Multi-Robot Task Allocation and Scheduling for Efficient Aircraft Structure Assembly

Veniamin Tereshchuk

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Santosh Devasia, Chair

Ashis G. Banerjee, Chair

Joseph Garbini

Program Authorized to Offer Degree:
Department of Mechanical Engineering

University of Washington

Abstract

Multi-Robot Task Allocation and Scheduling for Efficient Aircraft Structure Assembly

Veniamin Tereshchuk

Co-Chairs of the Supervisory Committee:
Professor Santosh Devasia
Mechanical Engineering

Professor Ashis G. Banerjee
Mechanical Engineering and Industrial & Systems Engineering

This thesis presents the development of methods for extending multi-robot systems to aircraft structure assembly and manufacture. The use of multiple, stationary, cooperating robotic manipulators in aircraft assembly lines has the potential to increase production throughput without increasing footprint, especially benefiting high-volume production lines. As such, multi-robot systems must maximally utilize the robot assets in the production cell, which entails balanced work allocation and collision-free task scheduling to enable efficient and safe cell operation. The task allocation and scheduling problem, often modeled as a multiple traveling salesman problem is an NP-hard problem. However, since task allocation in aircraft structure assembly is a large scale problem, often having as many as on the order of 10^3 total tasks, current methods for allocation and scheduling are computationally prohibitive, and excessive robot idling due to long computation of task schedules also has the potential to decrease the efficiency of the production cell. The scheduling is further complicated by the dynamic nature of the robots' capabilities that comes in the forms of periodic robot failures that require a robot to be pulled out of operation for maintenance, and tool changes for robots that are required for the robots to be able to service a variety of tasks. This poses some unique challenges in terms of (i) enabling fast, balanced and collision-free scheduling

of the robots that properly responds to robot failures, (ii) determining the proper placement of the robot bases in the robot cell to ensure maximal workload sharing potential, and (iii) ensuring that tool changes are handled such that the time cost of changing tools is minimized. In addressing these challenges, the contributions of this thesis are to (i) employ a partition-based scheduler and market-based schedule optimizer in a two stage scheduling framework to enable fast, collision-free and efficient cell operation, (ii) outline robot base placement and mobility strategies as a means to attenuate the effects of robot failures on cell efficiency, and (iii) adapt auction-based methods to minimize tool changes and develop a machine-learning framework for generating scheduling heuristics in an operationally robust manner. The methods are developed and tested on a physical multi-robot system that enables real-world application of the research.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

CENTRALIZED: computation methods that have a central computer responsible for all the computation.

DE-CENTRALIZED: computation methods that distribute some or all the computational load to the agents within the system.

MRTA: multi-robot task allocation: a class of problems that deals with allocating and/or scheduling multiple tasks among multiple robots.

NP-HARD: class of computational problems for which solutions can be generated only in non-polynomial time. This usually implies that a solution is intractable for large scale problems.

PARTITION(ING): commonly referred to as graph partitioning in the computing community, is a method of clustering vertices on a graph such that the edge cuts defined by the clusters is minimized.

PRECEDENCE CONSTRAINTS: inter-task relations that determine the order in which the tasks must be completed

TSP: the traveling salesman problem, a well studied problem wherein a salesman seeks to find the shortest route through a set of cities without visiting any city more than once.

Chapter 1

INTRODUCTION

1.1 Motivation

This work aims to increase the efficiency of automated aircraft assembly systems using multiple cooperating robotic manipulators. Aircraft structure assembly is a large scale manufacturing process with many repeated subprocesses and tasks that naturally lends itself to automation. To this end, as in the auto-manufacturing industry, robotic manipulators are increasingly being employed due to their dexterous and cross-process applicable nature. Airplane structures with irregular, flowing shapes and large surface areas, such as airplane skins (fuselages, wing skins, stabilizers, etc.) especially benefit from the use of robotic manipulators for their manufacture and assembly. Using multiple robots to work cooperatively offers a means of increasing the production speed, and hence, multi-robot systems (MRSs) have been widely studied in manufacturing applications [1–3]. This work considers the efficient scheduling of such an MRS for the assembly of large aircraft structures (e.g. wing, fuselage, etc). The system comprises multiple stationary robotic manipulators with limited but overlapping reach situated about a workpart with a large number (on the order of 10^3) of tasks. To maximize the production output, the MRS must make efficient utilization of all the robots by minimizing their idle times and travel costs [4]. Reduced idle time and travel cost have significant payoffs in increasing aircraft production rates, since the benefit extends to a large number of the structural components and manufacturing processes such as drilling, fastening, trimming, and painting. This benefit is especially felt in high volume production lines, where even small time savings have a significant contribution over time. For stationary robots, the idle time is governed by uneven workload distribution among the robots. Therefore, suitable task allocation is of primary concern, followed by appropriate

task scheduling for minimal travel cost. Often cast as a variant of the traveling salesman problem (TSP), multi-robot task allocation and scheduling is an NP-hard problem that is computation expensive and tends to be prohibitive for large scale problems. Long schedule computation times, however, also result in idling and reduce the MRS efficiency. Thus, efficient MRS operation is contingent on balanced work allocation and scheduling as well as fast schedule computation. Appropriate allocation and scheduling is important for the following reasons:

1.1.1 Shared Workspace

A key feature of MRS is the ability of the robots to share work, which is especially advantageous in dynamic environments where the robots' capabilities or the tasks to be performed can change. A shared workspace for multiple robotic manipulators, however, also introduces the potential for collisions, which can not only incur substantial repair costs, but also greatly reduce the efficiency of the whole system if it needs to be shut down for the repairs to be performed. This adds the challenge of a hard spatial proximity constraint on robot task scheduling in order to avoid collisions, and makes the robot schedules inter-dependent, further complicating the scheduling problem.

1.1.2 Failures/Breakdowns

Periodic failures, such as robot breakdown or end effector malfunctions, are inherent to industrial robot manipulators and occur stochastically during the system's operation. These are often the result of a broken tool tip, component (e.g. fastener, tape, paint) mis-feed, communication error, or an issue in the interface with the workpart, and require the failed robot to be pulled out of operation to be serviced. Failures disrupt robot schedules and must be addressed by the scheduling in order to preclude collisions or excessive robot idling.

1.1.3 Condition of Assembly

The assembly of aircraft structural parts often has variability in its tasks from part to part or airplane to airplane that arises from sub-components that are missing or, for some reason, are omitted from the part. For example, the right and left wings of an airplane may have different brackets in the wing structures, or one of the wings requires a subset of its fasteners to be oversized (changing the hole drill times and necessary tool) due to a rework. The definitions for exactly which tasks need to be completed on a given part are referred to as the condition of assembly (COA). This variability drives the need to generate task schedules for every instance of every part, and moreover, requires the schedule computation to be fast in order to not reduce the MRS efficiency if multiple instances of re-scheduling are required.

1.2 Problem Statement and Research Challenges

Problem Statement: How to schedule and coordinate multiple cooperating robots to maximize efficiency of aircraft assembly processes?

The following research challenges/questions arise from the problem statement:

R1: *How to address (break) the NP-hardness of the problem?*
As stated above, the multi-robot task allocation and scheduling problem is conceptualized as a multi-agent TSP (mTSP) and is NP-hard. This means that optimal solutions to the problem are found in non-polynomial time, and this is largely due to the combinatorial nature of the solution space. Consider that for a single agent TSP with n tasks, or cities, there exist $n!$ possible solutions. For large n, like in the application studied in this work, the problem becomes computationally intractable, and the complexity is further increased when allocating and scheduling for multiple agents. Current optimal, or even approximate near-optimal, methods, are challenged by poor scaling properties, meaning that computation time grows exponentially with the problem size. This makes them unsuited for the current application, as long wait times due to expensive computation also makes the MRS inefficient. Hence, this

research question seeks to identify the class of manufacturing problems for which fast and (near) optimal solutions are feasible, and the appropriate methods. To this end, the research question seeks to address the following challenges:

- **MRS Schedule Efficency:** Task allocation and scheduling methods have various optimization objectives that characterize the performance of a given allocation/schedule. Objectives are often application-dependent, and a manufacturing MRS assembling aircraft structures requires an optimization objective that takes into account (i) idle time, (ii) robot failures, (iii) schedule computation time, and (iv) collisions.

- **Collision-Free Scheduling:** Collision-free MRS operation is ensured by enforcing a hard spatial constraint on the robot scheduling. The challenge, however, is that the cross-schedule dependencies is the main thrust behind the NP-hardness of the allocation and scheduling problem [30]. Thus, for the specific class of manufacturing problems, addressing the NP-hardness implicitly brings with it the challenge of guaranteeing collision-free operation.

- **Failure Handling:** The scheduling approach must take into account the stochastic nature of failures. Current methods commonly propose rescheduling as a method of ensuring that failures don't reduce MRS efficiency [5]. As the failure frequency increases, though, constant re-scheduling can become challenging even for fast methods, and proper failure handling during the schedule execution is necessary to maintain MRS efficiency.

This research question is addressed in Chapter 3 of this thesis; in addressing the question, the main contribution is a partition-based nominal scheduler and a market-based schedule optimizer in the context of a two stage schedule execution framework that doesn't require immediate schedule re-computation at instances of failures. The work therein was in the *Robotics and Automation Letters (RAL)* Journal in 2019 [20].

R2: *How to optimize the design of the multi-robot system with intelligent robot base placement?*

Industrial implementation of MRS with stationary robots poses the question of where to place the robot bases, and simple solutions rely on ensuring that all tasks are within at least one robot's reach and either evenly spacing the robots (to enable maximum maneuverability) or placing them at some temporal or spatial centroid of their assigned tasks [41]. These approaches, however, don't take into account the need of robots to be able to share their workloads in order to reduce idle time that results from uneven robot failures. As the failure frequency and duration vary more, the need for overlapping robot reach for certain subsets of tasks may be more preferable than for others in order to enable a better work balance. This research question seeks to optimize the design of a manufacturing MRS by using robot placement and mobility as a means of increasing the potential for workload sharing and, consequently, increasing the MRS efficiency. This research question is addressed in Chapter 4 of this thesis.

R3: *How to minimize tool switching in the scheduling to maintain schedule efficiency?*

Aircraft structure manufacture often requires the use of various tools, and each robot in a multi-robot assembly cell is able to have its tool changed in order to perform a variety of the tasks, similar to how a computer numeric controlled (CNC) mill operates. The tool changes, however, introduce an extra time cost, as the tool needs to be installed into the end effector by an operator and then briefly tested and calibrated to ensure quality tool performance. Although tool changes are inevitable, they must be minimized in order to minimize their impact on the MRS efficiency, and hence, appropriate task ordering is of chief concern to achieve this end. Prioritizing tasks that don't necessitate a tool switch defines loose, or soft, precedence relation between the tasks that can be violated with a time penalty of a tool change. The challenge with the existing scheduling methods, however, is that they use hard precedence constraints (which cannot be violated) for efficient scheduling, which are absent in the current context. As far as we know, the literature does not deal with soft precedence

constraints, and more precedence relations must be determined in order to be able to leverage current methods. The first challenge, therefore, is determining the appropriate precedence relations to be able to leverage existing methods for generating task schedules with minimal tool switching. Additionally, the stochastic nature of the robot failure occurrences results in varying amounts of leftover work for the robots to complete, so it is desirable for the MRS to adapt, or learn, to deal with any new problem case [67]. Testing the space of all possible precedence relations for any given problem is computationally inefficient, and incurs a significant computation time as the number of tasks and task types increases. The second challenge, therefore, is selecting the appropriate scheduling technique for any given problem case in an operationally robust manner.

This research questions is addressed in Chapter 5; in addressing the question, the main contributions are an auction scheduling algorithm that employs decision-based scheduling heuristics to encode soft precedence constraints, and a data-driven framework for generating robust heuristic selection policies that is useful for a variety of multi-robot task scheduling problems. The work therein is in preparation for submission for publication in the journal of *Robotics and Computer-Integrated Manufacturing*.

Chapter 2

BACKGROUND

2.1 Multi-Robot Task Allocation

Multi-robot task allocation (MRTA) deals with various classes of problems of allocating and scheduling multiple tasks to multiple robots to maximize some reward or minimize some cost. This work considers the NP-hard problem of m robots, or agents, r_i performing n tasks, where $n >> m$. The NP-hardness of the problem arises from each robot being assigned multiple tasks where the task scheduling is constrained by inter-dependency of the task schedules of the robots. In this work, the schedule inter-dependency comes primarily in the form of a spatial proximity constraint to preclude inter-robot collision.

2.1.1 Problem Classification

In [30], Korsah et. al. proposed a taxonomy in which MRTA problems are classified according to four different parameters that deal with the robots' capabilities, the nature of the tasks, the type of assignment, and task scheduling inter-dependency.

- Robot are either single-task (ST) or multiple-task (MT) robots, signifying whether they are able to work on a single task or multiple tasks at a time.

- Task can be worked on by a single robot (SR) at a time, or multiple robots (MR).

- Task assignment is either instantaneous assignment (IA), where tasks are assigned to the robot instantaneously with no planning of future allocation, or as time-extended assignment (TA), where each robot is given several tasks that must be done according to

a specified schedule. TA problems are generally much more computationally demanding [33].

- Task schedule interdependence breaks down into four type:
  - no dependencies (ND), where the effective utility of a robot performing a given task does not depend on any other agent or task. These problems can be simplified into a linear assignment problem and solved in polynomial time [30, 31].
  - in-schedule dependencies (ID), where tasks have intra-schedule dependencies (affected only by the tasks in the given agent's schedule). These problems are often formulated as an mTSP and solved using mixed integer programs, and optimal solutions are generally found in non-polynomial time; however, since each agent's schedule is independent of all the other schedules, the problem complexity can be reduced to be more tractable, contingent upon use of appropriate heuristics [30].
  - cross-schedule dependencies (XD), where tasks have inter-schedule dependencies (affected by the tasks in the agent's schedule, as well as the schedules of the other agents). The schedule coupling in these problems increase the complexity, and require more clever heuristics [37].
  - complex dependencies (CD), where the allocation and scheduling of the tasks also depends on the decomposition of complex tasks into simpler tasks. These problems are generally all NP-hard.

  Computational complexity generally increases with complexity of the schedule dependencies, and solutions are computationally intractable for large scale problems with XD or CD dependencies [7, 30]

These various classes of problems are discussed at length in the MRTA taxonomy in [30]. Since the robots in this work have the potential for collisions, the scheduling problems falls into the the XD[ST-SR-TA] class of MRTA problems.

2.1.2 MRTA Solution Approaches

The various solution approaches are given in detail in [30] and [7]; here, MRTA solution approaches are broadly grouped by organizational paradigms (centralized vs. decentralized approaches) and solution approach (optimization vs. market-based approaches) [6, 7]. Centralized approaches rely on a central computer that allocates and schedules the task to the robots, whereas de-centralized approaches disperse some or all of the computation to the robots. In general, centralized approaches use optimization methods, since information about the tasks and agents is contained centrally, and de-centralized approaches use market-based methods, which are better suited for agent-agent communication [6]. There are, however, many centralized market-based methods and de-centralized optimization methods that have had significant success [15, 44].

2.2 Optimization Methods

Optimization-based solutions set up task assignment and scheduling as an objective function, usually a cost function to be minimized, or a reward function to be maximized, and seek to find globally optimal solutions to the function in order to find optimal allocation and scheduling. Various optimization objectives are used, depending on the application, and these are commonly spatial in nature (e.g., minimize total path of travel [18]), temporal in nature (e.g., minimize total time of completing all tasks [14]), or have to do with some objective regarding the tasks themselves (e.g., minimize the tardiness of completing the tasks [16], or maximize the number of tasks completed in a given time frame [17]). Some MRTA applications benefit from using a combination of multiple optimization objectives [14–16, 35]. The various objectives and their applications are covered in greater detail in [7] and [30]. The application in this work primarily benefits from minimizing the time to complete all tasks, or *makespan*, as the optimization objective.

As the XD[ST-SR-TA] MRTA problem is NP-hard, much of the recent work has focused on using approximation techniques to reduce computational complexity, and what follows is

a review of these techniques.

2.2.1 TSP Solution Heuristics

MRTA is often approached as a multiple traveling salesman problem (mTSP), a variant of the tradition traveling salesman problem (TSP), a well-studied computation problem that tries to find the sequence of cities a traveling salesman must visit such that the salesman's travel distance is minimized. Optimal solutions to TSPs exist most commonly in the form of branch and bound methods, where the state space of candidate solutions, represented as a tree, is pruned by using the upper and lower bounds of the optimal solution to prune the branches of the tree that have a higher cost than the computed lower bound [9, 10]. These methods are intractable for large-scale applications, as in this work, and heuristics are commonly used for approximation to reduce complexity. In [13], a Monte Carlo tree search-based heuristic was used in conjunction with the branch and bound algorithm to yield near-optimal robot task allocation for 100 tasks within an hour. The computation time for this method, however, increases as the number of tasks increases. In [11], a hybrid algorithm is proposed to solve a vehicle coordination problem formulated as an mTSP using known vehicle constraints, such as passenger capacity and vehicle battery capacity, as a heuristic to prune the solutions space. This approach yielded near optimal solutions for 50 tasks in approximately three minutes, and for 100 tasks in approximately 10 minutes. In [19], a multiple traveling robot problem was solved using mTSP methods and the closest cost (CC) and farthest addition cost (FAC) heuristic functions, and was able to allocate robot targets in polynomial time. The solution quality, however, increasingly deviated from the optimal solution as the number of targets increased, resulting in a 15% deviation at 100 targets. Heuristic methods offer a means of reducing the computational complexity of the problem to make it more tractable for various applications. However, they are often application-dependent, and heuristics that are applicable to a variety of problems (meta-heuristics) are still challenged by high computational complexity for large-scale problems. In other words, heuristics are useful, but need to be tailored to the specific application to

be of benefit. The methods in this work employ an implicit geometric heuristic to enable collision-free task scheduling.

2.2.2 Mixed Integer Linear Programming (MILP) Methods

Another common solutions method to MRTA is the use of MILPs, which set up the assignment problem as a linear program with integer program coefficients. A common formulation of a MILP is given in Equation 2.1. The program seeks to minimize the function $z = cx$, where c is the cost function and x is the decision variable. In the context of MRTA, x is the assignment decision variable and the values of c are the costs of assigning a task to a robot.

$$\begin{aligned} &\min_{x} z = cx \\ &\text{s.t.} \mathbf{A}x \geq b \\ &x \geq a \end{aligned} \tag{2.1}$$

Here, $\mathbf{A}$, b and a are the coefficients of the constraint equations. In general, MILPs are computationally demanding [37], and various works have successfully applied MILPs by combining them with other computational techniques and heuristics, like constraint programming (CP) [14] and decomposition [37, 39, 40]. A prominent example is in [14], where Gombolay et. al. employed a MILP-based method, using decomposition to solve the allocation and task sequencing problems separately. Their method used constraint programming (CP) heuristics for allocation, and a multi-agent sequencer inspired by processor scheduling [42] to near-optimally (within 15%) schedule 500 tasks to 10 agents in 20 seconds. Here, the MILP formulation relied on (CP) as a branch-and-bound technique to prune the solutions space in order to enable low computation times. The computation time still increases if the problem scale goes up or if the optimality criteria is increased. Similarly, in [12], robot task allocation was done using MILP and CP to achieve optimal allocation for 200 tasks in 100 seconds. A MILP in [38] was solved directly as an NP-hard problem, but a solution was found in reasonable time due to the small size of the problem. The trend with using MILP-based

allocation and scheduling techniques, as in the mTSP solutions, is that optimal solutions for small problems can be found with low computation times, and that clever heuristics are rquired as the problem size is increased in order to maintain fast computation [37].

2.2.3 Evolutionary Algorithms

Evolutionary or genetic methods are inspired by evolutionary biology, and rely on updating, or "evolving", solutions to achieve optimality by perturbing the solutions, or "mutating", and choosing the most optimal ones, or the "fittest", to continue to evolve. These methods have seen fair success in MRTA problems, particularly in the ND and ID classes [2, 8, 43]. In [8], a genetic algorithm was used to allocate targets to a robot team that was representative of a team of firetrucks tending to multiple fires. Their method relied on using a heuristic to bound the solution space in order to solve time-extended assignment (TA) case, and found near optimal (within 10%) for 300 tasks in 10^4 seconds. In [44], a decentralized evolutionary algorithm was used to solve a generic MRTA problem formulated as an mTSP. Here, only the instantaneous assignment (IA) case was solved, and to achieve optimality, a random mutation was periodically injected to guide the solution. For the TA case, as in the current work, a heuristic is required in order for the mutation to more quickly guide the solution towards optimality. These methods, however, are challenged in application to the current work because they suffer from getting trapped in local minima, especially in large scale problems where the solution space has more potential for local minima. Moreover, genetic algorithms still require heuristics to address the TA case, and are better suited for applications where computation time is not a limiting factor [8].

2.2.4 Partitioning Methods

Graph partitioning is a computation problem that aims to cluster graph vertices such that edge cuts are minimized, and is less commonly used in MRTA applications. Partitioning techniques are not as commonly used for MRTA as centralized optimization or market-based approaches, but they are shown to have direct benefits in solving task allocation problems

with favorable scaling properties in [27]. Moreover, partitioning has shown effectiveness in fair global task subdivision for heterogeneous robot teams in [28] by using locational optimization to subdivide a region to 50 robots in under two seconds. Partitioning methods also had favorable convergence for workload division in multi-robot exploration in [29] by use of a pair-wise optimization technique. The work in this thesis employs a similar technique for equitable workload allocation. Partitioning methods, however, are generally only used for allocation (IA), and the TA case is generally not addressed by partitioning. For ND and ID problems with the TA case, the tasks in the respective partitions can be sequenced independently of all other partitions. The work in this thesis is towards extending the partitioning methods for allocation and scheduling in a TA and XD MRTA problem. The methods are further outlined in Chapter 3.

2.3 Market/Auction-Based Methods

In a market economy, agents compute the cost of completing a task and bid on the tasks according to that cost; an auctioneer agent, either one of the agents or an external auctioneer, then allocates the task to the highest bidder. Auction based approaches are well suited for dynamic tasks and environments, and benefit from the ability to distribute the computational burden among the agents. They are especially useful when communication is limited: in [22], a consensus based algorithm was used to allocate task bundles, and was especially successful in resolving conflict that arose due to limited communication. However, market-based methods often produce less optimal solutions than the fully centralized optimization approaches [6]. These methods have enjoyed popularity in multiple mobile robot system (MMRS) task allocation, especially for unknown region exploration [5, 23–25]. Since centralized optimization methods generally produce more optimal solutions, recent efforts have focused on improving the optimality of the market-based solutions. For example, Korsah et. al. [26] seeded their market based method with pre-computed optimal schedules of static tasks, which, however, could be computationally expensive for large scale problems. Strategic pricing during the auction process was used to achieve global assignment

optimality in [32], but only considered the instantaneous assignment (IA) case, and did not consider temporospatial task constraints. Task cluster allocation was also shown to have promising results in improving the workload balance in [4]. However, it also led to increased computational complexity due to the necessity to bid on all the cluster combinations.

Therefore, similar to the centralized optimization methods, the main challenge is that improved solution quality often comes at the cost of increased computation times for large scale problems as in the current application. We address this challenge by employing a market-based method as a schedule refinement technique on a smaller scale problem defined by the leftover tasks, instead of using it to directly solve the full scheduling problem. In particular, our method exploits a schedule placement technique similar to the one used in [15], where optimal placement of the task within the schedule helped inform bids and increased solution optimality.

