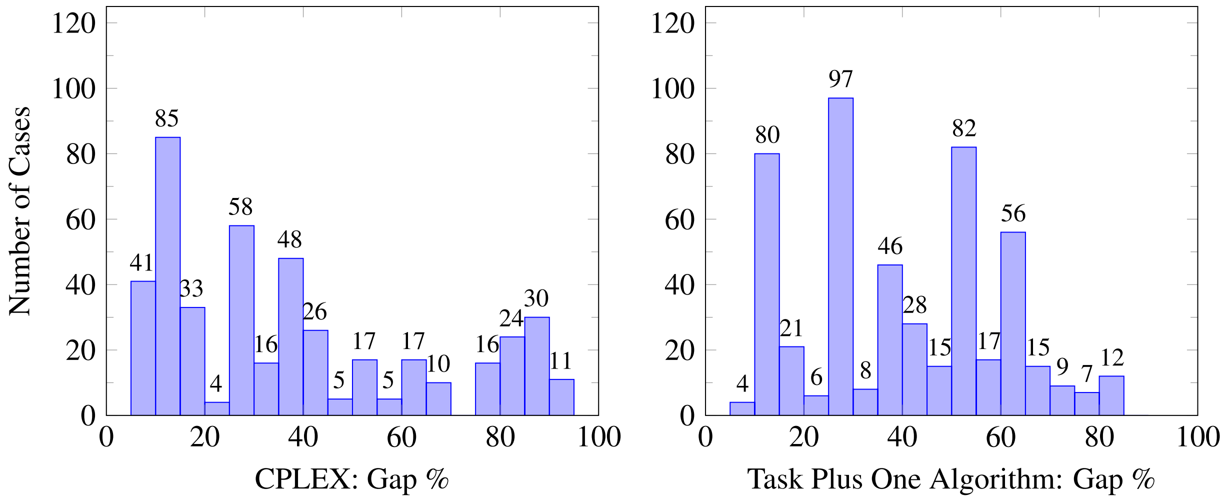

Figure 4.5.7: Distributions of non-zero optimality gap values for the Task Plus One Algorithm (right) and CPLEX (left) with a 48 hour time horizon. In the Task Plus One Algorithm, a gap of 6.25% or 12.5% indicates that the heuristic solution was within 1 task of the CPLEX reported best upper bound.

First we compared the solving times of all three methods. A summary of the solving time data is shown in Table 4.5.8. We next compared the solving times for all three methods for all 1860 runs and FTA had the fastest solving time most often. Figure 4.5.9 shows the breakdown of the fastest methods. We also compared the solutions by method and in many cases there was a tie in methods for the best objective value. In the 1860 cases or runs that we examined, 518 resulted in a three-way tie and 541 resulted in a tie for the best solution. The TPO algorithm tied with CPLEX for the best solution most often. Figure 4.5.10 shows the counts for the number of times

Table 4.5.7: Results for the Task Plus One Algorithm compared with CPLEX for the 48 hour time horizon. In the last two columns, negative values indicate that on average, the heuristic outperformed CPLEX for those instances. 48 runs failed to achieve any solution within the 24 hour solving time for CPLEX. 417/930 cases reached integer optimality and 465/930 cases hit the solving time limit for CPLEX.