2.4 Optimal Robot Placement

Stationary robots must be properly placed in order for them to maximally be able to reach their own assigned tasks and share tasks with proximate robots, or to reduce their accumulated effort in completing their assigned tasks [45]. A simple way to solve this problem is to maximize the reach of robots to their assigned tasks, often by placing the robot bases according to a geometric centroid of the workspace [41,46]. More complex approaches seek to minimize a motion or energy cost, or maximize a specific property of the robot's capabilities, in order to make a robot as efficient as possible in completing its assigned tasks. In [47], a multi-objective optimization criteria that seeks to minimize actuation energy and maximize robot dexterity (in order to maximize the ability to avoid collisions in an obstacle-laden workspace) is used to determine optimal robot placement. In [48], power consumption of industrial robots is minimized by placing robot in positions that minimize the use of the most energy-consuming actuators of the robots. Optimal robot placement was used in [49] to ensure proper coverage of complex surfaces, like a car shell; additionally, the methods in this paper also find the optimal number of robots required to cover the entire surface.

Of particular interest to the work in this thesis is robot placement informed by the scope of failures. As robot failures increase (along with the variety in failures across the robots), the need for robots to be able share work becomes more and more important to preserving the overall efficiency of the MRS. As such, the reach of the robots to certain tasks that would enable maximum workload sharing is the primary goal. So far, the literature on robot placement does not address this.

2.5 Ordering Constraints

2.5.1 MRTA Ordering Constraints

Tool changes implicitly increase the cost of some candidate solutions of the task ordering problem, particularly those that require the most tool changes, since excessive tool changes incur a time overhead and negatively impact MRS efficiency, and naturally introduce ordering constraints to the scheduling problem. Ordering and temporal constraints generally increase the complexity of the scheduling problem [7,37]. Per Nunes et. al.'s MRTA-TOC (Temporal and Ordering Constraints) taxonomy in [7], these constraints take the form of time window constraints or synchronization and precedence constraints. Time windows define a start and end time for the tasks, and are generally concerned with minimizing task execution tardiness (either being too early or too late). Precedence constraints can be implicitly derived from time windows, although time windows aren't sufficient. Precedence and synchronization constraints specify a partial or total order for the tasks, defining after or before which task(s) can a given task be completed, or what tasks must be done concurrently (with either the same start time, same end time, or both) [10]. Robot end effector tool changes fall more into the category of precedence constraints, as tool changes incur no cost upon being done earlier or later in the process. There are various works that have successfully implemented precedence constraints into their scheduling methods, merely a few are highlighted here. In [14], a precedence constraint is used as a heuristic to enable a robot to prioritize a task based upon how many tasks in its immediate proximity are assigned to other robots. In [53],

hard task precedence constraints were imposed on the robot team that required the robots to explicitly communicate to each other when a task was done. The constraints were also used to alter the schedules if a task execution failed. In [54], tasks were divided into disjoint subsets with precedence constraints imposed on the subsets. The current work considers a similar formulation, where tasks requiring the same tool are grouped, however, there are no precedence constraints on the groups, rather a preference to stay within a group.

There is a distinction between *hard* and *soft* constraints, and Nunes et. al.'s taxonomy provides literature on soft and hard temporal constraints for time windows. Hard constraints are those that are not allowed to be violated, whereas soft constraints can be violated but incur some penalty. The same distinction, however, isn't given for precedence constraints. A generalized model for precedence constraints is given in [55], and a clever prioritization technique informed by the precedence graph is developed in order to guide the scheduling such that tasks that root precedence chains are scheduled earlier to ensure that long precedence chains do not limit the minimization of the makespan. The work in this thesis explores a prioritization preference in the form of a soft precedence constraint, and adapts existing methods that deal with hard constraints.

2.5.2 Job Sequencing and Tool Switching

There is strong resemblance between our problem and the job sequencing and tool switching problem (SSP) a well studied problem that deals with sequencing tasks that require different tools. As in our application, where hard task precedence constraints do not inherently exist, the SSP seeks to sequence tasks such that some objective (i.e., makespan, efficiency, tool switches) is optimized [56]. SSPs often deal with only one machine, and optimal scheduling solutions have only been found for single-machine SSPs [58], which aren't related to our work. Multi-machine SSPs, like MRTA solutions, use approximate methods for scheduling. In [57], an integer linear program (ILP) was used with a neighborhood search heuristic to minimize schedule makespan on multiple parallel machines. In [71], two MILPs were used along with a tabu search heuristic to schedule parallel machines, and constraint programming was used

in [72] to solve the same problem. The difficulty with the SSP solutions to MRTA problems is that the machines don't have the same conflict potential as robots do, and therefore don't have the same cross-schedule dependencies that our problem has. We therefore approach the tool change minimization problem as an MRTA problem with precedence constraints.

Chapter 3

PARTITION-BASED TASK ALLOCATION AND SCHEDULING

This chapter describes the nominal task allocation and scheduling methods as well as the market-based schedule optimization algorithm in the two-stage schedule execution method. The methods are then tested on an example problem of wing skin attachment by four robots.

3.1 Problem Formulation

Consider the class of manufacturing problems where m pairs of homogeneous, stationary robots r_i perform n discrete assembly tasks on a work part, where $n >> m$. Since task allocation of any robot inherently depends on the schedules of all the other proximate robots in order to avoid collisions, this class of problems best fits in the single-task robots, single-robot tasks, time extended assignment with cross-schedule dependencies (XD[ST-SR-TA]) category of problems in the MRTA taxonomy [30]. The robots are situated about the part along opposing pairs. The reach of each robot is limited to a subset of tasks but overlaps with the reach of the adjacent and opposite robots. This arrangement, shown in Fig. 3.1, defines an axial direction that is leveraged for task scheduling in the next section.

The task allocation and scheduling problem seeks to minimize the spread in total schedule completion times, or makespans, amongst the robots and avoid all robot-robot collisions. The methods in this chapter aim to provide a foundation for solving this class of manufacturing problems.

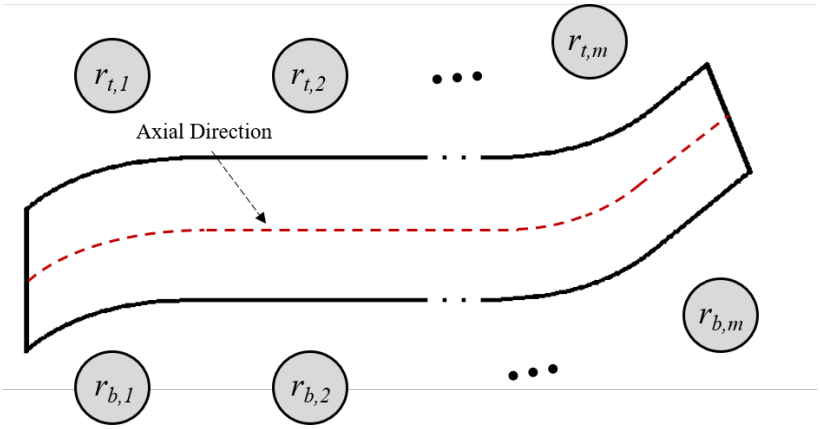


Figure 3.1: A generalized aerospace/aircraft part that has tasks (drilling, fastening, trimming, painting, etc.) (nearly) uniformly distributed across the work part. The part is assembled by m robot pairs situated around the work part defining an axial direction. The robots have limited but overlapping reach, and experience periodic failures that require them to be pulled out of operation for maintenance.

3.1.1 Schedule Efficiency Metric

The MRS schedule efficiency metric ϵ is given as

$$\epsilon = \frac{t_{min}}{t_{act}} \tag{3.1}$$

where t_{min} is the minimum possible time for all the robots to complete all the tasks, or makespan, and t_{act} is the actual makespan. Here, t_{min} is assigned to be the sum of the service times $t_{s,j}$ of all the n tasks, and $t_{f,i}$ is assigned to be the sum of the failure repair times for all the $2m$ robots, split evenly among all the robots. Repair time does not penalize efficiency as the robots are not capable of doing work during this time. This gives the expression

$$t_{min} = \frac{\sum_{j=1}^{n} t_{s,j} + \sum_{i=1}^{2m} t_f^i}{2m} \tag{3.2}$$

For the current application, travel time between the tasks is significantly less than the task completion time. So, we include an approximated travel time in each $t_{s,j}$ and do not consider it separately in our formulation. We use t_{act} to denote the time the last-to-finish robot takes to complete its allocated tasks, which is the maximum sum of the makespan t_s^i (the time for robot i to complete all its assigned tasks) and its repair time t_f^i, expressed as,

$$t_{act} = \max_i (t_s^i + t_f^i) \tag{3.3}$$

Combining (3.2) and (3.3), the efficiency metric for a task schedule is expressed as

$$\epsilon = \frac{\sum_{j=1}^{n} t_{s,j} + \sum_{i=1}^{2m} t_f^i}{2m \max(t_s^i + t_f^i)} \tag{3.4}$$

Task schedules are evaluated using this efficiency metric.

3.2 Conflict-Free Nominal Scheduling

This section outlines the proposed nominal scheduling method. We first show the existence of a solution for a uniform task distribution, and then show robustness of the solution due to COA variations by relaxing the uniformity assumptions. For m opposing robot pairs, we designate the makespan for the i-th pair as t_p^i, and further distinguish the top and bottom robot makespans as t_t^i and t_b^i, respectively.

3.2.1 Fair Partitioning

Our method makes the following three assumptions about task distribution uniformity, which are typical for aerospace assembly applications.

Assumption 1 *The task distribution uniformity allows the workpart to be divided into m regions along the axial direction with equal service time t_p, each assigned to a robot pair, assuming no failure occurrence.*

Assumption 2 *Each opposing robot pair is placed such that the tasks in its assigned region can be assigned evenly between the two robots into top and bottom regions (with task equal task service times $t_t^i = t_b^i = 0.5t_p$) as illustrated in Figure 3.2.*

Assumption 3 *As the robots complete their assigned tasks, they traverse the workpart in the axial direction at equal rates q.*

Remark 1 *These assumptions are valid for a variety of aerospace manufacturing tasks, e.g. drilling, fastening, trimming, and painting, using homogeneous robots (drilling, fastening, trimming, painting of wings, fuselages, empenages, stabilizers).*

By Assumptions 1 and 2, we use partitioning to achieve equitable task allocation among all the robots and minimal idle cost, resulting in $2m$ partitions, each one assigned to a different robot. We employ a pairwise optimization method as in [28] for partitioning. An example partitioning of the work part is shown in Figure 3.2.

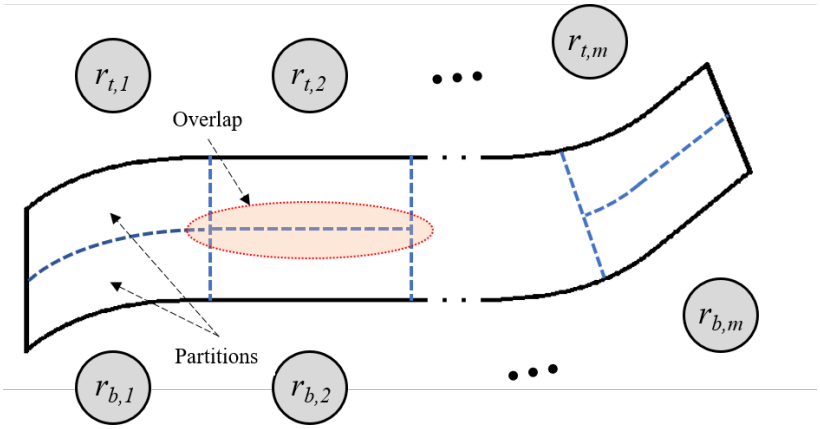


Figure 3.2: Partitioning allocates the tasks among all the robots, such that the completion time of all the partitions is equal in order to minimize idling. The overlap region contains tasks that can be serviced by more than one robot, and can, therefore, be reallocated to balance the workload of the robots.

3.2.2 De-conflicted Offset Scheduling

The partitioned work is scheduled by sequencing all the tasks in every partition in the same axial direction and offset the start locations of all the robots on one side (top or bottom) of the work part as shown in Figure 3.3 (a) [1]. We denote the i-th offset and non-offset robots as top r_t^i and bottom r_b^i robots (as in Figure 3.3), respectively, but either the top or bottom robot can be offset in practice. The robots r_t^i with offset start locations return to finish the rest of the work in their respective partitions using the same axial sequencing when they reach the partitions' ends, as shown in Figure 3.3 (a). For the i-th offset robot, r_t^i, we designate the offset location as a distance l_t^i from the start of its partition, which defines a distance l_t^{*i} to the end of its partition, as in Figure 3.3. These have traversal times (time to complete the work) t_t^i and t_t^{*i}, respectively.

Our first claim is that the offset-based task scheduling, subject to the following two constraints, ensures safety, i.e., collision-free operation. Safety is specified by the spatial proximity constraint that requires the distance between the end-effector positions (task locations) of any two robots to be at least αd_{ee} at all times, where d_{ee} is the diameter of the end-effectors and $\alpha \geq 1$ is a safety factor.

Constraint 1: We constrain the robot spacing to be sufficiently large such that the offset distances l_t^i and l_t^{*i} are larger than the proximity constraint:

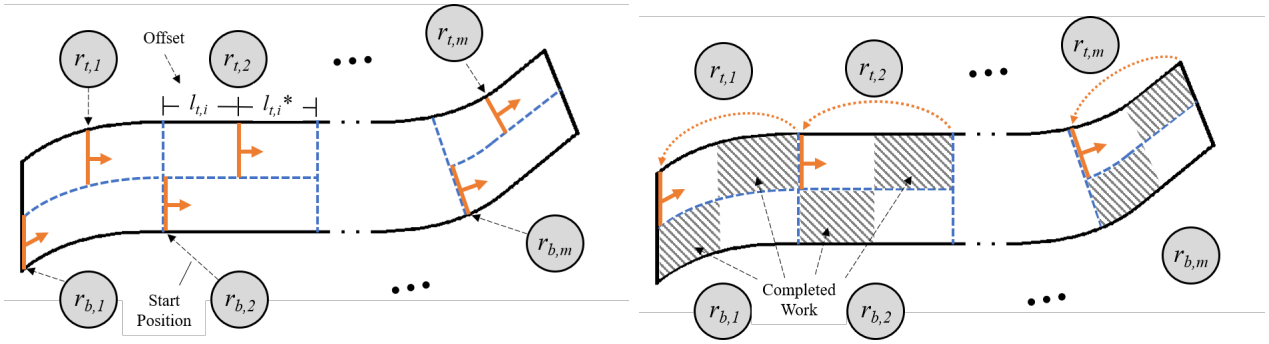
$$l_t^i > \alpha d_{ee} \quad \text{and} \quad l_t^{*i} > \alpha d_{ee} \quad . \tag{3.5}$$

Constraint 2: We constrain the offset of robot r_t^i such that the time it takes to reach the end of its partition, t_t^{*i}, is greater than that of r_t^{i-1}, for $i > 1$:

$$t_t^{*i} > t_t^{*i-1} \quad . \tag{3.6}$$

The first constraint ensures that each robot has sufficient spatial separation from its

[1] This is feasible only if the task ordering constraints do not prevent axial sequencing.



(a) Offset Start Locations

(b) Offset Robots Return

Figure 3.3: Sequencing the tasks in each partition in the axial direction and offsetting the start locations of one side of robots (top robots), as shown in (a), enables collision free operation. Once the top robots reach the end of their partitions, they return to the start of their partitions, as shown in (b), and service the remaining tasks in the same axially sequenced manner. The bottom robots have moved out of the way at this point, and no collisions occur.

opposite and adjacent robots at the start of the operation. Note that this constraint also implicitly limits the length of any partition to be no less than $2\alpha d_{ee}$, and, consequently, limits robot density. The second constraint ensures that r_t^{i-1} returns to the beginning of its partition before r_t^i does, which precludes collision when r_t^i returns. These two constraints together allow collision-free operation because **Assumption 3** ensures that the robots have equal axial traversal rates q, and, thus, the spatial proximity constraint is not violated.

In practice, variable task service times and COAs with missing tasks can cause the axial traversal rate of a robot to fluctuate during operations requiring the solution to be robust. Our second claim is that **Assumption 3** can be relaxed to allow for variation δ_q in the axial traversal rate q and still preserve the collision-free nature of the nominal schedule, provided that the maximum variation δ_q in the traversal rate q is small, and the constraint inequalities in (3.8) and (3.9) are satisfied. This follows from the fact that the distance $d_{b^i,t^{i-1}}(t)$ between

the i-th non-offset robot r_b^i and the previous offset robot r_t^{i-1}, given as

$$d_{b^i,t^{i-1}}(t) = l_t^{*i-1} - (q_t^{i-1} - q_b^i)t > \alpha d_{ee} \tag{3.7}$$

can decrease at the maximum by $2\delta_q t$ in time t where $2\delta_q$ is the maximum difference in the traversal rates of the two robots. Then, collision between robots r_b^i and r_t^{i-1} can be avoided (before robot r_t^{i-1} returns to the beginning of its partition) by ensuring that the distance $d_{b^i,t^{i-1}}(t)$ is sufficiently large, i.e.,

$$d_{b^i,t^{i-1}}(t) = l_t^{*i-1} - 2\delta_q t > \alpha d_{ee} \tag{3.8}$$

where the offset l_t^{*i-1} is the initial distance between them. Similarly, collision between the i-th non-offset robot r_b^i and the next offset robot r_t^i is avoided (before r_t^i returns to the beginning of its partition) provided

$$d_{b^i,t^i}(t) = l_t^i - 2\delta_q t > \alpha d_{ee}, \quad t \leq t_t^{*i} \quad . \tag{3.9}$$

We also avoid collision between robots r_t^i and r_b^i after r_t^i returns to the beginning of its partition, as in Figure 3.3. When robot r_t^i returns, the distance d_{b^i,t^i} between the two robots, which is initially at least $(q - \delta_q)t_t^{*i}$, needs to satisfy

$$d_{b^i,t^i}(t) = (q - \delta_q)t_t^{*i} - 2\delta_q(t - t_t^{*i}) > \alpha d_{ee} \tag{3.10}$$

for time $t > t_t^{*i}$. If the variation δ_q in the axial traversal rate q is small, then the inequalities in (3.8), (3.9), and (3.10) are satisfied because (i) their left hand sides approach the spacing between the robots (l_t^{*i-1}, l_t^i and l_t^{*i}, which are sufficiently large from Constraint 1; (ii) the time t is finite and bounded by the maximum makespan; and (iii) the minimal distance $(q - \delta_q)t_t^{*i}$ approaches l_t^{*i}. Thus, the above analysis shows the existence of a collision-free nominal schedule if the maximal variation δ_q in the axial task execution rate q (due to COA

variation) is not too large.

3.2.3 Schedule Execution and Failure Handling

To maintain the collision-free nature of the nominal schedule and avoid frequent rescheduling, a robot returning from maintenance after a failure occurrence returns to the place in its nominal schedule where it would have been had no failure occurred. This requires the robot to skip some of its work to be dealt with after the nominal schedule is completed. These leftover tasks are then reallocated among the robots, and are executed after the nominal schedule is completed. In this sense, our scheduling framework is an adaptive approach for handling dynamic environments.

3.3 Market-Based Leftover Schedule Optimization

This section presents the leftover scheduling method and task reallocation algorithm. Leftover work, hereafter referred to as leftovers, can be disbalanced and non-uniformly distributed, and the main thrust behind leftover scheduling is for robots to pick up work from other robots in order to balance the workload. For workload rebalancing to be possible, we guarantee that all the robots have leftover tasks in areas of overlapping reach with at least one other robot by omitting a subset of tasks in the region of overlap from the nominal scheduling and relegating them to the leftover scheduling. The approach first constructs a conflict-free leftover schedule that is not necessarily balanced, and then uses a market-based algorithm to balance the workload and optimize the schedule. It is necessary for the initial schedule to be conflict free because the algorithm can then be constrained to reallocate the tasks without introducing collisions. To simplify the construction and refinement of the leftover schedule, the adjacent tasks are grouped into larger portions of work, referred to as cities (as in the TSP formulation). The reduction of problem complexity decreases computation time (scheduling cities rather than individual tasks) at the cost of efficiency. This trade-off is discussed more later.

3.3.1 Initial Leftover Scheduling

The initial scheduling is constructed by modifying the partitioning and offset-scheduling method used for nominal scheduling to ensure that the decreased uniformity in leftovers does not cause collisions. For this purpose, we use the overlap region between the robot pairs with width $w > \alpha d_{ee}$, and extend part of the robots partitions over the width of this region, the $l_{t,i}$ part of the $r_{b,i}$'s partition, and the $l^*_{t,i}$ part of $r_{t,i}$ partition, as shown in Figure 3.4. Note that this region should contain sufficient number of tasks relegated from the nominal schedule such that the maximum deviation $\delta_{q,L}$ in the axial traversal rate q_L is sufficiently small in the extended regions even in the presence of leftovers.

We claim that the offset-scheduling method results in a collision-free initial leftover schedule, provided: (i) the robots are able to reach the tasks in the overlap region; (ii) the spacing constraint in (3.5) is satisfied; (iii) the time $t_{b,i}$ that $r_{b,i}$ takes to service the extended part of its partition is not more than the time $t^*_{t,i}$ that $r_{t,i}$ takes to service its extended part, i.e., $t^*_{t,i} = t_{b,i}$; and (iv) the partitioning is such that the service times of the extended parts increase along axial direction, i.e., $t_{b,i} < t_{b,i+1}$ to ensure that the constraint in (3.6) is satisfied. The claim follows since both robots, $r_{t,i}$ and $r_{b,i}$, do not move to their second non-extended regions (shown as the gray regions in Figure 3.4) before they complete the tasks in their extended regions. Note that the distance between the robots in the extended regions can be shown to remain sufficiently large, using arguments similar to those in inequalities (3.8) and (3.9), provided the variation in the axial traversal rate is sufficiently small. Moreover collisions are avoided when the robots are in the gray (non-extended) regions since the overlap distance is larger than $w > \alpha d_{ee}$.

3.3.2 Market-Based Optimization Algorithm

The schedule optimization algorithm uses a market based method to re-balance and reschedule the leftover cities in the initial leftover schedule in order to reduce idle cost. There are n_c cities, $2m$ robots, $r_i \in \mathcal{R} = \{r_1, \ldots, r_{2m}\}$, and a corresponding schedule matrix

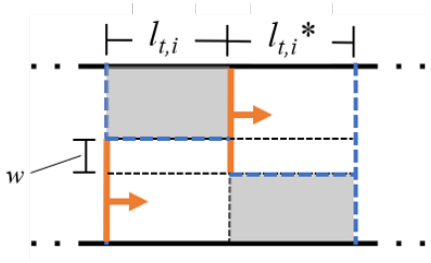


Figure 3.4: Extending the partitions to cover an overlap region (containing tasks intentionally left out from the nominal schedule and relegated to the leftover schedule) to ensure the maximum deviation $\delta_{q,L}$ in the axial traversal rate q_L is sufficiently small in the extended regions (shown in white) even in the presence of leftovers. This reduction of traversal rate variations along with sufficient sizing of the overlap region ensure that the initial leftover schedule is collision free.

$\mathbf{S} = [S_1 \dots S_{2m}]^T$, where S_i is the schedule of robot r_i. Note that $r_{2i-1} = r_{b,i}$ and $r_{2i} = r_{t,i}$ from the previous section, since the distinction between the top and bottom robots is not necessary for the market-based method. The vector $S_i = [s_{i,1} \dots s_{i,l-1}\ c_j\ s_{i,l+1} \dots s_{i,n_i}\ 0 \dots 0]$ contains cities $s_{i,l} = c_j$, denoting that city c_j is serviced by robot r_i and that it is the l-th city r_i services (has placement l in S_i); each S_i contains n_s elements, of which the first n_i are non-zero cities. We take n_s as either the smallest number of the lowest service time cities it takes to exceed an equal workload, or the number of cities in the schedule with the most cities, whichever is larger. City c_j has service time $t_{s,j}$; all the n_c cities must be in $\mathbf{S}$ for the schedule to be complete. The time it takes robot r_i to complete its schedule S_i is defined as $t_i = \Sigma t_{s,j}$ for all c_j in S_i.

In this market, robot r_i is looking to sell a city c_j in its schedule S_i to any robot $r_{k \neq i}$, as shown in Figure 3.5. The seller is a robot who *most* wants to sell one of its cities, and, likewise, the buyer is a robot who *most* wants to buy that city, and an agent's desire to buy or sell is proportional to the agent's deviation from an equal workload distribution, measured by a robot's *utility*. For robot r_i, $u_i = t_i / \Sigma t_i$, and the utility vector $\mathbf{U} = [u_1 \dots u_{2m}]$ contains

the utilities of all the robots. The *price* at which any robot r_i wants to sell a city c_j in its schedule is determined by the service time $t_{i,j}$ of c_j (normalized by the maximum city service time $\max(t_{i,j})$). The price matrix $\mathbf{P}$ consists of $2m$ row vectors of price, one for each robot r_i, and each price vector $\mathbf{P}_i$ contains n_c elements, where each element $p_{i,j}$ is the price of city c_j if c_j is in S_i, otherwise $p_{i,j} = -\infty$. Note that the price of a city c_j is independent of its placement in r_i's schedule S_i. The city most desired to be sold, and consequently its seller, is chosen on a basis of *margin*, which we define as $m_{i,j} = (\beta * u_i) + p_{i,j}$. Here, β is a weighting factor that is used to bias the sellers to be more concerned with the utility than price, a similar market biasing approach as used in [32] for increasing optimality. From the margin matrix $\mathbf{M}$, the best seller and best city to sell is found based on the maximum margin, $i^* = \text{argmax}_i\, m_{ij}$ and $j^* = \text{argmax}_j\, m_{ij}$, respectively.

Once the best seller r_{i^*} and the desired city to sell c_{j^*} is chosen, a set of candidate buyers is chosen on the basis of which robots have the *ability* to service c_{j^*}. Ability a_{k,j^*} in this application is determined by whether c_{j^*} is in r_k's reach, and is 1 if it is, 0 if it is not. Each candidate buyer is then checked to see if the maximum margin value resulting from proposed transaction, m^{*p}, is greater than the current m^*, and is disqualified from the candidate set if $m^{*p} > m^*$. For each buyer r_k in the candidate buyer set, the proposed transaction (resulting in proposed utility, price and margin matrices, $\mathbf{U}^p$, $\mathbf{P}^p$, $\mathbf{M}^p$) is evaluated by the reduction in the standard deviation of the utility vector, $\Delta_k = \sigma(\mathbf{U}) - \sigma(\mathbf{U}^p)$, and the best buyer is chosen to be the one whose transaction will yield the highest reduction in the standard deviation of the utility vector, $k^* = \text{argmax}_k\, \Delta_k$. The algorithm seeks to minimize the spread in utility to minimize the imbalance in work distribution. Once the best buyer r_{k^*} is chosen, c_{j^*} is placed in every placement l of r_{k^*}'s schedule S_{k^*}, and a minimum distance w_l between any two robots at any time for the entire schedule $\mathbf{S}$ is determined for each placement of c_{j^*} in S_{k^*}. A description of how w_l is calculated is given below. The best placement l^* for c_{j^*} in S_{k^*} is chosen as the one with the max minimum distance, $l^* = \text{argmax}_l\, w_l$. The pseudo-code for the method is provided in **Algorithm 1**.

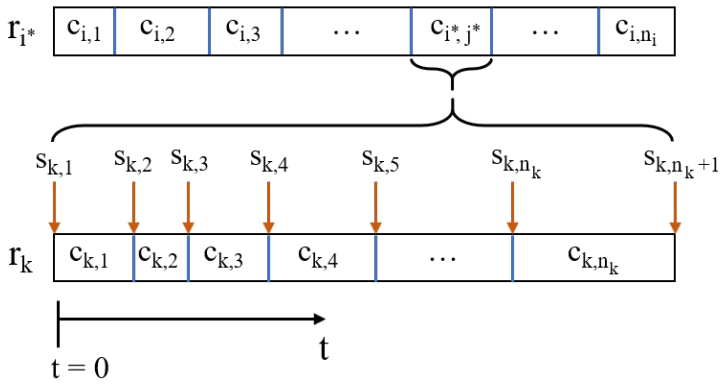
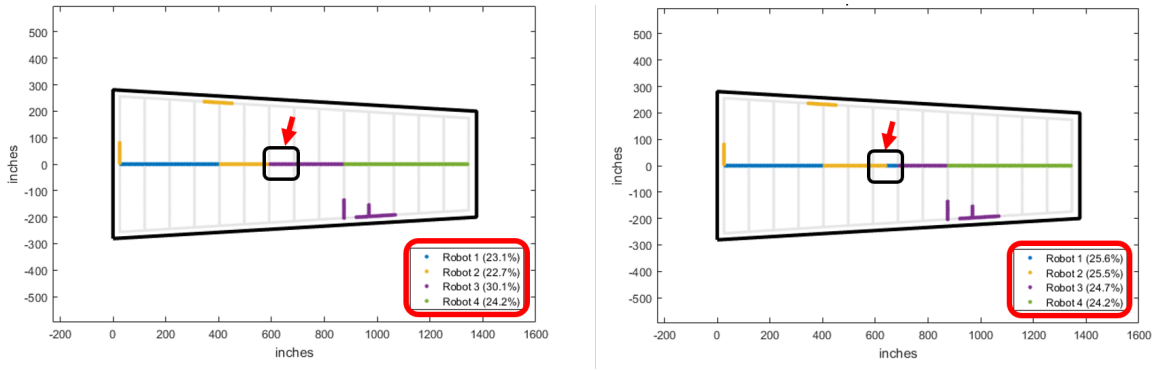


Figure 3.5: Shown here is a visualization of robot schedules, and an illustration of the market-based method. Cities are given an additional subscript that isn't used in the text to help illustrate which robot they belong to. The method finds the most motivated seller r_{i^*} and the best city to sell c_{j^*} in order to get the highest reduction of work dis-balance. Candidate buyers are chosen and ranked by the overall reduction in workload dis-balance. Candidates are tried by their ranked order to see if they are able to buy the city from the seller without introducing collisions. Here, buyer candidate k is attempting to find a placement for c_{j^*} (shown here as c_{i^*,j^*}) in its schedule.

If the largest minimum distance $w^* = max(w_l)$ is greater than the collision threshold, r_{i^*} sells c_{j^*} to r_{k^*} and inserts c_{j^*} in placement l^*, yielding the maximum geometric separation of the robots during the whole schedule. If w^* is less than the collision threshold, we choose the next best buyer (based on next highest Δ_k), and proceed to check all the placements of c_{j^*} in the new buyer's schedule. This ensures no collisions are introduced into the schedule. If no suitable buyers are available for c_{j^*}, the next highest margin $m_{i,j}$ is found (resulting in a new city and seller), and the process for selling the city is repeated. This is done until a city is sold, or until all the margins, $m^* - p_{i^*,j^*} \leq \mu(\mathbf{U})$, are chosen and checked for potential buyers[2]. If no sale happens, no further optimization is possible. This algorithm is executed on the initial leftover schedule. Figure 3.6 shows an example of how the market-based optimization balances the robots' schedules.



(a) Initial Schedule

(b) Schedule After Optimization

Figure 3.6: An example of how the market-based optimization algorithm balances the schedules between the robots. Notice that the algorithm re-allocates certain task cities from robots that have more assigned work to robots that have less assigned work in order to balance the spread in workload.

[2]This guarantees the convergence of Algorithm 1.

3.3.3 Distance Calculation for Collision Avoidance

Placement of the city within the schedule is chosen on the basis of maximum spatial separation of the robots throughout the operation of the whole candidate schedule. To this end, a pairwise distance checking process checks the distances between each robot pair for the whole schedule. Since cities are grouped by proximity, each city has its own collision-free radius that is taken to be the sum of the robot end effector radius and the largest distance between the city's geometric centroid and any hole in that city. This allows to check the distance between robots when they are servicing cities instead of checking for every hole in the schedule. The distances are checked by robot pairs by checking the distance between every temporally coincident pair of cities, as shown in Figure 3.7. The distance between any two robots for a given pair of cities is determined by subtracting the two cities' collision radii from the distance between their centroids. The minimum distance for the whole schedule for all the robot pairs is then taken to be as the distance w_l. Collision is then detected if $w_l < 0$ (since the end effector radius is already in the cities' collision-free radius).

3.4 Experiments and Results

Considered here is the problem of drilling holes in a section of an aircraft wing box by four robots situated around the wing An example aircraft wing is used, featuring 15 ribs and three spars. Each spar consisted of 266 holes, while the ribs contained 109, 107, 105, 101, 99, 95, 93, 91, 87, 85, 83, 79, 77, 73, and 71 holes, respectively. Rib hole drill (service) time was set to 30 s per hole for the largest rib, and then decremented by 0.5 s for each successive rib (e.g., 29.5 s per hole for the next largest rib, 29 s per hole for the one after that, and so on). Every spar section between any two adjacent ribs was assigned the same hole drill time as that of the longer rib. We used $\alpha = 1.5$ and $\beta = 20$ as the parameters in our methods, and d_{ee} was 2 ft. The example wing is shown in Figure 3.8.

All the scheduling methods were implemented in MATLAB R2016b on a quad-core Core i7-4870HQ machine with 16 GB of RAM running Windows 10. The results were validated

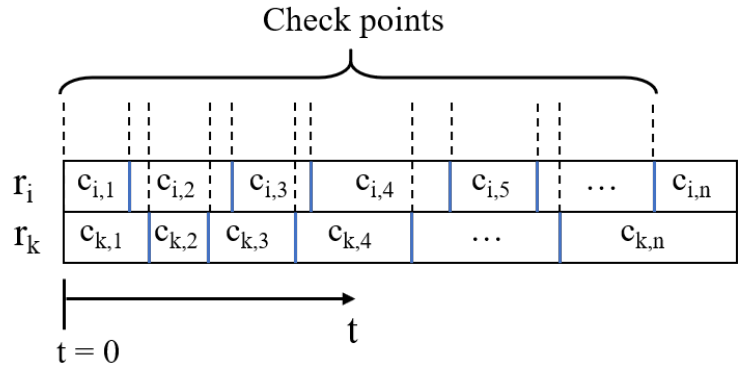


Figure 3.7: To check for collisions between any two robots, each temporally coincident pair of cities must be checked against a collision threshold. Since cities are grouped by proximity, each city has its own collision-free radius that is taken to be the sum of the robot end effector radius and the largest distance between the city's geometric centroid and any hole in that city. Here, the threshold is to see if the distance between the city pair's centroids is larger than the sum of their collision-free radii.

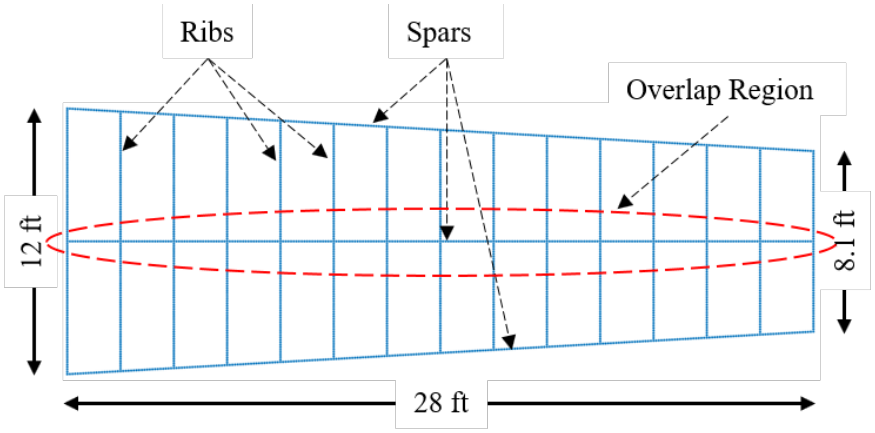


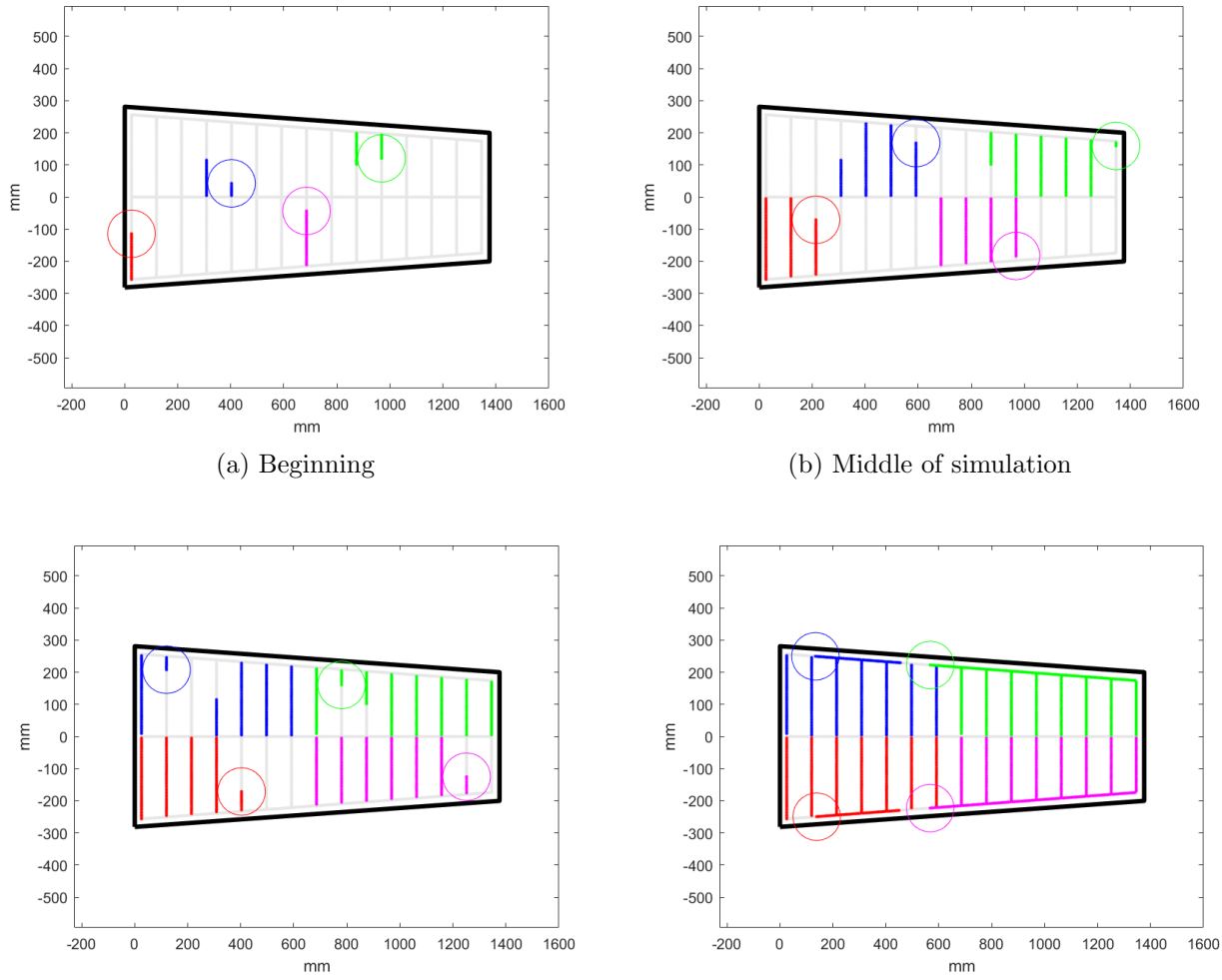
Figure 3.8: Mock three-spar, 15-rib wing used for the testing of our methods. Ribs have even, two-foot spacing. The middle spar, highlighted in red, offers regions where all the robots are able to service tasks in the workload re-balancing stage.

on a physical robot cell featuring four ABB IRB-120 arms positioned about a miniature aluminum mock wing built according to the dimensions shown in Figure 3.8. This setup is a 15% scale version of a feasible manufacturing setup for drilling a full-sized, real aircraft wing. The wing is raised from the floor on an aluminum T-frame that holds plexi-glass shields. The robot controllers, situated on the floor beneath the robots, are collectively directed via the cell control software. The physical cell is shown in Figure 3.10. The system validated our assumption of the travel times between holes being negligible relative to the task service times.

The cell control software was responsible for synchronized starting of the robots' programs, as well as keeping track of each robot's schedule and feeding hole targets to the controllers (each controller always had the next three targets). It also monitored the synchronization of the robots with each other, and determined how many holes were relegated to the leftovers for the robots returning from failures. The program was implemented in C# using the ABB PC SDK 6.07 on a quad-core Core-i5 6400 machine with 8 GB of RAM, and communicated with the robot controllers via Ethernet UDP through a network switch. A redundant collision monitoring system used streaming robot position data to reconstruct simple 2D shapes representing the robots with the desired safety margins, and checked for intersections among the shapes. This was implemented using Python 3.7 and ran on a separate machine with the same specifications. A more detailed description of the software components and the associated architecture developed are provided in Appendix B.

Greedy Algorithm

The performance of the proposed method was evaluated by comparing against the performance of a greedy scheduler. The greedy method used the same scheduling framework (time-constrained nominal scheduling and leftover work rescheduling). For the nominal schedule, the greedy method used the same partitions for task assignment and the same start positions as the proposed method, but sequencing of the holes was done using a nearest neighbor search. Similarly, the leftover partitioning was done using the proposed method, and the



(a) Beginning

(b) Middle of simulation

(c) Middle of simulation

(d) End of simulation

Figure 3.9: Example of a MATLAB simulation showing execution of a nominal schedule. Robot (and their completed tasks) are color-coded, and the location of a robot end effector is shown as a circle.

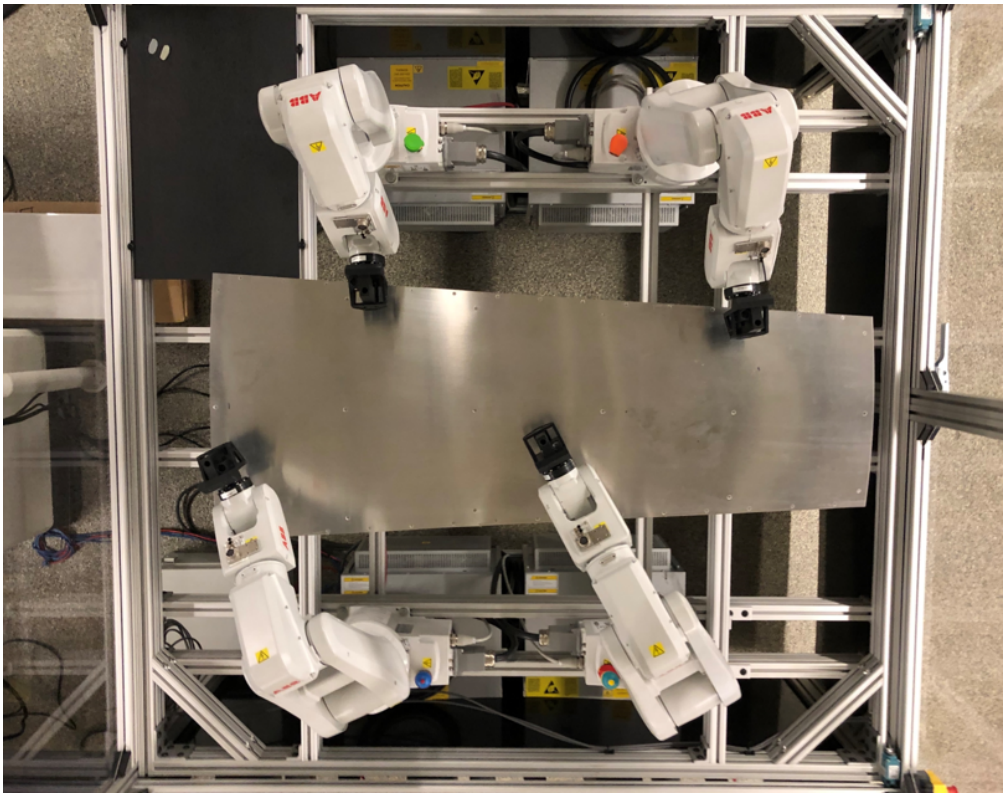


Figure 3.10: A 15% reduced scale physical cell used to test the scheduling methods and validate the simulation results. The robot controllers interfaced with the cell control software, where the generated schedules where executed.

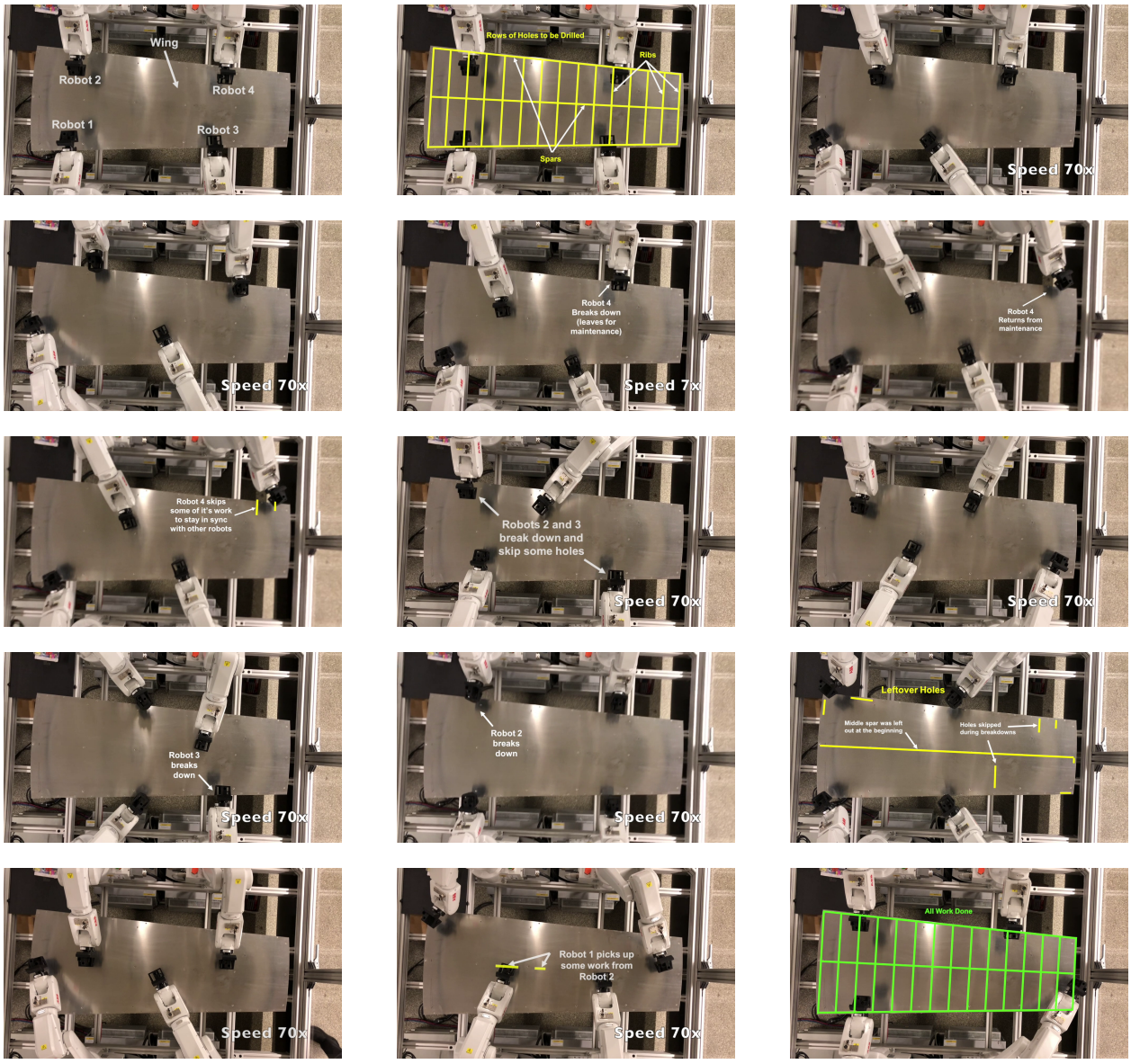


Figure 3.11: Snapshots of a video showing the operation of the physical robot cell.

sequencing done using nearest neighbor search. Because a nearest neighbor search does not guarantee a collision free schedule, the robot whose next move towards the next task would result in a collision must wait until the robot with whom the conflict would've arisen moves on, and then can resume with the next task no longer being a potential cause for collision. This eliminates any collisions and introduces wait gaps in the schedules. A greedy scheduler like this is representative of how a robot programmer would schedule the drilling tasks in a typical industrial application.