Factors				CPLEX 12.10		Heuristic		Comparison CPLEX - Heuristic	
Number of Tasks	Number of Servicers	Starting Fuel	Number of Runs	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)	Mean Gap (%)	Mean Solve Time (sec)
8	1	10	31	19.16	61868	59.86	320	40.71	-61547
		15	31	6.08	28685	41.40	660	35.33	-28024
		25	31	0.00	1285	21.74	551	21.74	-733
	2	10	31	30.24	85782	51.21	541	20.97	-85241
		15	31	4.84	50068	17.74	1069	12.90	-48999
		25	31	0.00	12209	0.00	1318	0.00	-10890
	3	10	31^2	35.48	85530	31.05	631	-4.44	-84898
		15	31	4.84	48470	0.81	1575	-4.03	-46894
		25	31	0.00	18416	0.00	1831	0.00	-16584
	4	10	31^7	57.26	85223	7.66	904	-49.60	-84319
		15	31^2	17.74	65592	0.00	2002	-17.74	-63589
		25	31	2.82	30901	0.00	1967	-2.82	-28933
	5	10	31^{10}	76.61	85463	1.61	1404	-75.00	-84059
		15	31	11.29	63673	0.00	2838	-11.29	-60834
		25	31	7.26	43702	0.00	2276	-7.26	-41426
16	1	10	31	41.00	86427	71.69	753	30.68	-85674
		15	31	2.31	30146	57.75	1132	55.45	-29014
		25	31	0.00	2701	42.64	1247	42.64	-1453
	2	10	31	33.27	86426	62.50	1393	29.23	-85032
		15	31	15.12	86433	42.94	2442	27.82	-83990
		25	31	8.67	86414	18.55	2793	9.88	-83620
	3	10	31	37.10	86413	43.15	1729	6.05	-84683
		15	31	8.47	75368	9.88	4020	1.41	-71348
		25	31	0.00	40601	0.00	3827	0.00	-36773
	4	10	31^9	75.40	86407	28.63	2509	-46.77	-83897
		15	31	43.95	83220	0.40	5280	-43.55	-77940
		25	31	25.81	68844	0.00	4993	-25.81	-63850
	5	10	31^{15}	83.27	86406	17.74	2007	-65.52	-84398
		15	31^3	55.24	83205	0.20	5368	-55.04	-77836
		25	31	39.92	76658	0.00	5780	-39.92	-70878

$\#^i$ where *i* indicates the number of cases which failed to achieve any CPLEX solution within the 24 hour solving period.

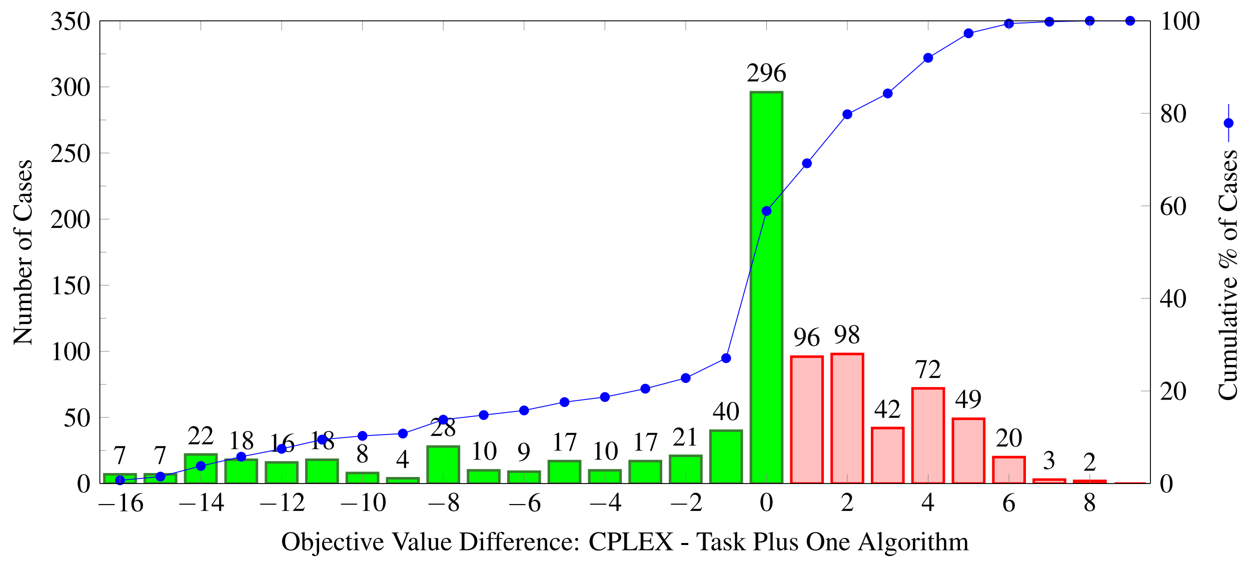

Figure 4.5.8: Task Plus One Algorithm with a 48-Hour time horizon compared with CPLEX. The Task Plus One Algorithm matched or outperformed CPLEX in 548/930 cases (59%) and was within 1 task of CPLEX in 644/930 (69.2%) cases. Positive values indicate CPLEX performed better and negative values indicate the heuristic performed better.

a solution methodology achieved the best solution value for that run; this shows more than 1860 because of the ties for best solution. We also examined a combination of speed and accuracy. There were 874 cases (of 1860) in which the method that had the fastest solve time also achieved at least a tie for the best solution. The distribution of these is shown in Figure 4.5.11.