3.4.1 Experiments

Both of the methods were evaluated using five different COA cases with 100 failure instances for each COA case, leading to a total of 500 test scenarios. The COA cases were generated to have gradually more components missing from the COA (inadvertently omitted parts from the drilling process, as typically seen in the industry), with the first case being the full wing with no missing component (generic COA case).

The failure instances were generated for each robot by randomly drawing a first occurrence time (time at which the first failure occurred), recurrence time, and repair time. These three parameters were drawn randomly for each robot (duration was randomized for each failure) from normal distributions. The means and standard deviations of the distributions were determined using a nominal value of how many holes a $\frac{1}{4}$ inch drill bit would last in aluminum and an average drill bit replacement time. These were taken to be 300 holes and 8 minutes, respectively. Table 3.1 outlines the distribution parameters used for the generation of failure instances. Note that first occurrence time is lower than the recurrence time, accounting for the possibility of first failures occurring earlier if drill bits are not replaced from a previous drilling operation. These parameters resulted in an average of 45 leftover cities per test scenario, each containing between two and 16 holes.

3.4.2 Results

Performance improvement: Figure 3.12 compares the efficiency of our method to

Table 3.1: Normal Distribution Parameters for Robot Failure Occurrence Time, Recurrence, and Repair Time

Parameter	μ	σ
First Occurrence	5073 s	1602 s
Recurrence	6942 s	1068 s
Repair Time	480 s	80 s

that of the greedy scheduler, where we observe about 98% efficiency on an average with our method. For every COA case, the proposed method generates more efficient schedules than the greedy scheduler, with the efficiency increasing by about 13% on an average. This result is especially promising for high volume airplane assembly lines as 13% improvement on a five-hour wing skin attachment lead time results in a 40 minute saving. It is also observed that our method yields significantly more consistent schedule efficiency across the COA cases (all the values are within 0.5% of each other) as compared to the greedy method (the values vary by as much as 12%). This trend indicates that the proposed method is not affected by non-symmetric task distribution as much as the greedy scheduler.

Impact of market-based optimization: To characterize the impact of the market-based optimizer, it is compared with the partition-based scheduler without leftover optimization. We observe that the partition-based scheduler still outperforms the greedy method significantly, even in the absence of optimization. The market-based optimizer, however, improves the schedule efficiency by about 2% on an average, from 96% to 98%. This improvement is quite substantial when implemented across multiple processes for a high-flow aircraft production line. The results in Figure 3.12 are grouped by COA case[3], and are summarized in Table 3.2.

Computation time: Figure 3.13 compares the computation times of both the meth-

[3]The inferior performance of the greedy method on COA 2 is a result of a rib missing from this COA, causing the robots to move toward each other, leading to long wait times in their schedules.

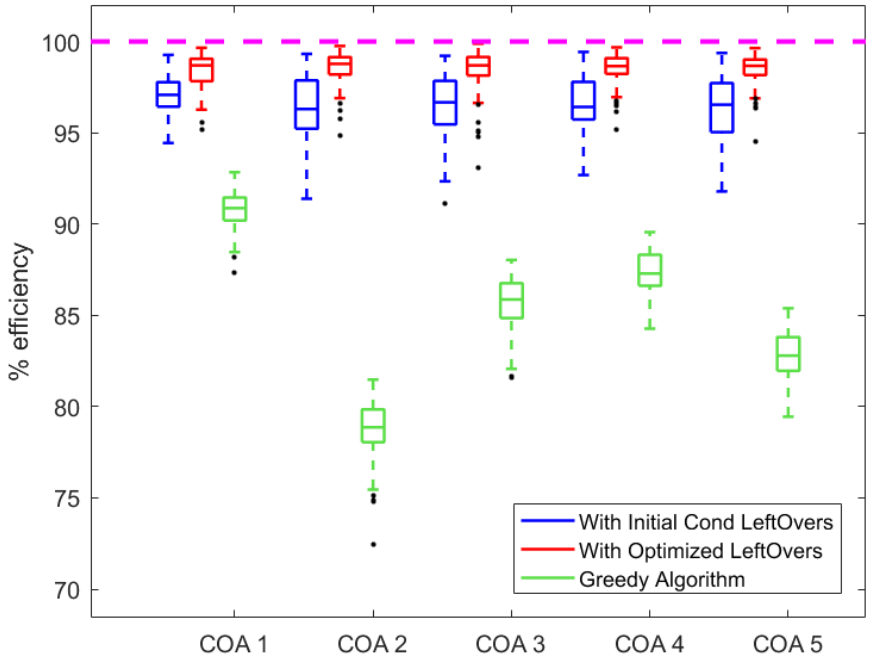


Figure 3.12: Comparison of the overall schedule efficiencies of our two-stage scheduling method with market-based leftover optimization, without leftover optimization, and a greedy M-TSP solver.

Table 3.2: Summary Statistics of Schedule Efficiencies for the Proposed and Greedy Methods

Avg. ϵ (Proposed)	98.5% $\pm$ 0.92%
Min. ϵ (Proposed)	93.1%
Avg. ϵ (Greedy)	85.0% $\pm$ 4.27%
Min. ϵ (Greedy)	72.4%

ods, and shows them both to be competitively low. The greedy method has lower computation times, which is expected since the greedy scheduler is computationally simpler, the difference being about 18 ms on an average. Even so, the proposed method solves the scheduling problem in 122 ms on an average, and has a maximum computation time of only 300 ms. In practice, this would allow an industrial MRS to start drilling almost immediately. The computation time results in Figure 3.13 are grouped by COA case, and are summarized in Table 3.3.

Table 3.3: Summary Statistics of Computation Times for the Proposed and Greedy Methods

Avg. Comp. Time (Proposed)	122.5 $\pm$ 37.3 ms
Max. Comp. Time (Proposed)	300.5 ms
Avg. Comp. Time (Greedy)	104.9 $\pm$ 21 ms
Max. Comp. Time (Greedy)	170.31 ms

Effect of discretization: The discretization of the leftover tasks into cities reduces the resolution of the problem, and decreases the potential for finding a perfect workload balance among the robots. Consequently, none of the experiments yield an efficiency of 100%. This coarsening of the resolution, however, makes the initial scheduling and subsequent leftover optimization problems smaller by an order of magnitude (40 cities versus 400 holes). This greatly reduces the computational complexity of the market-based method, since there

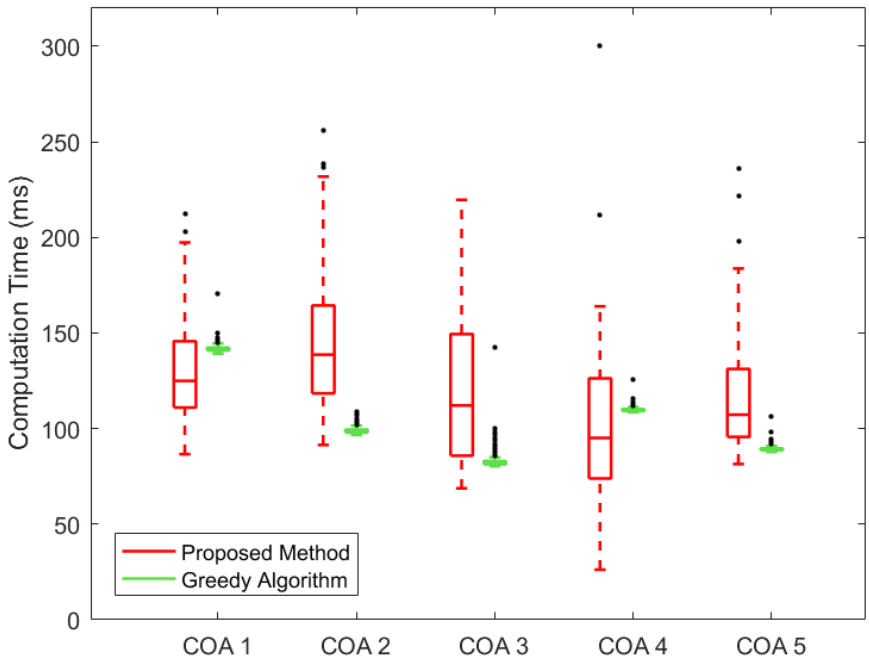


Figure 3.13: Computation times for all the experiments using the proposed method and greedy method, grouped by COA case.

significantly fewer schedule placements to check [10] Therefore, it enables competitive computation time to that of the greedy scheduler, but still yields significantly higher schedule efficiency. The value of fast leftover optimization is expected to become more pronounced when the size of the problem is increased, as, for example, when there are more than 10 robots on a full-sized wing.

Failures during leftover scheduling: This method assumes that no failures occur during the execution of the leftover schedule. While this assumption holds true when the amount of leftovers is small, the risk of incurring a failure in the leftover stage grows as the accumulation of failures increases. The possibility of failures occurring during the leftover schedule execution can be largely precluded by performing preventive maintenance just before the execution of the leftover schedule. The limit for the amount of leftover work is then driven by the failure recurrence distribution.

3.5 Conclusions

In this chapter, partition-based task allocation is adapted to the multi-robot task scheduling problem for the assembly of aircraft structures. The method takes advantage of known problem structure for efficient, collision-free task scheduling, and employs a dual-stage schedule execution strategy to handle robot failures. A market-based optimization algorithm is presented to help recover the efficiency lost due to the robot failures. Results show that the method produces high schedule efficiencies and favorable computation times as compared to a greedy method, and works well on a physical four robot assembly cell. Since aircraft assembly processes are generally well structured problems, the methods are adaptable to suit a wide variety of aircraft components.

Algorithm 1 Update schedule matrix $\mathbf{S}$ by selling a city such that the robots' utilities are maximally equalized

Compute $\mathbf{U}$, $\mathbf{P}$, $\mathbf{M}$ from $\mathbf{S}$
$m^* = \max(m_{ij})$
$i^* = \text{argmax}_i\, m_{ij}$; $j^* = \text{argmax}_j\, m_{ij}$
SALE $\leftarrow$ FALSE
while SALE = FALSE **do**
    $\Delta \leftarrow \emptyset$
    for each $r_k \in \mathcal{R} \backslash r_{i^*}$ **do**
        if $a_{k,j^*} = 1$ **then**
            $\mathbf{S}^p = \mathbf{S}$
            $S^p_{i^*} = [s_{i^*,1} \dots s_{i^*,j^*-1}\ s_{i^*,j^*+1}\ 0 \dots 0]$
            $S^p_k = [s_{k,1} \dots s_{k,n_k}\ c_{j^*}\ 0 \dots 0]$
            Compute $\mathbf{U}^p$, $\mathbf{M}^p$ from $\mathbf{S}^p$
            $m^{*p} = \max(m^p_{i,j})$
            if $m^{*p} < m^*$ **then**
                $\Delta_k = \sigma(\mathbf{U}) - \sigma(\mathbf{U}^p)$; $\Delta \leftarrow \Delta \cup \Delta_k$
    while $\Delta \neq \emptyset$ **do**
        $\mathbf{S}^p = \mathbf{S}$; $k^* = \text{argmax}_k\, \Delta_k$
        for $l = 1, \dots, n_{k^*+1}$ **do**
            $S^p_{k^*} = [s_{k^*,1} \dots s_{k^*,l-1}\ c_{j^*} \dots s_{k^*,n_{k^*}} \dots 0]$
            $w_l \leftarrow$ min pairwise robot distance in $\mathbf{S}^p$
        $w^* = \max(w_l)$; $l^* = \text{argmax}_l\, w_l$
        if $w^* > \alpha d_{ee}$ **then**
            $S_{i^*} = S^p_{i^*}$
            $S_{k^*} = [s_{k^*,1} \dots s_{k^*,l^*-1}\ c_{j^*} \dots s_{k^*,n_{k^*}} \dots 0]$
            SALE $\leftarrow$ TRUE
            return $\mathbf{S}^p$
        else
            $\Delta \leftarrow \Delta \backslash \Delta_{k^*}$
    $m^* \leftarrow$ next highest $\max(m_{i,j})$
    $i^* \leftarrow$ next highest $\text{argmax}_i\, m_{ij}$
    $j^* \leftarrow$ next highest $\text{argmax}_j\, m_{ij}$
    if $m^* - p_{i^*,j^*} \leq \mu(\mathbf{U})$ **then**
        SALE $\leftarrow$ TRUE ▷ Sale is not useful
        return $\mathbf{S}$

Chapter 4

OPTIMIZING MRS DESIGN BY ROBOT PLACEMENT AND MOBILITY

This chapter addresses the second research question (**R2**): *How to optimize the design of the multi-robot system with robot base placement and mobility?*

4.1 Introduction

To maximize MRS efficiency, stationary robots must be properly placed in order for them to maximally be able to reach their own assigned tasks and share tasks with proximate robots [45]. In the absence of failures, the straight-forward solution to the placement problem is to place the robots to maximize coverage of each robots' assigned tasks. However, as failures increase in variety and magnitude (occurrence frequency and duration), the need for robots to be able to share work becomes increasingly vital to minimizing robot idle time and maintaining MRS efficiency. As such, the locations of the robot bases not only depend on the coverage of all their assigned tasks, but also on ensuring that appropriate overlap is given to ensure maximal workload sharing potential. Moreover, significant changes in the work conditions, either from severe robot failures or substantial changes in the COA, can require more task sharing than stationary robot MRS can afford, reducing the efficiency of the system. As a results, the loss of efficiency of a production cell becomes increasingly costly to the manufacturer when more of these significant failures occur, and thus, robot mobility to increase the reach of the robots must be considered. However, while mobile robots potentially make for a more efficient MRS, the added cost of a mobility system, such as a rail-mechanism or an automated guided vehicle (AGV), must be considered and weighed against the potential benefit of the mobility system. The research in this chapter explores

stationary robot base placement and robot mobility as potential solutions to the problem of increased robot failures, and

4.2 Related Work

4.2.1 Robot Base Placement

Optimal robot base placement is popularly used for ensuring tasks and task-paths are within a robot's reach [50]. In [50] and [51], this was done using inverse reachability maps [52] to find feasible robot base placements. Where tasks are reachable, robot base placement is used for optimizing task execution. The work in [41] increased the cycle-time of a multi-robot automotive assembly cell through optimal robot base placement. In [48], power consumption of industrial robots was minimized by placing the base so as to minimize the use of the most energy-consuming actuators of the robots. The literature seems

4.3 Cost of Efficiency

4.3.1 Effects of Increased Failures

To show the motivation behind the need for robot base placement/mobility, this section shows the effects of increased robot failures on MRS efficiency. To this end, five failure profiles were generated, each progressively increasing in scope, and are given in Table 4.1 below. Since robot failures are generated from a normal distribution, as in Chapter 3, the failures are increased by increasing the failure occurrence and recurrence frequency, failure duration, and variation of failures across the robots. Note that Profile 1 is the failure distribution used in Chapter 3. At each profile, variation of failures across robots is effected by assigning a failure profile to each robot randomly selected from the set of profiles ranging from Profile 1 to the profile in question (e.g., for Profile 3, each robot is randomly assigned a failure profile between 1 and 3).

The methods in Chapter 3 are applied using the profiles in Table 4.1 in order to show the effects of increased failures, and Figure 4.1 shows the resulting efficiencies for all the failure

Table 4.1: Failure Profile Distribution Parameters for Robot Failure Occurrence Time, Recurrence, and Repair Time

Parameter ($\mu \pm \sigma$)	First Occur.	Recur.	Repair Tm.
Profile 1	5073 ± 1602 s	6942 ± 1068 s	480 ± 80 s
Profile 2	4532 ± 1704 s	6311 ± 1185 s	535 ± 92 s
Profile 3	3991 ± 1806 s	5680 ± 1302 s	590 ± 104 s
Profile 4	3450 ± 1908 s	5049 ± 1419 s	645 ± 116 s
Profile 5	2909 ± 2010 s	4418 ± 1536 s	700 ± 128 s

profiles. Note that as the failures increase, the efficiency decreases.

4.3.2 Cost of Lost Efficiency

Robot mobility systems come in the form of complex mechanisms (rail, AGV, etc) that have an associated cost.To determine if robot mobility is cost-effective, it is necessary to determine whether or not the cost recovered by robot mobility exceeds the cost of a mobility system. To this end, it is necessary to quantify the recovered efficiency in terms of cost (or reward), so that a comparison to the cost of mobility can be done.

The cost of efficiency is determined by first considering the cost of wings manufactured per month, given as

$$c_{wing/month}(t_m) = \frac{24 * 30}{t_{wl}} p_w t_m \epsilon_{nom} \tag{4.1}$$

where t_{wl} is the lead time of drilling one wing in hours, p_w is the price of the wing, t_m is the number of months, and ϵ_{nom} is the nominal efficiency (the mean efficiency as found in Chapter 3). The cost of lost efficiency from increased failures is then

$$c_{eff,lost}(t_m, \epsilon_{low}) = \frac{24 * 30}{t_{wl}} p_w t_m (\epsilon_{nom} - \epsilon_{low}) \tag{4.2}$$

where ϵ_{low} is the efficiency of low efficiency cases. Conversely, the cost (or reward) of recovered

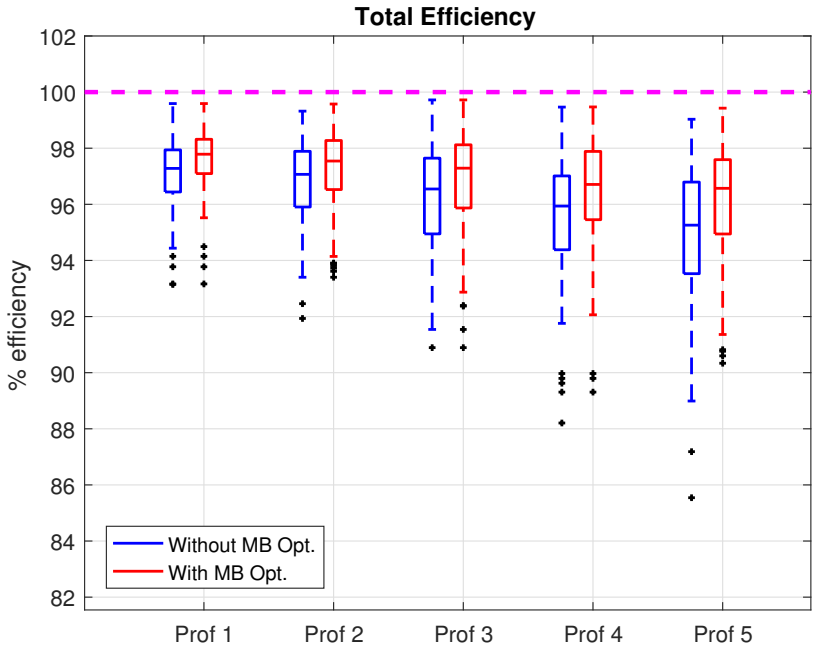


Figure 4.1: Total efficiency before (red) and after (blue) market-based optimization as the failures increase. Failures increase in occurrence frequency, duration, and variation across robot. Profile 1 is the same failure profile as used in Chapter 3.

efficiency is

$$c_{eff,rec}(t_m, \epsilon_{low}, \epsilon') = \frac{24 * 30}{t_{wl}} p_w t_m (\epsilon' - \epsilon_{low}) \tag{4.3}$$

where ϵ' is the efficiency after using a mobility solution on a ϵ_{low} case. The cost of the rail is more straight-forward, and is given as

$$c_{rail}(t_m) = p_r + p_{r,op} t_m \tag{4.4}$$

where p_r is the price of the rail (with installation) and $p_{r,op}$ is a monthly rail operation cost. Cost effectiveness of the rail can be determined if the condition

$$c_{eff,rec} > c_{rail} \tag{4.5}$$

is met.

4.4 Robot Placement and Mobility

The solution to the problem of decreased efficiency proposed here is to increase the overlap of the robot reach in order to increase task-sharing capabilities. To this end, the two robot base placement paradigms considered here are smart placement of stationary robot bases and robot base mobility. These methods are described in this section. Since robot base location is used here as a means to increase task-sharing (or re-scheduling), these methods are naturally suited for use with the market-based schedule optimization algorithm from Chapter 3. For the methods outlined below, the algorithm from Section **3.3.2** is modified to accept robot base locations as an input from which the ability matrix **A** is constructed.

4.4.1 Stationary Robot Base Placement

The stationary base placement paradigm is outlined here. We consider this paradigm because it offers some control over the overlap of robot reach without the cost that is associated with implementing a mobility system. The strategy proposed here is to find the locations

of the robot bases that result in maximum work balanceability by doing an exhaustive base placement search, as shown in Figure 4.2. For a given case, the best base placements are determined by running the scheduling method and optimization algorithm for all combinations of robot base placements, and finding the base placements that result in the highest efficiency. This requires the optimization algorithm to be run m^{n_x} times, where m is the number of robots, and n_x is the number of placement locations considered for each robot, assuming all robots have the same number of candidate base placements. Although the market-based algorithm is computationally inexpensive, a solution-space search can still be computationally taxing if the number of robots and potential base locations become large. However, an exhaustive placement location search provides a simple way to identify the upper bound of potential benefit in recovered MRS efficiency for this method. If the upper bound is not sufficiently high, there is no need to consider more efficient placement search strategies. Note that placement is done *prior* to schedule generation and execution.

4.4.2 Mobile Robot Bases

The mobile robot paradigm is outlined here. Since base mobility can produce the same base placement effect as stationary base placement (changing the base location before schedule generation and execution), the proposed base mobility strategy considers base mobility during schedule execution. This allows robots to relocate their bases in between their scheduled tasks. Task sharing during the execution of the schedule implies that the optimization algorithm will be run again during the execution of the schedule. The proposed strategy allows the first-to-finish robot $r_{i'}$ to move its base during the re-optimization. From the market-based formulation in Chapter 3, $i' = \text{argmin}_i(u_i)$, where u_i is the i-th's robot utility, a measure of its schedule's makespan (time to finish it's assigned tasks).

4.4.3 Splitting Task Cities

To run the optimization after $r_{i'}$ completes its schedule, the market-based algorithm initializes a new market (utility, price, and margin matrices) with the remainders of the schedules

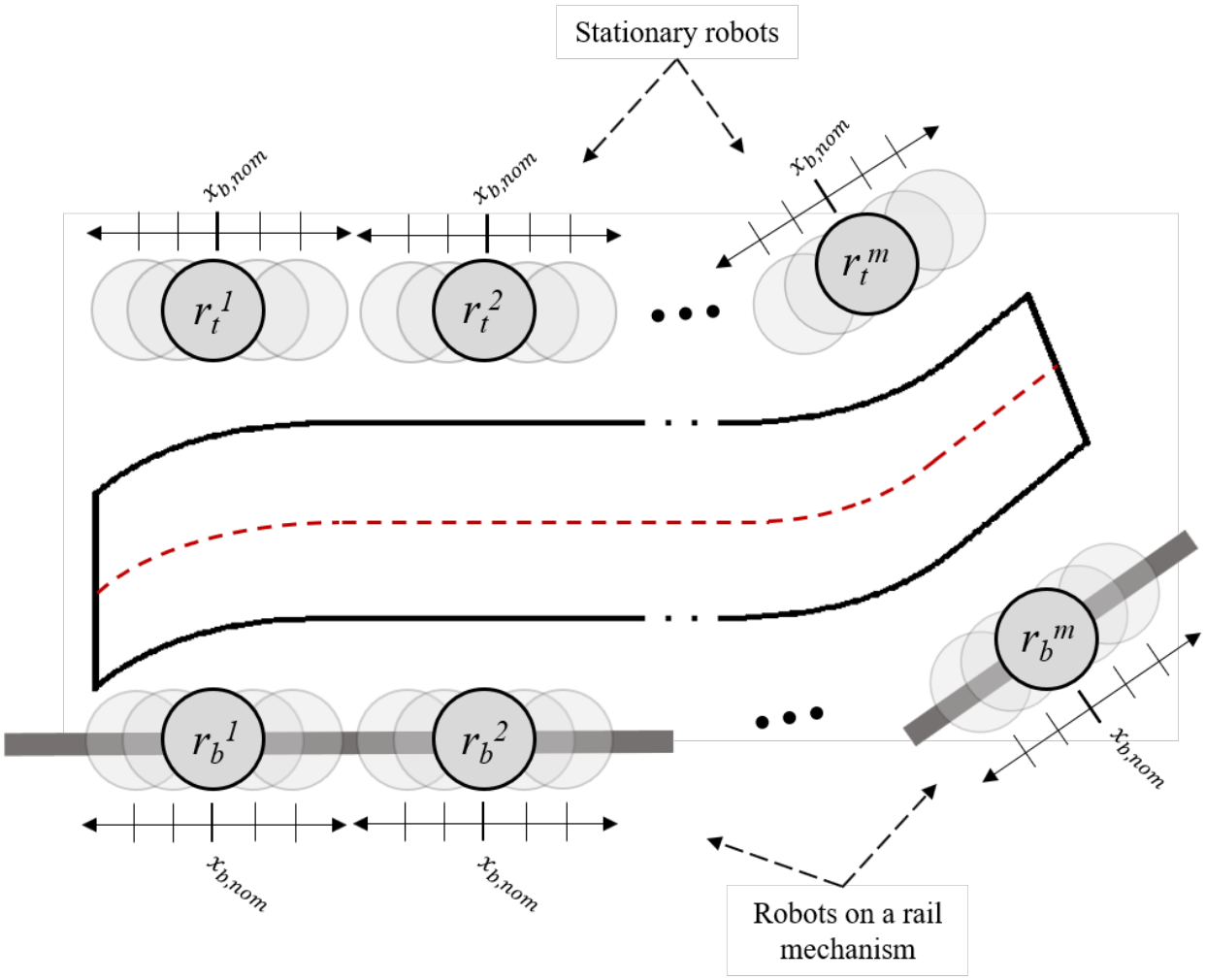


Figure 4.2: Visualization of robot base placement, whether of stationary robots (top robots) or of robots on a rail mechanism (bottom robots). A MRS can have either all stationary robots, have a rail mechanism on both sides, or have a mixed setup as shown. For the stationary robot paradigm, a base placement search is done prior to schedule generation and execution in order to determine upper bounds on the potential benefit. For mobile robots, robot base relocation is considered for the first to finish robot, along with schedule re-optimization, during the execution of the schedule.

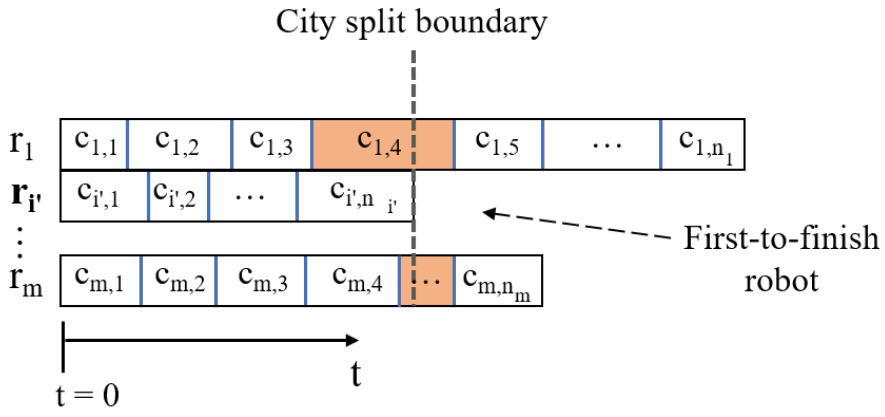


Figure 4.3: Visualization of task city splitting (CitySplit) for mid-schedule re-optimization. The tasks highlighted in orange are cut at the city split boundary to form new tasks. Then, the market-based algorithm is initialized with the new schedules that are to the right of the city split boundary for re-optimization.

of all the other robots, where the schedule of the first-to-finish robot is empty (i.e., $\mathcal{S}_{i'} = \{\}$. Since there is no guarantee that the all the robots will be between tasks when $r_{i'}$ completes its schedule, the cities are cut at the boundary in time where the $r_{i'}$'s schedule ends, as shown in Figure 4.3. This creates $m - 1$ new cities which contain the tasks that remained in the cities the robots $r_{i \neq i'}$ were servicing when $r_{i'}$ completed its schedule. This method is henceforth referred to as CitySplit. The optimization is then done with these new schedules. This process can be repeated with the next first-to-finish robot, and so on.

For the re-optimization, new base placements are considered for $r_{i'}$. However, the creation of new cities alone can aid in the re-optimizing of the schedule, and therefore it necessary to see the effects of the CitySplit with no base mobility in order to see if the benefit from the re-optimization is a result of CitySplit or base mobility.

4.5 Simulation and Results

To show the benefit of all the base placement approaches, we first compare the efficiencies from using each approach (stationary placement, CitySplit without mobility, CitySplit with mobility) across all the failure profiles, and then compare the cost of efficiency to the cost of mobility. An example case is then shown to illustrate the benefit of robot mobility. All methods were tested on the aircraft wing hole drilling problem used in Chapter 3.

4.5.1 Experiment Cases

We tested the methods using the failure profiles given in Table 4.1 and all the condition of assembly (COA) scenarios from Chapter 3. Results from the first COA are given in this chapter, and the results from the rest of the COA are given in Appendix C. Since the mid-schedule optimization and robot mobility require the market-based algorithm, the methods were used for the leftover scheduling, although the nominal schedule was still required for the generation of failures and for calculating efficiency. For each experimental run, five resulting efficiencies are tracked: (i) baseline (), (ii) stationary base placement search, (iii) CitySplit with no robot mobility (robot base location = x_{nom}), (iv) CitySplit with robot base mobility allowed in robots on only one side of wing to simulate the effects of having one rail (robots 2 and 4), and (v) CitySplit with mobility in all robots. 300 cases were generated from each failure profile, and the methods were tested on the bottom (lowest efficiency) 15% of the cases, 45 cases for each failure profile. In order to quantify the added benefit of mobility on top of CitySplit, from the 45 cases for each profile, we also tracked the cases that benefited from having mobility on top of CitySplit (i.e., the cases where there was a difference between efficiency in (iii) and (v)). Since the robots can only be re-positioned along the axial direction, we considered candidate placements only along the x-direction. We used the placements of the robots from Chapter 3 as our nominal placements, $x_{b,nom}$, and incremented the placements (for the 15% scaled wing from Figure 3.8), by 1.5 inches. The candidate placements were generated with five increments in either direction from the

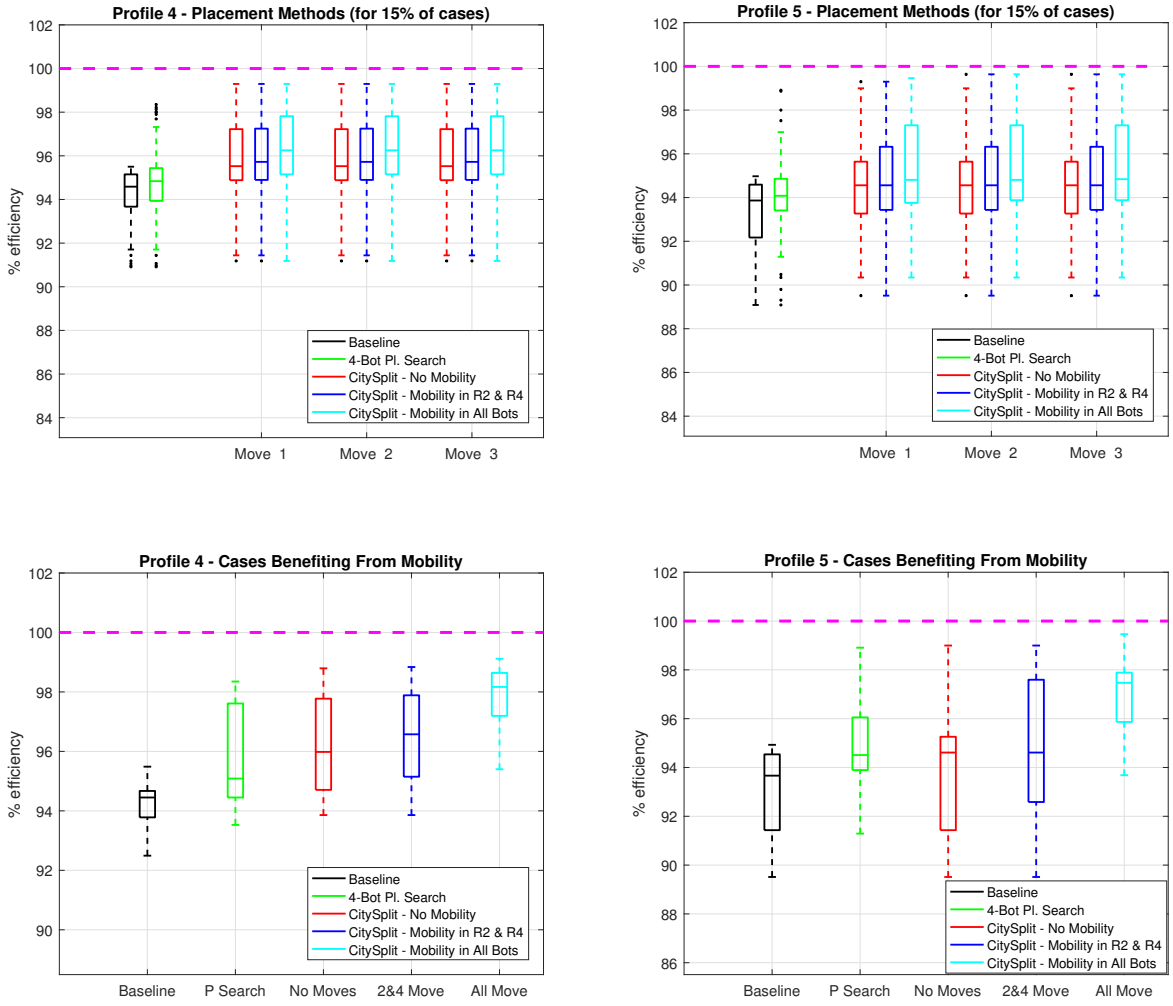
nominal placement.

4.5.2 Comparing the Base Placement Paradigms

We first compare the resulting efficiencies of all the approaches to the baseline efficiency. The top plot in Figure 4.4 shows the efficiencies of all the approaches for failure Profile 5, and also the efficiencies of repeating the CitySplit process for multiple base moves. Note that CitySplit with mobility outperforms stationary base placement by X% on average. The bottom plot shows the same results but only for the cases that benefited from mobility on top of CitySplit (i.e., cases where there was a difference between CitySplit without mobility and with mobility).

4.5.3 Effects of City Splitting and Mobility

We compare the efficiency between CitySplit with no mobility and CitySplit with all mobile robots to show the added benefit of mobility on top of the CitySplit. This result is the basis for determining the cost-effectiveness of mobility. Figure 4.5 shows the increase in efficiency over the baseline for CitySplit with no mobility and CitySplit with mobility. Shown are only the cases that benefited from the added mobility, across all Failure profiles. Out of 45 cases tested, 12 cases on average benefited from having added mobility over just CitySplit. The number of cases benefiting from mobility are taken into account for cost calculation. The benefit of added mobility is calculated by taking the difference between the efficiency from CitySplit with no mobility and CitySplit with mobility. Figure 4.5 shows this benefit for each failure profile. Note that as failures increase, the benefit associated with allowing robot base mobility also increases.



(a) Profile 4 (b) Profile 5

Figure 4.4: Efficiency results from (a) failure Profile 4 and (b) failure Profile 5. The bottom (lowest efficiency) 15% of cases are selected from failure Profile 5, and the various robot base placement methods are tested. The baseline efficiency is in black. Iterated base mobility are tested (top), and the cases out of the bottom 15% that benefited from base mobility (single mobility) are compared for each method (bottom).

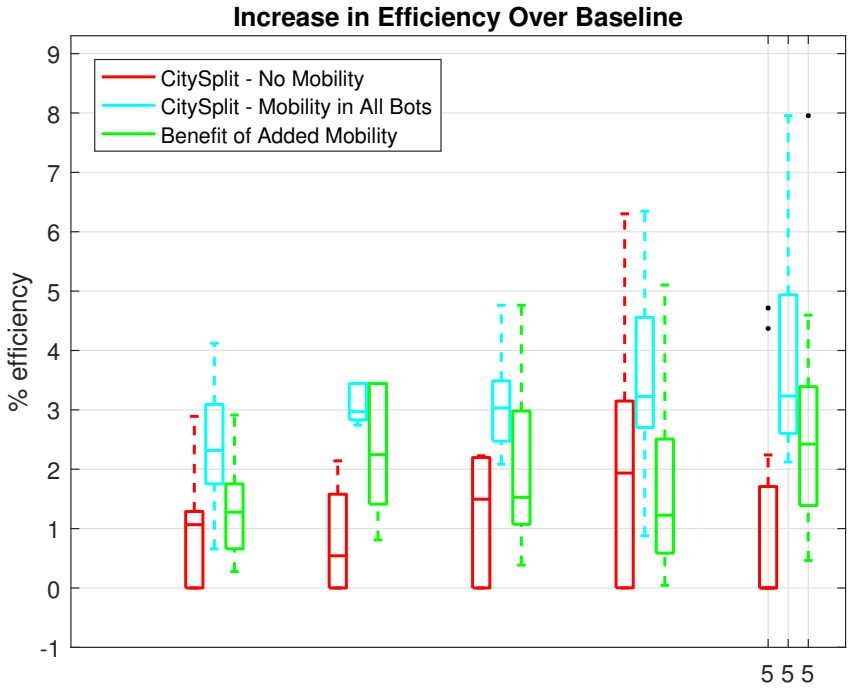


Figure 4.5: Shown is the increase in efficiency over baseline of the CitySplit without mobility (red) and CitySplit with mobility (cyan) methods, and the difference between the two (green) for all the failure profiles. Only the cases where the efficiency increase from added mobility are shown here.

4.5.4 Cost Balance

Since not every case benefits from either CitySplit or robot mobility, Equation 4.3 is modified here to account for the number of cases that benefited from robot mobility. This gives

$$c_{eff,rec}(t_m, \epsilon_{low}, \epsilon') = \frac{24 * 30}{t_{wl}} p_w t_m (\epsilon' - \epsilon_{low}) \frac{n_{CABM}}{n_{cases}} \tag{4.6}$$

where n_{CABM} is the number of cases affected by mobility, an n_{cases} is the total number of cases. This equation is used to calculate the cost of the recovered efficiency for all the profiles, using the efficiency of CitySplit with mobility as ϵ' and the efficiency of CitySplit without mobility as ϵ_{low}. We run an example scenario with the values: the price of the wing is $p_w = \$30,000,000$, the wing lead time is $t_{wl} = 8$ hrs, and the time to consider cost-effectiveness $t_m = 12$ months. For the rail cost in Equation 4.4, we take the rail cost $p_r = \$2,000,000$, and the monthly operating cost $p_{r,op} = \$15,000$ per month. Using these values, the cost-effectiveness of robot mobility can be evaluated using Equation 4.5. The cost is calculated for all failure profiles, and the results are shown in Figure 4.6.

4.5.5 An Example Case

Here, an example case of how the schedule efficiency increases with robot base mobility is shown in Figure 4.7. Note the increase in efficiency as the mobility of Robot 3's base allows it to reach and take on some of Robot 1's tasks. In addition, because the CitySplit method generates higher resolution task cities, other task trading is enabled.

4.5.6 Discussion

Stationary vs. mobile robots:

Figure 4.4 shows the comparison between MRS efficiency from using stationary robot base placement or mobile robots. In general, robot mobility outperforms stationary robots by 1% for all cases and by 3% for cases affected by mobility, on average. Smart placement of

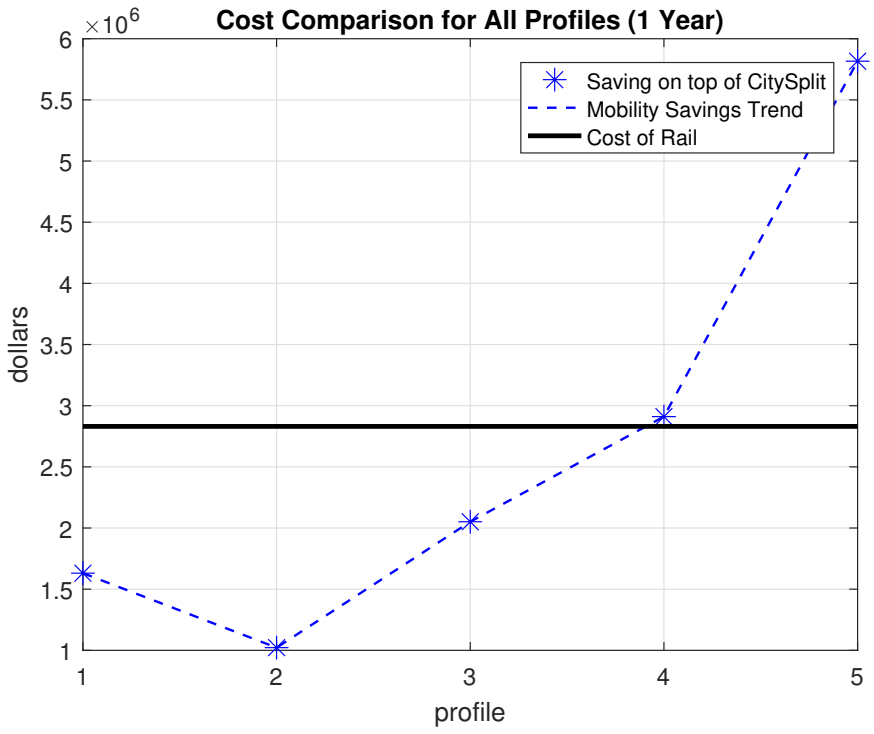
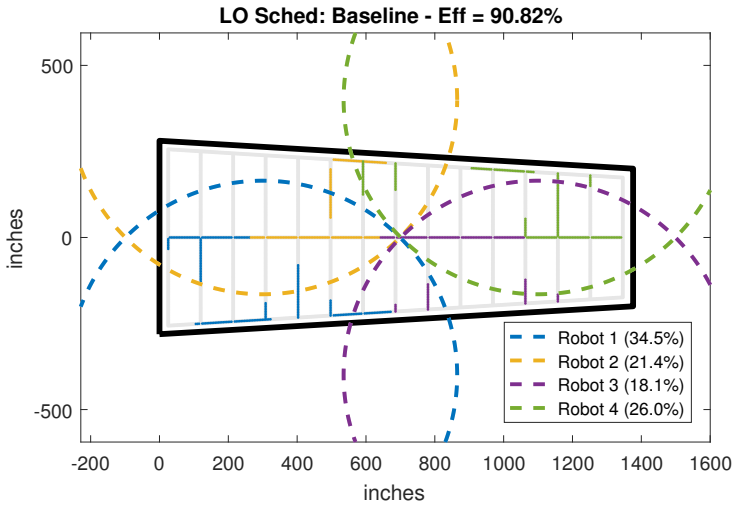
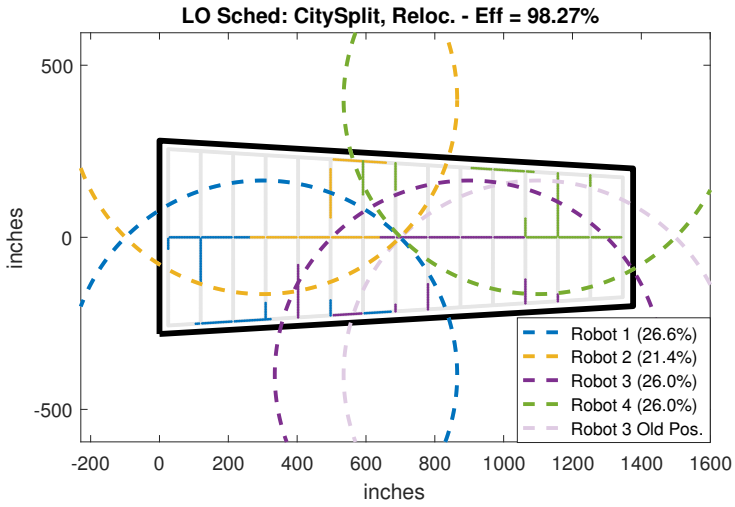


Figure 4.6: Comparing the cost of the recovered efficiency from robot mobility (savings on top of CitySplit) to the cost of a mobility system, in this case a rail mechanism. Note that as the failures increase, the benefit of robot base mobility increases, and exceeds the cost of mobility at a certain point.



(a) Baseline



(b) CitySplit with Mobility

Figure 4.7: A plot showing task assignment and robot reach on an example case of the cases that benefited from robot motion during leftover schedule execution. Shown is the task assignment and robot reach of the baseline leftover schedule (top) and the task assignment and robot reach after robot base mobility (bottom).

stationary robots appears to be a good strategy (1% improvement over baseline, on average, and over 5% on some cases) in the absence of a robot mobility system. However, placement was searched for in every case, and a single placement for all the cases must be considered, since moving the robots around for every manufactured wing is not possible. Future work will do a placement search using single candidate placements for all cases to find the potential benefit of leaving the robots in a single place for all cases. Additionally, not all cases benefit from mobility. Another future work will focus on determining the potential benefit of mobility for any case based on the problem characteristics.