Table 4.5.8: Table with solving times for for each methodology. The Task Plus One Algorithm times do not include the pre-processing times.

Time Horizon	Solving Method	Solving Time (hours unless indicated)			
		Minimum	Median	Maximum	Total
24 Hours	CPLEX	12.07(sec)	0.06	12.06	2110.21
	Fast Task Algorithm	5.47(sec)	0.02	0.57	38.86
	Task Plus One Algorithm	12.89(sec)	0.04	0.14	45.49
48 Hours	CPLEX	0.10	24.00	24.02	15779.38
	Fast Task Algorithm	27.25(sec)	0.34	3.10	508.80
	Task Plus One Algorithm	0.02	0.36	4.09	561.15

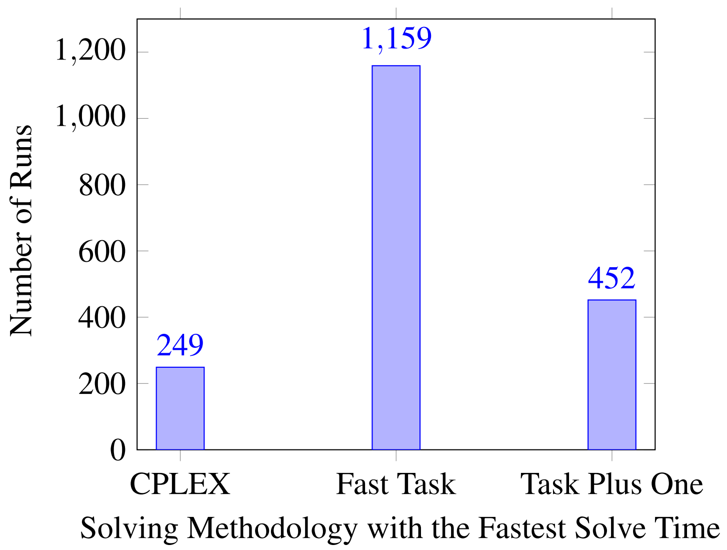

Figure 4.5.9: Histogram showing the number of times each methodology achieved the fastest solve time.

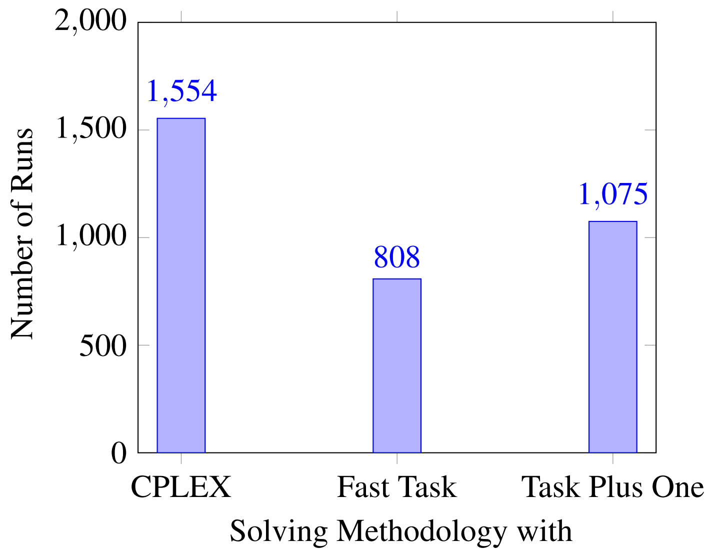

Figure 4.5.10: Histogram showing the number of times each methodology achieved the best solution. This has more than 1860 because there were ties for the best solution on several runs. 518 resulted in a three-way tie and 541 resulted in a tie for the best solution.

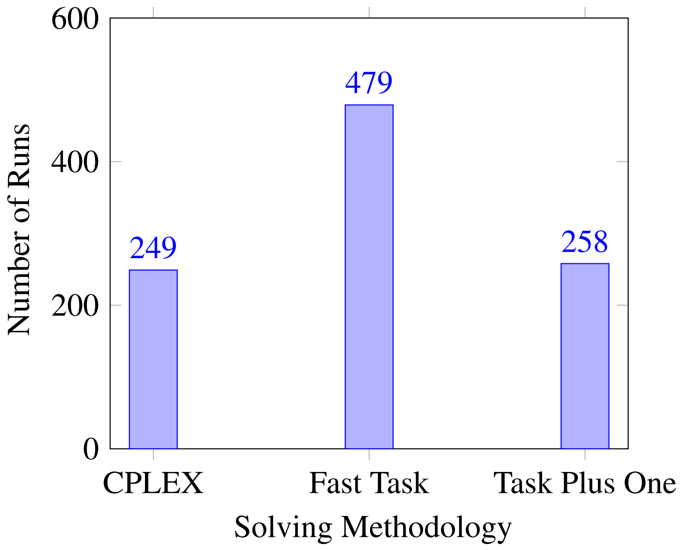

Figure 4.5.11: Cases in which the method that achieved the fastest solution time also achieved at least a tie for the best objective value solution.

4.6 Conclusions

In this work we examined the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times (MORSO-KTT) optimization problem. We presented a new mixed integer linear program model for solving this problem with an algorithm for constructing the parameters for solving. Next we presented two solving methods for MORSO-KTT and completed computational tests comparing the speed and accuracy of the methods with CPLEX. We showed that the solving heuristics result in a significant reduction in processing times while providing near optimal solutions to the problem.

For future study we first recommend extending the time horizon and increasing the number of tasks and robot servicers. The speed of the heuristics reduced the overall solving time by over 95% which would allow for a longer time horizon with a smaller step size to be examined. In this work the step size was 15 minutes, a step size of 1 to 5 minutes would decrease the error and allow for more accurate mission duration estimations.

We also recommend incorporating stochastic task processing times into the problem, model, and the heuristics. In most cases, it is unlikely that we would know exactly how long a task would take before the mission begins. If we allowed for the inherent randomness that occurs in maintenance operations, we could better estimate robot servicer numbers and mission duration.

Another direction for the work would be to have task times and service types also allowing for different robot servicers carrying and different tool types for the different types of services. The technology is growing fast and undoubtedly the need for solutions optimizing the routing and scheduling of space robots will continue to grow.

Acknowledgements

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

This research was supported by the Arkansas High Performance Computing Center which is funded through multiple National Science Foundation grants and the Arkansas Economic Development Commission.

Bibliography

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J.

[2] Alfriend, K. T., Lee, D. J., and Creamer, N. G. (2006). Optimal Servicing of Geosynchronous Satellites. *Journal of Guidance, Control, and Dynamics*, 29(1):203–206.

[3] Bourjolly, J., Gürtuna, Ö., and Lyngvi, A. (2006). On Orbit Servicing: A Time Dependent, Moving Target Traveling Salesman Problem. *International Transactions in Operational Research*, 13(5):461–481.

[4] Corbin, B. A., Abdurezzak, A., Newell, L. P., Roesler, G. M., and Lal, B. (2020). Global Trends in On-Orbit Servicing, Assembly and Manufacturing (OSAM). `https://www.ida.org/research-and-publications/publications/all/g/gl/global-trends-in-on-orbit-servicing-assembly-and-manufacturing-osam`. IDA Document D-13161.

[5] Daneshjou, K., Mohammadi-Dehabadi, A. A., and Bakhtiari, M. (2017). Mission Planning For On-Orbit Servicing Through Multiple Servicing Satellites: A New Approach. *Advances in Space Research*, 60(6):1148–1162.

[6] Davis, J., Mayberry, J., and Penn, J. (2019). On-orbit servicing: Inspection, repair, refuel, upgrade, and assembly of satellites in space. `https://aerospace.org/sites/default/files/2019-05/Davis-Mayberry-Penn_OOS_04242019.pdf`. Last Accessed: 4 July 2022.

[7] Du, B., Zhao, Y., Dutta, A., Yu, J., and Chen, X. (2015). Optimal Scheduling of Multispacecraft Refueling Based on Cooperative Maneuver. *Advances in Space Research*, 55(12):2808–2819.

[8] Dutta, A. and Tsiotras, P. (2007). A Greedy Random Adaptive Search Procedure for Optimal Scheduling of P2P Satellite Refueling. In *AAS/AIAA Space Flight Mechanics Meeting*, pages 07–150.

[9] Dutta, A. and Tsiotras, P. (2010). Network Flow Formulation for Cooperative Peer-to-Peer Refueling Strategies. *Journal of Guidance, Control, and Dynamics*, 33(5):1539–1549.