Effects of splitting task cities:

The efficiency recovered from using CitySplit alone is not negligible (1.5% on average), as seen in Figure 4.5. This means that using CitySplit offers an advantage even if there is no robot mobility system. It can be used as an extra layer of optimization in stationary robot MRS. The added benefit of mobility, however, is 2% on average, being most beneficial for profiles 4 and 5, being even greater than the increase in efficiency from CitySplit alone.

Cost-effectiveness of mobility:

From Figure 4.6, it can be seen that for a given scenario, a cost analysis can be performed to determine whether mobility is cost-effective. Depending on the failures that a manufacturer is anticipating, and on the cost of the manufactured part and mobility system, a plot like shown in Figure 4.6 can be used to make a decision on whether or not a cell should have robot mobility.

4.6 Conclusion

This chapter explores robot base placement and mobility as a cost-effective means of recovering efficiency lost due to increased robot failures. Stationary robot base placement as well as mobile robot bases are considered in order to determine whether mobility is cost-effective. We found that, for the wing application from Chapter 3, as the failures increase,

robot mobility is a cost-effective means to recover lost efficiency.

Chapter 5

DECISION-BASED HEURISTICS FOR MINIMIZING TOOL SWITCHING

This chapter addresses the third research question (**R3**): *How to minimize tool changes in the scheduling in order to maintain MRS efficiency?*

5.1 Introduction

In this chapter, we first present a heuristic-based approach to determine maximally efficient task precedence relations. Our approach builds on existing auction-based algorithms for conflict-free scheduling. Subsequently, we provide a robust data-based strategy to learn a policy for automatically selecting the appropriate scheduling heuristics to minimize tool switching and maximize efficiency for any problem case. We then present promising results from testing our methods on a wing assembly problem involving the drilling of approximately 2000 holes with varying diameters by four robotic arms. Therefore, our main contributions are an auction scheduling algorithm that employs decision-based scheduling heuristics to encode soft precedence constraints, and a data-driven framework for generating robust heuristic selection policies that is useful for a variety of multi-robot task scheduling problems.

5.2 Related Work

There is much resemblance between our problem and the job sequencing and tool switching problem (SSP) [56]. The SSP deals with sequencing jobs that each require a set of tools on a machine that is able to hold only a limited amount of tools at a time. In order to perform some of the jobs, the tools must be switched on the machine, and the sequencing problem aims to minimize the amount of tool switches [57]. Like MRTA problems, the SSP is NP-hard

and current solutions employ heuristics coupled with integer programming, especially mixed integer linear programs (MILP) [68–70]. Methods that have been able to solve the SSP in polynomial time are restricted to single machine SSPs [58], and are not related to our work. Our problem is more related to the multi-machine SSP, for which heuristics and integer programming are also widely used [57, 71, 72]. However, since SSPs deal with stationary machines, their solutions don't take into account for the cross-schedule dependencies that arise from potential robot-robot collisions, as is the case in this research, and therefore aren't suited to solved our problem.

We approach our problem as an MRTA problem with precedence constraints. Our problem fits into the single-task robot, single-robot task, time extended assignment with synchronization and precedence constraints (ST-SR-TA:SP) category of problems per Nunes et. al.'s MRTA-TOC (Temporal and Ordering Constraints) taxonomy [7]. For these problems, auction-based methods are widely used due to their ease of implementation and robust execution [66]. In auction-based algorithms, robots are agents that bid on tasks per some objective, and the tasks are allocated to the highest bidding agent. To address precedence constraints, in [54], tasks were divided into disjoint subsets per their precedence, and the constraints were imposed on the subsets. An iterated auction algorithm with prioritization (pIA) that leverages these subsets, was proposed in [55]. During the auction, tasks were assigned priorities on the basis of longest successive precedence chains, in order to allocate tasks that are more critical first. These methods, however, aren't readily suited to deal with our problem because they leverage hard precedence constraints, where the ordering of the tasks is known before hand and cannot be changed, which are absent in our problem. Our work, then, adapts the pIA to handle soft precedence constraints using a heuristic approach. Additionally, we employ a schedule placement technique similar to the one used in [15], where the best placements of the task in the bidder's schedule helped inform the bids and increased solution quality. Therefore, our work is towards extending the use of auction-based methods to deal with a class of MRTA problems with soft precedence constraints.

5.3 Problem Formulation

The main challenge presents itself in the absence of hard precedence relations between the tasks. The main problem, then, stated generally, is to determine the task precedence relations such that schedule efficiency is maximized and tool switches are minimized. To approach this challenge, we first formulate the problem as posed in [55].

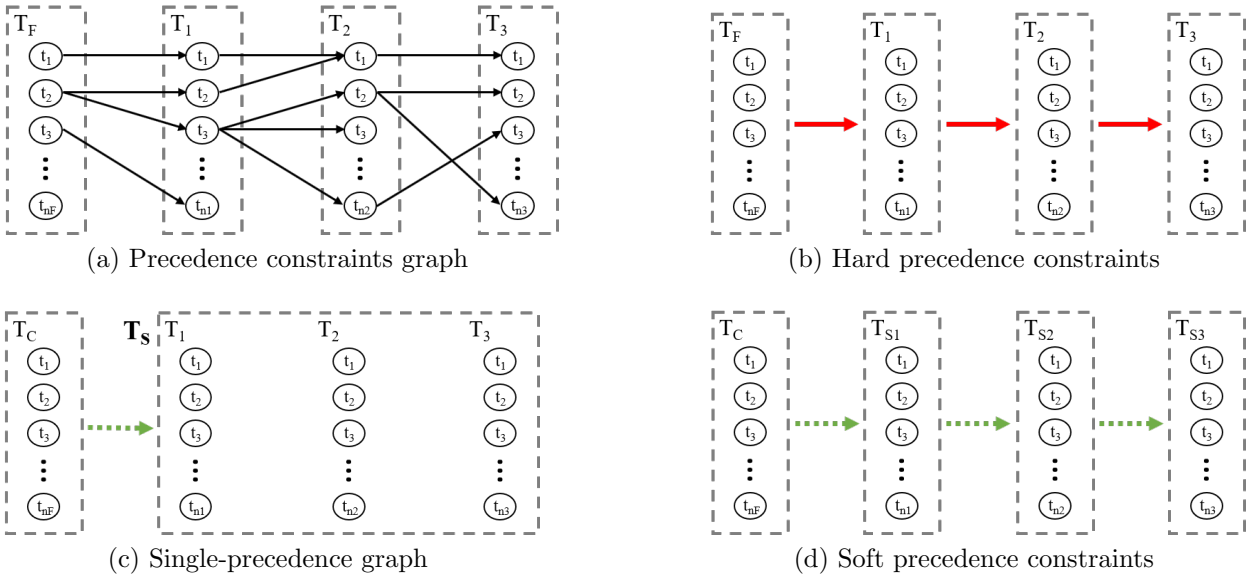
5.3.1 Precedence Layers

In the hard precedence constraint MRTA problem, we are given $\mathcal{T}$, a set of n_t tasks that have task precedence relations defined by a directed graph $\mathcal{G}$, as shown in Figure 5.1a, and m robots $r_j \in \mathcal{R}$ that the tasks are to be assigned to.

Per Luo et. al.'s formulation in [54], the tasks are arranged into subsets, or layers, with precedence being defined between the layers, as shown in Figure 5.1b. The tasks that have no antecedents are put into the *free* layer, T_f. The layers that have no antecedents are denoted as T_i, where $i \in \{1, \ldots, n_a\}$ indicates the number of antecedents any task in layer T_i has and n_a is the maximum number of antecedents than any task in $\mathcal{T}$ has.

In our problem, the only precedence chain that is given is that a robot should do tasks that do not require a tool change. Following a similar approach of dividing tasks into subset on the basis of precedence, we put all the tasks that correspond to the robot's current tool into layer T_c, and all other tasks, which would typically have their own layers, into a single layer T_s, as showing in Figure 5.1c. The precedence relation between these two layers is loose, meaning that not all the tasks in T_c have to be completed before any tasks in T_s can be done. Switching to a task in T_s is allowable, but incurs a penalty (the time cost of a tool change). We refer to this type of constraint a *soft* precedence constraint.

Similar to the formulation in [55], we separate all the tasks into layers by common tool that is required for the task. T_s here is split into $i \in \{1, \ldots, n_l\}$ layers T_{S_i}, where n_l is the number of tool types necessary to complete all the tasks in $\mathcal{T}$. T_c is then set to $T_{S_{i^*}}$ and $T_S = T_S \setminus T_{S_{i^*}}$, where i^* corresponds to the tool the robot starts with, determined by either



(a) Precedence constraints graph

(b) Hard precedence constraints

(c) Single-precedence graph

(d) Soft precedence constraints

Figure 5.1: (a) A hard precedence constraint, usually given in the form of a directed graph [55]. (b) The graph given in (a) can be arranged into layers with determined hard precedence relations, denoted by the red arrows, as in [54]. (c) A precedence graph with a single, loose antecedent layer. (d) Soft precedence constraint layers formed by grouping tasks of a common tool type. The green dashed arrows indicate that the layers have no set precedence relations.

it's current tool or the first tool it selects. We refer to T_c as the *current* layer. The precedence relations between T_c and T_{S_i} are also soft precedence constraints, and are equivalent for all i. When a task is to be executed in T_{S_i}, the bidding agent *jumps* from T_c to T_{S_i} (i.e., T_{S_i} becomes T_c, and vice versa), and a tool switch penalty is incurred. The incurred penalty is the same whether or not T_c is empty. The scheduling problem is illustrated in Figure 5.2.

5.3.2 Problem Statement

We set up the main problem by posing four decision questions that flow out of the soft precedence constraint formulation:

1. How to designate the initial layer T_c?

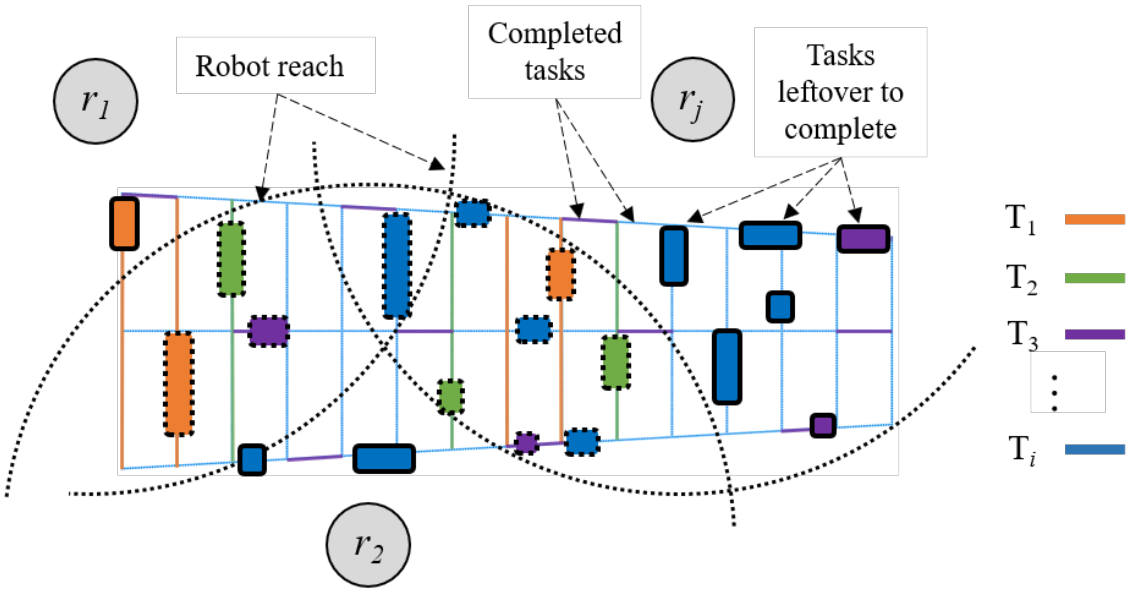


Figure 5.2: An example of a soft-precedence constrained, multi-robot task allocation problem. The task types are represented by the different colors, and the reach of the robots is denoted by the dashed arcs. The tasks that are initially completed according to a nominal schedule, as in [20], are shown by the orange, green, purple and blue lines, and the leftover tasks resulting from robot failures are outlined by the solid and dashed lines. Note that some of the tasks are in overlapping robot reach areas, shown in dashed outlines, whereas, the other tasks are in areas reachable by only one robot, as indicated with solid outlines.

2. How to assign task priority in T_c?

3. When should the bidding agent jump from T_c to $T_{S_{i^*}}$?

4. How to select the layer $T_{S_{i^*}}$ to jump to?

The main problem statement then, is to find the way, or heuristic, to answer these questions for a given set of tasks and robots that maximizes efficiency. The example scheduling problem shown in Figure 5.2 shows multiple task layers (denoted by color) that contain tasks with

varying robot reachability, and illustrates how the questions guide the scheduling, as different ways to answer the questions result in different tasks schedules.

We define efficiency as the ratio between the actual schedule makespan t_{act}, and an idealized schedule makespan t_{opt}, as in [20]. The actual makespan is given as the time the last-to finish robot takes to complete all its tasks, expressed as

$$t_{act} = \max_{j}(t_{tasks,j} + t_{ts}n_{ts,j}) \tag{5.1}$$

where t_{tasks_j} is the time it takes the j-th robot to complete its assigned tasks, t_{ts} is the tool switch time, $n_{ts,j}$ is the number of tool switches the j-th robot has to make, and $j \in \{1, \ldots, m\}$. The idealized makespan is given as the sum of the completion times of all the tasks and an ideal minimum time required for tool switches, split evenly amongst the m robots, is expressed as

$$t_{min} = \frac{\sum_{k=1}^{n_t} t_{task,k} + t_{ts}n_{ts}^*}{m} \tag{5.2}$$

where $t_{task,k}$ is the time to complete the k-th task, and n_{ts}^* is an idealized number of tool changes. We determine n_{ts}^* by finding the difference between the number of required tool types n_l and the number of different tools the robots currently have installed n_c. This gives the efficiency expression as

$$\epsilon = \frac{\sum_{k=1}^{n_t} t_{task,k} + t_{ts}n_{ts}^*}{m \max_j(t_{tasks,j} + t_{ts}n_{ts,j})} \tag{5.3}$$

Note that even though comparing the actual makespan to an idealized makespan is more restrictive than comparing to an optimal makespan, since the n_{ts}^* does not take into account robot task reachability, this measure of efficiency provides a valid way of comparing the relative performances of various scheduling methods.

5.4 Iterated-Auction-Based Policy Learning

To address the main problem, we first adapt the iterated auction with prioritization (pIA) algorithm from [55] to handle the questions posed in Section 3 in a heuristic manner. We then develop a data-driven approach to generate a policy to select the appropriate way to answer the questions from Section 3 for any given problem case based on problem characteristics.

5.4.1 Soft Precedence Iterated Auctions (SPIA)

We present an iterated auction algorithm that is able to handle soft precedence constraints, adapted from the iterated auction with prioritization (pIA) algorithm presented in [55]. We treat each of the four questions in Section 3 as a decision variable d_a, where $a \in \{1, 2, 3, 4\}$. We set up discrete choice categories $d_{a,b}$ in each decision variable, where $b \in \{1, \ldots, n_a\}$ and n_a is the number of categories in the a-th decision variable. Every category $d_{a,b}$ provides a heuristic way for answering the a-th question, and only a single category can be selected for each decision variable. The set of selected categories b_a^* for all a then generates a scheduling *heuristic*, and is denoted by $[b_1^*, b_2^*, b_3^*, b_4^*]$.

In our market, we have a set of m robots $r_j \in \mathcal{R}$, and n_t task layers, T_{S_i}. We denote global layers as $T_{G_i} = T_{S_i}$ and local layers as $T_{i,j} = A_j(T_{S_i})$, where A_j returns the set of all the tasks in T_{S_i} that the j-th robot is able to reach, and $j \in \{1, \ldots, n_r\}$. Note that the same task can appear in a local layer of multiple robots. Every robot then can only bids on tasks in it's own local layers. When a task is assigned, it is removed from every layer it appears in, local and global.

To set up the auction, we initialize a set of m^* buyers $\mathcal{B}$ as the set of all robots that have any non-empty $T_{i,j}$ (i.e., robots that have any tasks that they can reach). For each $r_j \in \mathcal{B}$, a local current layer $T_{c,j} = T_{i^*,j}$ is initialized, where i^* is the j-th robot's local layer that is selected to be its current layer. $T_{i^*,j}$ must be non-empty. The basis on which i^* is selected is determined by the category selected in d_1.

The auction entails the robots taking turns in bidding on and buying tasks in their

respective current layers. For each r_j, a schedule S_j, an order set, is initialized as an empty set. A bidding round occurs in the following way: a robot r_{j^*} is selected to be the bidder, and calculates bids on all the tasks in T_{c,j^*}. For every task $t_k \in T_{c,j^*}$, n_l bids are calculated for every placement (insertion point) $l \in \{1, \ldots, n_l\}$ in r_{j^*}'s schedule S_{j^*}. For every potential placement l of t_k in S_{j^*}, a schedule $S_{j^*}^p$ for is proposed. Using the proposed schedule $S_{j^*}^p$ and the schedules of the non-bidding robots $S_{j \neq j^*}$, a minimum pairwise distance $w_{k,l}$ between all the robots is found. If $w_{k,l} > \rho * d_{ee}$, the bid, then, is $b_{k,l} = \beta_k * w_{k,l}$, otherwise $b_{k,l} = 0$, where β_k is a biasing factor we use to assign a priority to task t_k, and is determined by the selected category in d_2. $\rho * d_{ee}$ is the spatial collision constraint, where d_{ee} is the diameter of the robot's end effector, and ρ is a safety factor.

The number of placements n_l is pruned by only considering (i) placements adjacent to those with tasks of a common tool type, and (ii) the last placement in S_{j^*}. We denote these placements as $\hat{l}$. The highest bid determines the purchased task t_{k^*} and it's placement $\hat{l}^*$ in S_{j^*} by taking $\text{argmax}_{k,\hat{l}}(b_{k,\hat{l}})$. In this way, bids that maximize the physical separation of the robots are preferred. Once the task has been purchased, it is removed from $T_{c,j}$, and from any layer of any other robot that contains it, as well as from the global layers. At this point the next bidder is selected, and in this sense, the auction is iterated.

If no bid is acceptable (i.e., $\max(b_{k,\hat{l}}) = 0$), the bidder will decide on whether to *jump* from T_{c,j^*} to another local layer, or to append a wait gap task t_{wg} to the schedule, no longer that δ, which is based upon the selected category in d_3. Once the conditions set by d_3 are met, or the current T_{c,j^*} is empty, the bidder will jump to a new layer, and append a tool change task t_{tc} to the end of it's schedule. The selection of T_{i*,j^*} as the bidder's next current layer is done as directed by the category selected is d_4.

The current bidder r_{j^*} is selected from the buyer set $\mathcal{B}$ as the robot that has the shortest makespan (i.e., $j^* = \text{argmin}_j(makespan(S_j))$), where $makespan(S_j)$ returns the time it takes to perform all the tasks in S_j. For the first m^* rounds of bidding, when the robots are bidding on their first tasks, the bidders are selected on the basis of which robot has the least tasks in their current layer (i.e., $j^* = \text{argmin}_j(makespan(T_{c,j}))$). This ensures that the robots that

have a narrower selection in their current layers are able to go first, since non-empty schedules of other robots can restrict task selection because of the spatial collision constraint. A robot leaves the buyer set $\mathcal{B}$ when all its local layers are empty, and the auction is done when all the global layers are empty. The pseudocode for this algorithm is given in **Algorithm 2**.

Remark 1: The categories in d_3 determine a maximum allowable wait gap that the bidder can append to its schedule before jumping to allow conflicting robots to bid on other tasks and eliminate the conflict. The execution time of an appended t_{wg} is set to be only larger than the difference between the makespans of bidder's schedule and the shortest schedule of the conflicting robots. The execution time of t_{wg} is increased in this manner to allow for other other robots to move on to other tasks to relieve conflict to the threshold set by d_3, at which point t_{wg} is removed from the end the bidder's schedule.

Remark 2: We make the bidding robot append a mandatory wait gap if there are no other non-empty layers it can jump to. This prevents deadlock between robots vying for task in close proximity towards the end of the auction.

5.4.2 Learning Scheduling Policy

The combinations of the categories in the decision variables result in a variety of decision paths, which are essentially scheduling heuristics. Since different heuristics perform differently for various case problems, the goal of the scheduling policy is to determine the best heuristic to employ for any given case. The aim, then, is to use differentiating characteristics inherent to the cases to determine the best performing heuristic for a given case.

For a given problem case p_i, we have characteristic variables
$C_i = [c_{i,1}, \ldots, c_{i,j}, \ldots, c_{i,n_C}]$, and schedule efficiencies $\epsilon_{i,k}$ resulting from using heuristics M_k, where $k \in \{1, \ldots, n_M\}$. We seek models A_k

$$\hat{\epsilon}_{i,k} = A_k(C_i) \tag{5.4}$$

such that $\hat{\epsilon}_{i,k} = \epsilon_{i,k}$ for all k. The best heuristic M_{i,k^*} to select for any given case i, then,

can be expressed as

$$k^* = \underset{k}{\text{argmax}}(\hat{\epsilon}_{i,k}) \tag{5.5}$$

In practice, however, fitting A_k models is difficult, and having many characteristic variables can lend itself to overfitting. We use a linear regression model instead to fit the $M_{i,k}$ efficiency data to the C_i:

$$\epsilon_{i,j} = \sum C_i \alpha \tag{5.6}$$

To determine the correlation between $\epsilon_{i,k}$ and elements of C_i, and to avoid overfitting, we seek to find the dominant modes $c_{i,j^*} \subset C_i$ of the model by eliminating characteristic variables that don't contribute to the fit. We use a readily available least squares fit algorithm, originally tailored for Sparse Identification of Non-linear Dynamics (SINDy) from [73], to fit the data and determine the dominant modes. In SINDy, the fit sparsity is controlled by parameter λ, such that any elements of $\alpha_j < \lambda$ are set to zero, and the model is fit again with only the non-zero coefficients, α_j^*. The dominant modes j^* are determined by increasing λ to the point that fit quality starts to markedly decrease. We denote the set of dominant modes as $\hat{C}_i$. The collection of $\hat{C}_i$ for a set of training cases $i \in \{1, \ldots, n_{train}\}$ is separated into n_l clusters D_l in the dominant modes, where $l << n_{train}$. The clustering is done per the efficiency of the highest performing heuristic in the cluster.

From this, we give a data-driven, cluster-based policy $\pi(l^*)$ to determine the best heuristic M_{i',k^*} to use for any test problem case $p_{i'}$, where $\hat{C}_{i'}$ classifies i' to the l^*-th cluster. The policy is expressed as

$$\pi(l^*) = \underset{k}{\text{argmax}}((\bar{\epsilon}_{i^*,k})) \tag{5.7}$$

where $\bar{\epsilon}_{i,k}$ is the average efficiency resulting from heuristic k across i^* problem cases, i^* are the cases in the l^*-th cluster. The policy returns the best heuristic to use for case i' based upon the cluster i' belongs to.

5.5 *Experimental Results*

To demonstrate the performance of our approach, we present computational results for the example wing assembly problem used in [20]. We first show the performance of our heuristic-based auction method by comparing it to the performance of the baseline partition-based scheduling method in [20]. Then we show the performance of our method selection policy by comparing to a greedy method selection approach.