[10] Gürtuna, Ö. and Trépanier, J. (2003). On-Orbit Satellite Servicing: A Space-Based Vehicle On-Orbit Servicing Routing Problem. In *Operations Research in Space and Air*, pages 123–141. Springer.

[11] Kannon, T. E., Nurre, S. G., Lunday, B. J., and Hill, R. R. (2015). The Aircraft Routing Problem with Refueling. *Optimization Letters*, 9(8):1609–1624.

[12] Li, W. J., Cheng, D. Y., Liu, X. G., Wang, Y. B., Shi, W. H., Tang, Z. X., Gao, F., Zeng, F. M., Chai, H. Y., Luo, W.-B., Cong, Q., and Gao, Z. L. (2019). On-orbit Service (OOS) of Spacecraft: A Review of Engineering Developments. *Progress in Aerospace Sciences*, 108:32–120.

[13] NASA (2021a). About - Hubble Servicing Missions. `https://www.nasa.gov/mission_pages/hubble/servicing/index.html`. Last Accessed: July 4, 2022.

[14] NASA (2021b). On-orbit Servicing, Assembly, and Manufacturing 1 (OSAM-1). `https://nexis.gsfc.nasa.gov/osam-1.html`. Last Accessed: July 4, 2022.

[15] qian Chen, X. and Yu, J. (2017). Optimal Mission Planning of GEO On-Orbit Refueling in Mixed Strategy. *Acta Astronautica*, 133:63–72.

[16] Sarton du Jonchay, T., Chen, H., Gunasekara, O., and Ho, K. (2020). Rolling Horizon Optimization Framework For the Scheduling of On-Orbit Servicing Operations Under Servicing Demand Uncertainties. In *ASCEND 2020*, page 4131. American Institute of Aeronautics and Astronautics, Inc.

[17] Shen, H. and Tsiotras, P. (2005). Peer-to-Peer Refueling for Circular Satellite Constellations. *Journal of Guidance, Control, and Dynamics*, 28(6):1220–1230.

[18] Sorenson, S. E. and Nurre Pinkley, S. G. (2022a). Complexity and solution methods for the multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots. Manuscript submitted for publication.

[19] Sorenson, S. E. and Nurre Pinkley, S. G. (2022b). Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots. Manuscript submitted for publication.

[20] The Space Report Online (2022). On-orbit Refueling, Servicing Extends Life for Old Satellites, Promises Longer Mission Capabilities with Network of Stations. `https://www.thespacereport.org/uncategorized/on-orbit-refueling-servicing-extends-life-for-old-satellites-promises-longer-mission-capabilities-with-network-of-stations/`. Last Accessed: July 4, 2022.

[21] Yu, J., Yu, Y. G., Huang, J. T., Chen, X. Q., and Liu, H. Y. (2017). Optimal Scheduling of GEO On-Orbit Refuelling with Uncertain Object Satellites. *MATEC Web of Conferences*, 114:3001.

[22] Zhang, J., Parks, G. T., Luo, Y. Z., and Tang, G. J. (2014). Multispacecraft Refueling Optimization Considering the J2 Perturbation and Window Constraints. *Journal of Guidance, Control, and Dynamics*, 37(1):111–122.

5. Conclusions and Future Work

In this dissertation we thoroughly study the routing and scheduling of space robots to complete tasks on satellites in space. First, in Chapter 2, we introduce the Multi-Orbit Routing and Scheduling of Refuellable On-Orbit Servicing Space Robots (MORSO) problem and present a novel Mixed Integer Linear Program (MILP) formulation seeking to maximize the weighted number of tasks completed within a given time horizon. We created new algorithms which enable the creation of data used when solving the problem. Beginning with the nodes, which are fixed locations in space, we present an algorithm to create the arcs to connect the nodes using orbital mechanics to calculate the arc time and fuel costs. We created a novel way to represent the movement of the tasks and refuelling depots on the network and present the two algorithms to construct the data and parameter inputs which are also used when solving the problem.

To validate MORSO, we used our algorithms to create the data used in the solving the problem. We created multiple case studies based on satellites which are currently operating in Low Earth Orbit (LEO), Mid Earth Orbit (MEO), and Geosynchronous Earth Orbit (GEO) and used the MORSO MILP to solve them. From the case study solutions, we provided sample insights for decision makers on the number and configuration of robot servicers needed for a set of tasks. All results were based on our network but similar insights could be gained with a larger network with more accurate arc costs which could include not only the time and fuel for tasks but the expected rendezvous and negotiation time and fuel costs.