5.5.1 Example Wing Problem

We consider the problem of drilling rows of holes in an aircraft wing box by four robots positioned on either side of the wing. The aircraft wing has 15 ribs and three spars defining rows of evenly spaced holes to drilled, as in [20]. Four different tool sizes were assigned to the holes, 7/16 in., 3/8 in., 5/16 in. and 1/4 in., and are shown in orange, green, purple and blue colors, respectively, in Figure 5.3. The tool change time is set to 420 seconds.

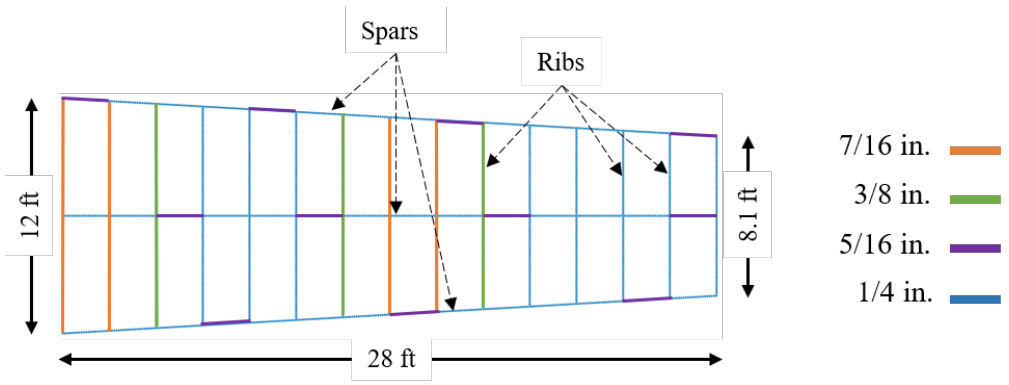


Figure 5.3: Mock three-spar, 15-rib wing used for testing our methods, from [20]. The various tool size requirements are denoted by the various colors: 7/16 in., 3/8 in., 5/16 in. and 1/4 in. are shown in orange, green, purple and blue, respectively. The time to change a tool to perform the various tasks is 420 s.

In order to evaluate performance on a variety of problems, we tested our methods on the

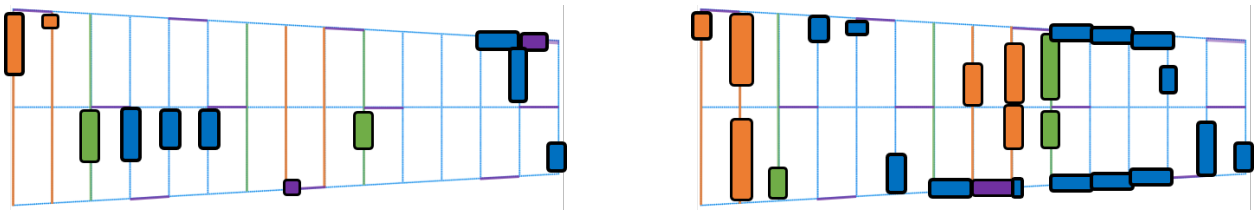
leftover scheduling problem from [20], where the tasks to be done are generated probabilistically. Per their method, a nominal schedule is generated using a partition-based approach, where robots skip work upon returning from failures to be done after the nominal schedule is complete. We use these leftover tasks, which come in the form of groups of holes (i.e., each task requires only one tool) and are probabilistically driven, to test our methods. The failures are generated by randomly sampling first failure occurrence time, failure recurrence time and failure duration time from the normal distributions given for two failure profiles in Table 5.1. The failures occur more frequently and last longer from failure profile 2 than they do from failure profile 1. The difference between the two profiles is illustrated in the examples in Figure 5.4. We test our methods on 1000 such failure cases. The spatial constraint parameters were $\rho = 1.5$ and $d_{ee} = 2$ft.

	Profile 1		**Profile 2**	
Parameter	μ	σ	μ	σ
First Occurrence	4123 s	1643 s	3480 s	2008 s
Recurrence	5498 s	1205 s	4640 s	1472 s
Repair Time	696 s	214 s	803 s	268 s

Table 5.1: Normal Distribution Parameters for Robot Failure Occurrence Time, Recurrence, and Repair Time

5.5.2 Experiments

We ran the algorithm and experiments in MATLAB R2016b on a Windows 10 computer with 16 GB of RAM and a Core i7-4870HQ processor. A physical system, built to a 15% scale to be representative of a feasible aircraft assembly cell, was used to simulate a wing-drilling process and validate the MATLAB simulations results. The cell was made of aluminum T-frame and plastic shields, with four ABB IRB-120 robotic arms situated around a miniature mock wing built to the dimensions shown in Figure 5.3. The robot controllers were collectively controlled over Ethernet UDP by supervisory cell control software implemented in C# using the ABB



(a) Profile 1 Example

(b) Profile 2 Example

Figure 5.4: Two example cases of the problem cases used to test the proposed methods. The tasks shown are the leftovers that results from robots skipping work when failures occur during the execution of a nominal schedule. Failures occur per the distributions given in Table 5.1. Note that the failures resulting from (b) Profile 2 are more substantial than from (a) Profile 1.

PC SDK 6.07. The supervisory software fed the hole targets to the robot controllers and simulated drilling by having the robots wait at the hole for a specified drill time, and was implemented on a separate machine with a Core-i5 6400 processor and 8 GB of RAM. The physical cell is shown in Figure 5.5.

Decision Variable Categories and Characteristic Variables

We use the categories for our decision variables as shown in Table 5.2. The categories are set to be tied to the geometric locations of the tasks (e.g., selecting the next layer that has tasks with the most overlapping reach), since the problem cases differ from each other by the geometric locations of the tasks. There are 32 possible combinations of the categories, so there are 32 total heuristics to test. We select seven variables to capture the problem characteristics, and these are given in Table 5.3.

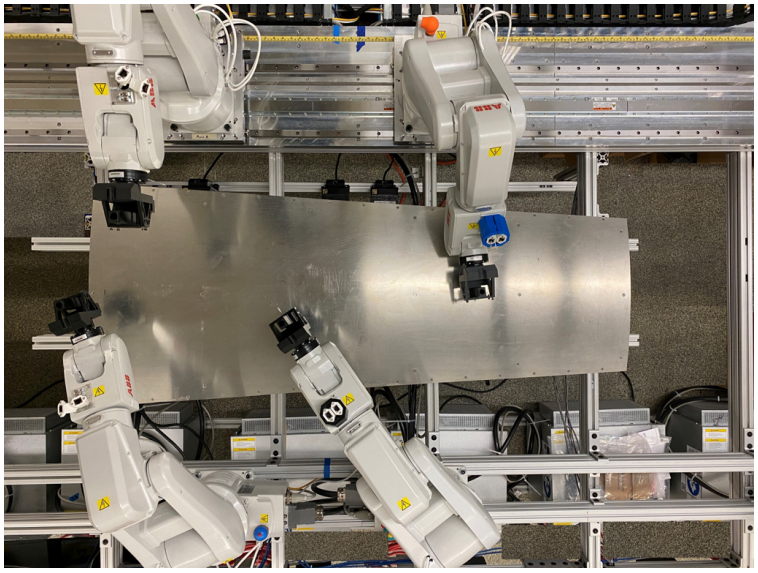


Figure 5.5: A physical multi-robot cell on which the algorithm was tested and the computational results were validated, built to a 15% reduced scale. A supervisory cell control software was used to communicate with the robot controllers and execute task schedules. The robots are simulating a wing drilling process.

Regression and Clustering

We set up the SINDy regression model as outlined in Section 4, and we tune the sparsity parameter λ to find the most dominant modes. We first normalize the characteristic variables before fitting the model. To generate the heuristic selection policy, we use two different approaches to cluster the characteristic variable space in order to be able to capture differences in the distribution of the cases in different dominant characteristic modes.

(I) We use axis-aligned partitioning to separate the variable space into four regions. In each partition, we determine the top (highest average efficiency) heuristic, and find the deviation of the top heuristic efficiency from the max possible (from all heuristic) efficiency for each case in the partition. We search the characteristic variable space for the the set of partitions that give the best efficiency performance (minimum total deviation from the globally

d_a	$d_{a,b}$	**Category Description**
1	1	T_c is layer corresponding to robot's current tool
	2	T_c is layer selected per d_4 criteria
2	1	tasks with minimal robot reach overlap ($\beta_k = 20$)
	2	tasks with maximal robot reach overlap ($\beta_k = 20$)
3	1	max wait gap of $\gamma * makespan(t_{tc})$: $\gamma = 0$
	2	$\gamma = 0.5$
	3	$\gamma = 1$
	4	$\gamma = 1.5$
4	1	layer with maximum shared reachability of tasks
	2	layer with minimum shared reachability of tasks

Table 5.2: Descriptions of the Categories in the Decision Variables

Char. Var.	**Description**
1	total duration of failures
2	average x-value of failures
3	spread in duration of various tool types
4	spread in reach overlap of various tool types
5	average reach overlap of all failures
6	average distance between all tasks
7	density of failures

Table 5.3: Summary of Characteristic Variables

optimal efficiency) by checking every possible partition set on a grid with an increment of 0.01 in the characteristic variable axes. The resulting policy is determined by selecting the top heuristic in the partitions that result in the lowest deviation.

(II) We use k-means clustering to separate the data. The resulting policy is determined by the top-performing heuristic from each cluster.

We use 800 randomly selected failure cases to train the SINDy model, and generate the policy. We test the policy with the remaining 200 failure cases. This is done for each failure profile separately, for a total of 2000 cases.

5.5.3 Results And Discussion

Performance of SPIA algorithm: We show that our heuristic-based auction method outperforms the baseline partition-based scheduling method by presenting efficiency results. Table 5.4 compares the efficiency obtained using our SPIA algorithm (across all 32 heuristics, as well as for the top heuristic, determined by highest mean over the failure cases) to the baseline efficiency obtained using the methods in [20], for all 1000 failure cases in each failure profile. We observe that the average efficiency obtained using SPIA is consistently higher [$F(2, 2997) = 1058.9, p = 0$] than the average baseline efficiency by 12.5% and 14.3% for failure profiles 1 and 2, respectively. This indicates a significant time savings for a wing drilling process that takes many hours to complete. If we rank order the 32 heuristics by their average resulting efficiency across all failure cases, and select the heuristic with the highest average efficiency (top heuristic), its efficiency is consistently higher [$F(1, 1998) = 156.8, p = 0$] than the average SPIA efficiency by 6.4% and 7.5% with for the two profiles, respectively. This shows that it is necessary to select the proper categories in the decision variables, and provides the motivation for a policy search. The top heuristic is $M_{28} = [2, 2, 2, 2]$ for both profiles. The top SPIA heuristic average efficiency is also higher than the baseline efficiency by 16.9% and 18.8%, for each profile respectively, which results in a significant time savings on wing manufacturing processes that take many hours to complete.

	Profile 1 Eff. (%)			**Profile 2 Eff. (%)**		
Method	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
Baseline	40.2	58.4	78.6	34.0	55.7	76.7
SPIA (Avg.)	52.1	70.9	86.1	47.4	70.0	84.6
SPIA (Top)	53.8	75.3	92.5	46.8	74.5	92.1

Table 5.4: Summary of Efficiency Results of Our Methods (Average and Top Heuristics) Compared to Baseline for Both Failure Profiles Across Training and Test Cases

Dominant modes from SINDy:

To show that our data-based policy approach is robust to characteristic differences in the

problem profiles, Table 5.5 shows the dominant modes and fit quality (R^2) as the regression sparsity parameter λ is increased. We observe that for the selected characteristic variables, SINDy is able to generate an acceptable fit using only two modes. From this, we observe: (i) that the fit between the characteristic variables and heuristic performances is acceptable, even with only two modes, and (ii) since the dominant modes are able to capture heuristic performance, we can use them to inform heuristic selection for a given case. Note that different sets of dominant modes are found for the two different problem profiles, indicating that the our data-based approach is robust to inherent differences in the two problem profiles.

	Profile 1		**Profile 2**	
Sparse (λ)	**Avg. R^2**	**Modes**	**Avg. R^2**	**Modes**
0	0.541	[1,2,3,4,5,6,7]	0.720	[1,2,3,4,5,6,7]
3	0.538	[1,2,5,6,7]	0.718	[1,2,5,6,7]
6	0.528	[2,5,6]	0.713	[1,2,5]
8	0.501	**[2,6]**	0.708	**[2,5]**

Table 5.5: Summary of Dominant Modes and Quality of Fit as Sparsity is Increased

Performance of heuristic selection policy: We show that our cluster-based heuristic selection policy performs well compared to a greedy selection policy by comparing the resulting efficiencies for the test problem cases. Figure 5.6 shows the resulting clustering technique that resulted in the best policy, and selected heuristics in the clusters that yield the lowest efficiency deviation for the selected training cases. For failure profile 1, we show only the k-means cluster-based policy in Figure 5.6a, and for failure profile 2, we show only the axis-aligned partition-based policy in Figure 5.6b. We observe that for a plurality of the training cases, the top heuristic (i.e., the heuristic with the highest average resulting efficiency) is chosen.

We show the performance of our heuristic selection policies in Table 5.6. We compare the average resulting efficiency (across the 200 randomly selected test cases in each failure profile) for: (i) greedily using only the single top heuristic for every test case, (ii) using the heuristics

Heuristic Selection Policy	**Profile 1 Eff. (%)**	**Profile 2 Eff. (%)**
Single Top Heuristic	74.95	73.94
Axis-Aligned Partitions	74.95	74.08
K-means Clustering	75.00	73.96
Max Possible Efficiency	78.69	77.90

Table 5.6: Summary of Resulting Efficiency from Heuristic Selection Policies for Only Test Cases

selected by the axis-aligned-partition-based policy, (iii) using the heuristic selected by the K-means cluster-based policy, and (iv) the globally maximum possible efficiency (i.e., selecting the best possible heuristic for each test case). We observe that the efficiency obtained using the heuristics selected by the axis-aligned partition-based and k-means cluster-based policies is consistently higher $[F(3, 796) = 8.13, p = 2.44 \times 10^{-5}]$ than that obtained by greedily selecting the single top heuristic by 0.14% and 0.02%, respectively, on an average for failure profile 2. For failure profile 1, the k-means cluster-based policy outperformed $[F(3, 796) = 14.69, p = 2.57 \times 10^{-9}]$ the top heuristic by 0.05%, and the partition-based policy performed equally with the top heuristic. This indicates that our policies perform at least as well as greedily selecting the top heuristic. We also observe that the efficiencies obtained using the heuristic selection policies are not significantly worse than the globally maximum efficiency, by 3.7% and 3.8% for on an average for profiles 1 and 2, respectively.

Extensions of data-driven policy learning: Our machine-learning approach is operationally robust, and is particularly beneficial when there are many more heuristics to select from. We considered 32 possible heuristics in our auction method, and exhaustively testing each heuristic would potentially be tractable only by virtue of the computational simplicity of the auction-based methods. Moreover, since our data-driven policy learning paradigm informs the selection of the appropriate scheduling heuristic (or method) based on the inherent characteristics of a particular problem, it is therefore generalized to be suitable to a variety of MRTA solution methods. Our approach is particularly beneficial for

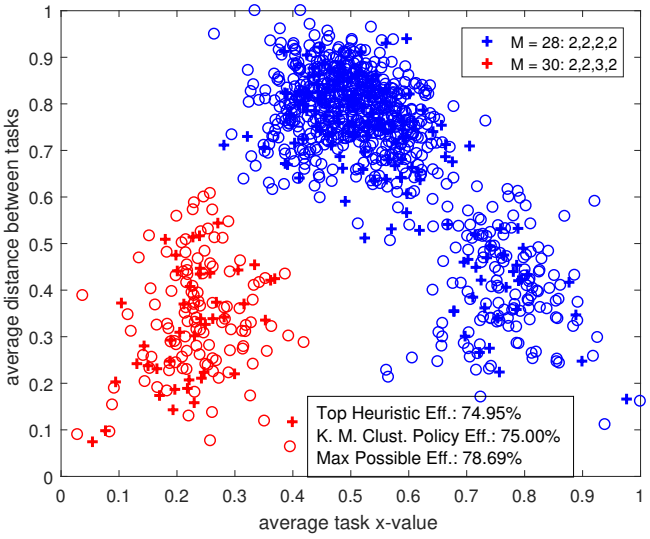
computationally time-consuming solutions, like genetic algorithms and traveling salesman problem solutions, where the time-savings from using an automated method selection are substantial.

5.6 Conclusions

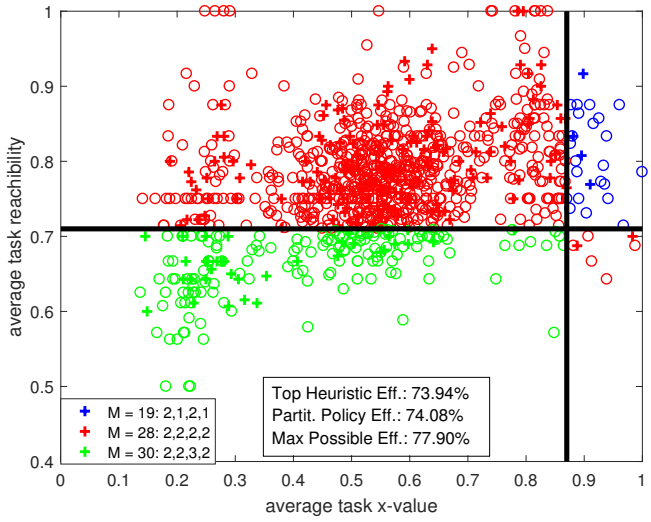
In this chapter, we present a data-based approach for learning how to best select a scheduling heuristic for multi-robot task allocation problems with soft precedence constraints. We adapt existing auction-based approaches to be suitable for soft precedence constraints by introducing a four decision variable heuristic into the auction process. We then present a machine-learning-based policy to select the best heuristic for any given problem case that takes advantage of known inherent problem characteristics. Results show that our adapted auction algorithm produces higher efficiency task schedules compared to a partition-based scheduler. Moreover, using heuristics selected by our learned policy yields higher efficiencies than greedily selecting a single best heuristic.

Algorithm 2 Iterated auction for multi-robot scheduling of task with soft precedence constraints.

Initialize global layers T_{G_i} from $\mathcal{T}$
for each $r_j \in \mathcal{R}$ **do**
    $S_j = \{\}$
    Initialize local layers $T_{i,j}$
Initialize buyer set $\mathcal{B}$
for each $r_j \in \mathcal{B}$ **do**
    Initialize $T_{c,j}$ per d_1
while T_{G_i} not empty for all i **do**
    $j^* = \text{argmin}_j(makespan(S_j))$ for $r_j \in \mathcal{B}$
    if T_{i,j^*} not empty for all i **then**
        if T_{c,j^*} not empty **then**
            for each task $t_k \in T_{c,j^*}$ **do**
                for each placement $\hat{l}$ in S_{j^*} **do**
                    $S^p_{j^*} = S_{j^*}$; $S^p_{j^*} \leftarrow t_k$ in placement $\hat{l}$
                    $w_{k,\hat{l}} \leftarrow$ min pairwise robot distance using $S^p_{j^*}$ and $S_{j \neq j^*}$
                    if $w_{k,\hat{l}} > \rho * d_{ee}$ **then**
                        $b_{k,\hat{l}} = \beta_k * w_{k,\hat{l}}$; Compute β_k per d_2
                    else
                        $b_{k,\hat{l}} = 0$
            $b_{k^*,\hat{l}^*} = \max(b_{k,\hat{l}})$; $k^*, \hat{l}^* = \text{argmax}_{k,\hat{l}}(b_{k,\hat{l}})$
            if $b_{k^*,\hat{l}^*} > 0$ **then**
                $S_{j^*} \leftarrow t_{k^*}$ in placement $\hat{l}^*$
                remove t_{k^*} from all global and local layers
            else
                $S_{j^*} \leftarrow$ wait gap t_{wg} with min time to select next bidder
                $\delta \leftarrow$ condition from d_3
                if $\sum(makespan(t_{wg})) > \delta$ **then**
                    $S_{j^*} = S_{j^*} \setminus t_{wg}$
                    $T_{c,j^*} \leftarrow T_{i^*,j^*}$, selected per d_4; $S_{j^*} \leftarrow t_{tc}$ appended
        else
            $T_{c,j^*} \leftarrow T_{i^*,j^*}$, selected per d_4; $S_{j^*} \leftarrow t_{tc}$ appended
    else
        $\mathcal{B} = \mathcal{B} \setminus r_{j^*}$
$\mathcal{S} \leftarrow S_j$
return $\mathcal{S}$



(a) Failure Profile 1



(b) Failure Profile 2

Figure 5.6: Plots of the heuristic selection policies resulting from characteristic variable clustering. The clustering method used in (a) is K-means clustering, and the clustering method used in (b) is axis-aligned partitioning. The different colors represent the heuristic to select in a particular cluster. The training cases are marked with 'o's, and the test cases are marked with '+'s.

Chapter 6
CONCLUSIONS

6.1 Summary

The objective of this thesis is to increase the efficiency of aircraft assembly MRS by balanced task allocation, collision-free scheduling, and fast computation. To this end, the thesis' main contributions are (i) a partition-based scheduling approach and a market-based schedule optimizer in a two-stage scheduling framework to enable fast computation and balanced, collision-free task execution; (ii) a robot base placement and mobility strategy to maximize the workload sharing capacity of the robot team; and (iii) a decision-based approach for minimizing tool switching, coupled with a machine-learning-based framework for automatically selecting a scheduling technique based on problem characteristics.

6.1.1 Partition-Based Task Allocation and Scheduling

Chapter 3 presents an extension of partition-based allocation methods to nominal task scheduling, and develops a market-based schedule optimizer along with a two stage scheduling framework. These methods rely on assumptions that are reasonable for a variety of problems, and are therefore adaptable to a variety of aircraft manufacturing processes. Results show high MRS efficiency (98%), and computation times that are less than a second. This work was published in the *Robotics and Automation Letters* in 2019 [20].

6.1.2 Robot Base Placement and Mobility

Chapter 4 shows the effectiveness of robot base placement and mobility as a cost-effective means of increasing MRS efficiency. As robot failures increase, the ability for robots to share the workload and maintain MRS efficiency relies on their reach and the placement of their

bases. The methods in this chapter employ a modified market-based schedule optimization algorithm that enables task splitting and allows robot mobility and schedule re-optimization during the execution of the schedule. Results show an increase of efficiency by 3% on an average for systems that employ this base placement and mobility strategy.

6.1.3 Decision-Based Heuristics for Minimizing Tool Switching

Chapter 5 presents an adapted auction-based method that employs decision-based heuristics for minimizing tool switching, and then develops a machine-learning-based framework for automatically selecting the best heuristic to use for any given problem case based on the problem characteristics. Results show that our adapted auction method yields MRS efficiencies that are consistently higher than those resulting from using the only the partition-based scheduling methods from Chapter 3. Our data-driven approach for the selection of scheduling heuristics outperforms greedily selecting a top-average heuristic, and is furthermore generalized to be suitable for a variety of scheduling methods This work is in preparation for submission for publication in the journal of *Robotics and Computer-Integrated Manufacturing*.