During the computational experiments for the case studies in Chapter 2, we observed that CPLEX often ran out of memory, or was unable to find a solution within the allotted solving time likely due to the large number of decision variables and constraints. This motivated the hypothesis that the MORSO problem is hard to solve. In 3, we used the known $NP-$Complete $1 \mid d_j = d \mid w_jU_j$: single machine scheduling problem that seeks to minimize the weighted number of late tasks when all tasks have the same due date [3] to prove our hypothesis that MORSO is indeed $NP-$Hard. With the complexity of MORSO established, we turned our focus to developing constructive heuristic solution methods.

In Chapter 3 while working towards the development of a constructive heuristic, we developed a node labelling process which expands upon the type of node labelling used in Dijkstra's algorithm [1] and in an aircraft routing with refuelling problem [2]. We then used the labels to develop a greedy algorithm for assigning the robot servicers to tasks. We also investigated and successfully implemented a second constructive heuristic, Greedy Clustering, for solving based on the clustering of pairs of tasks. The idea was to find the next closest task to the current task under consideration. Both of these heuristics performed well in terms of accuracy and speed. In terms of solution quality, the heuristics performed better, matched the performance, or completed one less task than the CPLEX solution in more than 70% of the 1,860 cases we examined. Furthermore, in terms of speed, the heuristics found these high-quality solutions in a matter of minutes or seconds compared to hours needed by CPLEX.

Although the assumption of instantaneous task completion is valid for OOS tasks such as inspection, it is not valid for tasks that require the servicer to physically touch and connect with the satellite needing service. In contrast to the MORSO in Chapter 2 in which the tasks had instantaneous processing time, in our next problem, the tasks have known task processing times. In Chapter 4, we examined the Multi-Orbit Routing and Scheduling of Refuellable Space Robots for On-Orbit Servicing with Known Task Times (MORSO-KTT) optimization problem to address the OOS problem for tasks that are not instantaneous in nature. We presented a new mixed integer linear program model for solving this problem and include an algorithm for constructing the data and parameters representing the tasks, subtasks, and subtask sets. The new MILP uses subtasks with location time pairs to represent the start of tasks along with a set of arcs which are consecutive in the rotational direction of the orbit to ensure that a robot servicer stays on task until completion.

We augmented the case study created in Chapter 2 to add processing times and then used CPLEX to solve 1,860 instances of the MORSO-KTT MILP. We modified the constructive heuristics created in Chapter 3 to account for the subtask sets and task duration to create a new greedy heuristic called, Fast Task, and a new clustering algorithm called, Task Plus One. The heuristics performed well when solving the runs in the case study with a significant reduction in solving time.

Although the technology for the MORSO and MORSO-KTT does not yet exist, the field is growing rapidly and there are many areas for future work. For future study, we first recommend extending the time horizon and increasing the number of tasks and robot servicers. The speed of the heuristics reduced the overall solving time by over 90% which would allow for a longer time horizon with a smaller step size to be examined. In this work, the step size was 15 minutes, and a step size of 1 to 5 minutes would allow for more accurate mission duration estimations.

Because fuel is such a valuable commodity in space, further research could also include examining the MORSO-KTT as a multi-objective optimization maximizing the weighted number of tasks completed while simultaneously minimizing the number of refuelling events or minimizing the amount of fuel expended to complete the tasks. This could be explored with an expansion to the node labeling algorithm similar to [2].

We also recommend incorporating uncertain task processing times into the problem, model, and the heuristics. In most cases, it is unlikely that we would know exactly how long a task would take before the mission begins. If we allowed for the inherent randomness that occurs in maintenance operations, we could better estimate robot servicer numbers and mission duration. These are just a few of the directions this research could go. The technology is growing fast and undoubtedly the need for solutions optimizing the routing and scheduling of space robots will continue to grow.

Bibliography

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J.

[2] Kannon, T. E., Nurre, S. G., Lunday, B. J., and Hill, R. R. (2015). The Aircraft Routing Problem with Refueling. *Optimization Letters*, 9(8):1609–1624.

[3] Lenstra, J. K., Kan, A. R., and Brucker, P. (1977). Complexity of Machine Scheduling Problems. In *Annals of Discrete Mathematics*, volume 1, pages 343–362. Elsevier.