6.1.4 Applications and Impact

The methods developed in this thesis are directly applicable to multi-robot aircraft structure assembly systems with stationary robotic arms. The development of the methods was in part enabled by the use a physical robot cell that is representative of feasible industrial manufacturing cells, as well as the guidance of aerospace industry experts. The physical system was instrumental in validating the assumptions that our methods made and the results that our methods obtained. As such, the methods can be applied to systems existing in the industry today. Moreover, because aircraft structure assembly is a time-consuming process (on the order of many hours), and aircraft parts are generally very expensive (on the order of millions of dollars), the time-savings in assembly that these methods provide is of substantial monetary benefit to manufacturers.

6.2 Future Investigation

This section provides directions for furthering the work in this thesis.

6.2.1 Task City Definitions

The leftover tasks arising due to robot failures, from Chapter 3, are grouped into task cities primarily by the particular failure during which they were skipped. The city grouping in Chapter 3 also ensures that the tasks in a city are geometrically adjacent (failures that results in city residing on two different ribs or spars are split into two cities). Chapter 4 further splits these cities by truncating them at the point of the shortest robot schedule, which allowed the market-based optimization to further increase the schedule efficiency. Therefore, a more robust approach to split these cities can investigated. This can be done in a similar manner to the approach used for the selection of the scheduling heuristics in Chapter 5. A data-driven approach for determining the appropriate task city definitions to enable maximal work balancability would be a worth-while effort.

6.2.2 Automated Selection of Problem Characteristic Variables

The framework for automatically selecting the appropriate scheduling heuristic given Chapter 5 is generalized to be suitable for a variety of methods. Similarly, the initial selection of the characteristics variables can be done in a more automated fashion, since, in Chapetr 5, the initial selection of the variable set was informed by the geometric nature of the problem. To this end, developing an automated approach to selecting the characteristic variables that describe the features of the problem would be a worthwhile effort in making the data-based approach also general to a variety of problems.

6.3 Remarks

I would like to express my gratitude to my advisors, Santosh Devasia and Ashis Banerjee, for providing guidance throughout my PhD tenure, as well as to my research colleagues and

the Boeing Advanced Research Center (BARC) at UW, without whom and which this work would not have been possible.

BIBLIOGRAPHY

[1] I. Landa-Torres, J. L. Lobo, I. Murura, D. Manjarres, and J. D. Ser, "Multi-objective heuristics applied to robot task planning for inspection plants," in *IEEE Cong. Evol. Comput.*, 2017, pp. 1621-1628.

[2] M. G. Filho, C. F. Barco, and R. F. T. Neto, "Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (FMS): A literature survey, classification and analysis," *Flex. Serv. Manuf. J.*, vol. 26, no. 3, Sep. 2014, pp. 408-431.

[3] A. Viharos and I. Nemeth, "Simulation and scheduling of AGV based robotic assembly systems," in *IFAC Symp. Inf. Control Problems Manuf.*, 2018, pp. 1415-1420.

[4] M. Elango, S. Nachiappan, and M. K. Tiwari, "Balancing task allocation in multi-robot systems using K-means clustering and auction based mechanisms," *Expert Syst. Appl.*, vol. 38, no. 6, Jun. 2011, pp. 6486-6491.

[5] S. Sariel and T. Balch. "Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments," in *AIAA Workshop Integrating Planning Scheduling*, 2005.

[6] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative Robots and Sensor Networks*. A. Kouba and J. R. M. Dios, Eds. Cham, Switzerland: Springer-Verlag, May 2015, pp. 31-51.

[7] E. Nunes, M. Manner, H. Mitiche and M. Gini, "A Taxonomy for Task Allocation Problems with Temporal and Ordering Constraints," in *Robot. and Auton. Syst.*, vol. 90, 2016, pp. 55-70.

[8] E. G. Jones, M. B. Dias, A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," in *Auton. Robots*, vol. 30, no. 1, 2011, pp. 41-50.

[9] J. Clausen, "Branch and bound algorithms-principles and examples," *Parallel Comput. Optimization*, 1999, pp. 239-267.

[10] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *AAAI Conf. Artif. Intel.*, 2017, pp. 4863-4869.

[11] A. Hussein, P. Marin-Plaza, F. Garcia, and J. M. Armingol, "Hybrid optimization-based approach for multiple intelligent vehicles requests allocation," in *Journal of Advanced Transportation*, 2018, pp. 1-12.

[12] K. E. C. Booth, T. T. Tran, G. Nejat, and J. C. Beck, "Mixed-integer and constraint programming techniques for mobile robot task planning," *IEEE Robotics and Automation Letters* vol. 1, no. 1, Jan. 2016, pp. 500-507.

[13] B. Kartal, E. Nunes, J. Godoy, and M. Gini, "Monte Carlo tree search for multi-robot task allocation," in *AAAI Conf. Artif. Intel.*, 2016, pp. 4222-4223.

[14] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," in *Robot.: Sci. Syst.*, 2013.

[15] E. Nunes and M. Gini, "Multi-robot auctions for allocation of tasks with temporal constraints," in *Proc. AAAI Conf. on Artif. Intel*, 2015, pp. 2110-2116.

[16] S. Ponda, J. Redding, H.-L. Choi, J. How, M. Vavrina, and J. Vian, "Decentralized planning for complex missions with dynamic communication constraints," in *American Control Conf.*, 2010, pp. 3998-4003.

[17] H. C. Lau, M. Sim and K. M. Teo, "Vehicle routing problem with time windows and a limited number of vehicles", in *European Journal of Operational Research* vol. 148, no. 3, 2003, pp. 559-569.

[18] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain, "Auction-based multi-robot routing," in *Robotics: Science and Systems (RSS)*, 2005, pp. 343-350.

[19] S. Sariel-Talay, T. Balch, N. Erdogan. Multiple traveling robot problem: A solution based on dynamic task selection and robust execution. *IEEE/ASME Trans. on Mechatronics*, vol. 14, no. 2, 2009, pp. 198-206.

[20] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, A. G. Banerjee, An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures, IEEE Robot. Auto. Let. 4(4) (2019) 3844-3851.

[21] N. Rahimi, J. Liu, A. Shishkarev, I. Buzytsky, and A. G. Banerjee, "Auction bidding methods for multiagent consensus optimization in supplydemand networks," *IEEE Robotics and Automation Letters* vol. 3, no. 4, Oct. 2018, pp. 4415-4422.

[22] L. Brunet, H.-L. Choi, and J. P. How, "Consensus-based auction approaches for decentralized task assignment," in *AIAA Guidance Navigation Control Conf.*, 2008.

[23] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proc. IEEE*, vol. 94, no. 7, Jul. 2006, pp. 1257-1270.

[24] T. Shima, S. J. Rasmussen, and P. Chandler, "UAV team decision and control using efficient collaborative estimation," in *American Control Conference*, 2005, pp. 4107-4112.

[25] P. B. Sujit and R. Beard, "Distributed sequential auctions for multiple UAV task allocation," in *American Control Conference*, 2007, pp. 3955-3960.

[26] G. A. Korsah, B. Kannan, I. Fanaswala, and M. B. Dias, "Improving market-based task allocation with optimal seed scheduling," *International Conference on Intelligent Autonomous Systems*, 2011, pp. 249-259.

[27] L. T. Liu and D. A. Shell, "Large-scale multi-robot task allocation via dynamic partitioning and distribution". *Auton. Robot.*, vol. 33, no. 3, Oct. 2012, pp. 291-307.

[28] J. Higuera, C. Gamboa, and G. Dudek, "Fair subdivision of multi-robot tasks, in *IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2999-3004.

[29] L. Klodt and V. Willert, "Equitable workload partitioning for multi-robot exploration through pairwise optimization," in *IEEE/RSJ Int. Conf. Intel. Robot. Syst.*, 2015, pp. 2809-2816.

[30] G. A. Korsah, M. B. Dias, and A. Stentz, "A comprehensive taxonomy for multi-robot task allocation," *Int. J. Robot. Res.*, vol. 32, no. 12, Oct. 2013, pp. 1495-1512.

[31] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," in *Intl. Journal of Robotics Research*, vol. 23 no. 9, 2004, pp. 939-954.

[32] L.T. Liu and D.A. Shell, "Optimal market-based multi-robot task allocation via strategic pricing," in *Robot.: Sci. Syst. Conf.*, 2013.

[33] A.M. Khamis, A.M. Elmogy, F.O. Karray, "Complex task allocation in mobile surveillance systems," in *J. Intell. Robot. Syst.* vol. 64, 2011, pp. 33-55.

[34] A. Hussein and A. Khamis, "Market-based approach to multi-robot task allocation," in *Int. Conf. on Indiv. and Coll. Behav. in Rob. (ICBR), IEEE*, 2013.

[35] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and control of multiple UAVs with timing constraints and loitering," in *American Control Conf.*, 2003, pp. 5311-5316.

[36] V. Jain and I. E. Grossmann, "Algorithms for hybrid MILP/CP models for a class of optimization problems", in *Journal on Computing*, vol. 13, no. 4, 2001, pp. 258-276.

[37] M. C. Gombolay, "Human-Machine Collaborative Optimization via Apprenticeship Scheduling" Ph.D. Thesis, MIT, 2017.

[38] N. Atay and B. Bayazit, "Mixed-integer linear programming solution to multi-robot task allocation problem", in *Technical Report WUCSE-2006-54, Department of Computer Science and Engineering, Washington University*, 2006.

[39] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," in *Numerische Mathematik*, vol 4. 1962, pp. 238-252.

[40] A. M. Geoffrion, "Generalized benders decomposition," in *Journal of Optimization Theory and Applications*, vol. 10 no. 4 1972, pp. 237-260.

[41] D. Spensieri, J. Carlson, R. Bohlin, J. Kressin, and J. Shi, "Optimal robot placement for tasks execution", in *6th CIRP Conf. on Assm. Tech. and Sys. (CATS)*, pp. 395-400, 2016.

[42] K. Lakshmanan and R. R. Rajkumar, "Scheduling self-suspending realtime tasks with rate-monotonic priorities," in *Proc. Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[43] G. Levitin, J. Rubinovitz and B. Shnits, "A genetic algorithm for robotic assembly line balancing," in *European Journal of Operational Research*, vol. 168, 2006, pp. 811-825

[44] M. U. Arif and S. Haider. An evolutionary traveling salesman approach for multi-robot task allocation. *9th Itrn. Conf. on Agents and Art. Intel. (ICAART)*, 2017, pp. 567-574.

[45] M. Aly, A. Abbas and S. M. Megahed, "Robot workspace estimation and base placement optimisation techniques for the conversion of conventional work cells into autonomous flexible manufacturing systems," in *International Journal of Computer Integrated Manufacturing*, vol. 23 no. 12, 2010, pp. 1133-1148.

[46] N. Vahrenkamp, T. Asfour and R. Dillmann, "Robot placement based on reachability inversion," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on IEEE*, 2013 pp. 1970-1975.

[47] F. L Hammond III and K. Shimada, "Improvement of redundant manipulator task agility using multiobjective weighted isotropy-based placement optimization," in *IEEE Int. Conf. on Robotics and Biomimetics*, 2009, pp. 645-652.

[48] M. Gadaleta, G. Berselli and M. Pellicciari, "Energy-optimal layout desing of robotic work cell: Potential assessment on an idustrial case study," in *Robotics and Computer-Integrated Manufacturing*, vol. 47, 2017, pp. 102-111.

[49] R. Kalawoun, S. Lengagne and Y. Mezouar, "Optimal robot base placements for coverage tasks," in *IEEE 14th Itrl. Conf. on Aut. Sci. and Engin. (CASE)*, 2018.

[50] A. Makhal, A. K. Goins, "Reuleaux: robot base placement by reachability analysis," in *IEEE Itn. Conf. Rob. Comp.*, 2018.

[51] J. Dong, J. C. Trinkle, "Orientation-based reachability map for robot base placement," in *IEEE/RSJ Int. Conf. Intl. Rob. Sys. (IROS)*, 2015.

[52] F. Zacharias, C. Borst, M. Beetz, and G. Hirzinger, Positioning mobile manipulators to perform constrained linear trajectories, in *IEEE/RSJ Int. Conf. Intl. Rob. Sys. (IROS)*, pp. 2578-2584. IEEE, 2008.

[53] G. A. Korsah, B. Kannan, B. Browning, and M. B. Dias, "xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2012, pp. 115-122.

[54] L. Luo, N. Chakraborty and K. Sycara, "Multi-robot assignment algorithms for tasks with set precedence constraints," in *Proceedings of IEEE International Conference on Robotics and Automation, 2011*, May 2011.

[55] M. McIntire, E. Nunes and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. AAMAS '16* 2016, pp. 1078-1086.

[56] Dorothea Calmels, "The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends," in *International Journal of Production Research*, vol 57 no. 15-16, 2019, pp. 5005-5025.

[57] Beezo, A. C., J.-F. Cordeau, G. Laporte, and H. H. Yanasse, Scheduling Identical Parallel Machines with Tooling Constraints, in *European Journal of Operational Research* vol. 257 no. 3, 2017, pp. 834844.

[58] David Adjiashvili, Sandro Bosio, Kevin Zemmer, "Minimizing the number of switch instances on a flexible machine in polynomial time", in *Operations Research Letters*, vol. 43 no. 3, 2015, pp. 317-322.

[59] Farughi, H., M. Dolatabadiaa, V. Moradi, V. Karbasi, and S. Mostafayi, Minimizing the Number of Tool Switches in Flexible Manufacturing Cells Subject to Tools Reliability Using Genetic Algorithm in *Journal of Industrial and Systems Engineering*, vol. 10 (special issue on Quality Control and Reliability), 2017, pp. 1733.

[60] Crama, Y., L. S. Moonen, F. C. R. Spieksma, and E. Talloen, The Tool Switching Problem Revisited. *European Journal of Operational Research* vol. 182 no. 2, 2007, pp. 952957.

[61] S. Pellegrinelli, N. Pedrocchi, L. M. Tosatti, A. Fischer, T. Tolio, "Multi-robot spot-welding cells for car-body assembly: Design and motion planning", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 44, 2017, pp. 97-116.

[62] E. Glorieux, S. Riazi, B. Lennartson, "Productivity/energy optimisation of trajectories and coordination for cyclic multi-robot system", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 49, 2018, pp. 152-161.

[63] K. Baizid, A. Yousnadj, A. Meddahu, R. Chellali, J. Iqbal, "Time scheduling and optimization of industrial robotized tasks based on genetic algorithms", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 34, 2015, pp. 140-150.

[64] P. Zheng, J. Wang, J. Zhang, C. Yang, Y. Jin, "An adaptive CGAN/IRF-based rescheduling strategy for aircraft parts remanufacturing system under dynamic environment", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 58, 2019, pp. 230-238.

[65] K. Geetha, D. Ravindran, M. Siva Kumar, M. N. Islam, "Concurrent tolerance allocation and scheduling for complex assemblies", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 35, 2015, pp. 84-95.

[66] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints", *AAAI Conf. Artif. Intel.*, 2017, pp. 4863-4869.

[67] G. D. Orio, G. Cndido, J. Barata, "The Adapter module: A building block for Self-Learning Production Systems", *Rob. Comp. Integ. Mfg. (R-CIM)*, vol. 36, 2015, pp. 25-35.

[68] J. E. Amaya, C. Cotta, and A. J. Fernndez-Leiva, "Solving the Tool Switching Problem with Memetic Algorithms", *Artif. Intel. Eng. Des. Anl. Mfg.*, vol. 26 no. 2, 2012, pp. 221235.

[69] A. P. Burger, C. G. Jacobs, J. H. van Vuuren, and S. E. Visagie, "Scheduling multi-colour print jobs with sequence-dependent setup times", *J. of Sched.*, vol. 18 no.2, 2015, pp. 131145.

[70] G. S. Paiva, and M. A. M. Carvalho, "Improved Heuristic Algorithms for the Job Sequencing and Tool Switching Problem", *Comp. & Oper. Res.* vol. 88, 2017, pp. 208219.

[71] S. zpeynirci, B. Gkgr, and B. Hnich, "Parallel machine scheduling with tool loading", *Apld. Math. Modelling* vol. 40 no. 910, 2016, pp. 56605671.

[72] B. Gkgr, B. Hnich, and S. zpeynirci, "Parallel machine scheduling with tool loading: a constraint programming approach", *Int. J. Prod. Res.* vol. 54, 2018, pp. 117.

[73] S. L. Brunton, J. L. Proctor, N. J. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems", *Proc. Natl. Acad. Sci.*, vol. 113 no. 15, 2016, pp. 3932-3937.

Appendix A

SUPERVISORY CELL CONTROL SYSTEM

This appendix outlines the supervisory control software that was developed to operate the robots per computed schedules.

A.1 System Architecture and Components

A.1.1 System Architecture

Here, the architecture of the MRS software system is described. The architecture comprises the supervisory control (M-RACC: multi-robot assembly cell control) software, the scheduling software, the redundant collision avoidance software, on-board robot vision system, and the robots themselves (physical or virtual). Figure A.1 shows the system architecture.

A.1.2 Robot Vision

The vision system entails cameras mounted on the robots and an NI LabView image processing program. Robot vision is here used for calibrating robots to a given section of the wing. The robot jogs to where it thinks a fiducial is located, and the LabView program calculates by how much the robot is off. Using three fiducial markers in a given section of the wing, the robot is able to calibrate its coordinate system to that of the wing section.

A.1.3 Scheduler

The scheduler is a MATLAB program that runs the scheduling methods described in this thesis. The schedules are written to a CSV file that M-RACC then retrieves and interprets to appropriately control the robots.

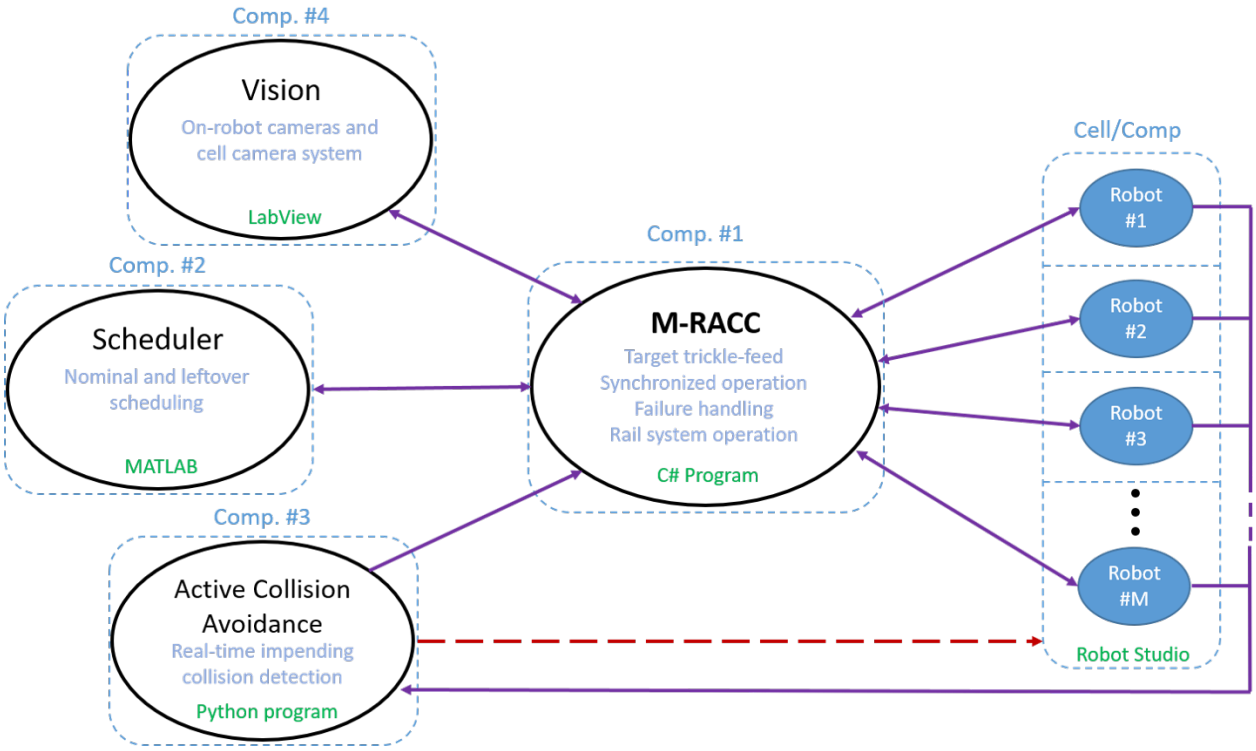


Figure A.1: MRS system software architecture. The supervisory control software (M-RACC: multi-robot assembly cell control) is the centrally responsible agent. The other agents answer to and are controlled through M-RACC, with the exception of a safety stop that can be triggered by the collision avoidance software directly. This is to ensure that the robots are safe even if M-RACC doesn't control the robots properly.

A.1.4 Active Collision Avoidance (ACA)

The collision avoidance software monitors the poses of the robots by streaming robot poses every 50 ms. Impending collision is detected by overlaying envelopes over the robots' dimensions (with a margin of safety) and detecting intersections between the safety envelopes, as shown in Figure A.2. Two envelopes are used: a larger safety margin envelope is used to

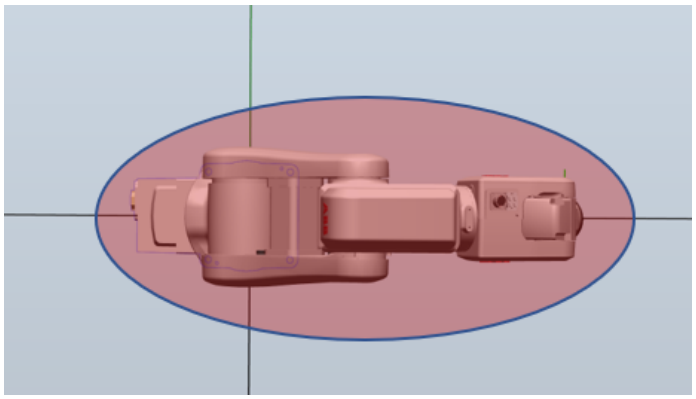


Figure A.2: A robot with an overlaid safety envelope. The collision avoidance software detects intersections between the safety envelops and prevents the robots from physically colliding with each other.

inform M-RACC that a collision is impending, and a smaller safety envelope is used for the collision avoidance software to then trigger an emergency stop to ensure a collision doesn't occur.

Appendix B

ROBOT BASE PLACEMENT AND MOBILITY RESULTS

This appendix contains results from the various cases considered in Chapter 4. The two sections in this appendix cover (i) various individual cases where robot mobility was either beneficial or ineffective, and (ii) the efficiency benefit results for all the conditions of assembly (COAs) from Chapter 3.

B.1 Individual Case Studies

This sections shows several example cases that illustrate when mobility is beneficial and when it gives no benefit.

B.1.1 Cases Benefiting from Robot Mobility

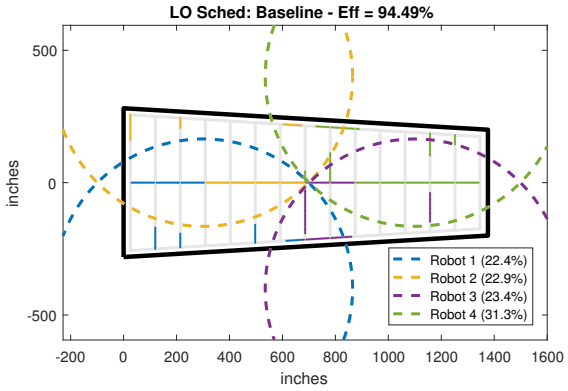
Four cases that benefited from robot mobility are given here. For each case, three plots are given: (a) baseline allocation, (b) allocation using only CitySplit, and (c) allocation with CitySplit and mobility.

B.1.2 Cases Not Benefiting from Robot Mobility

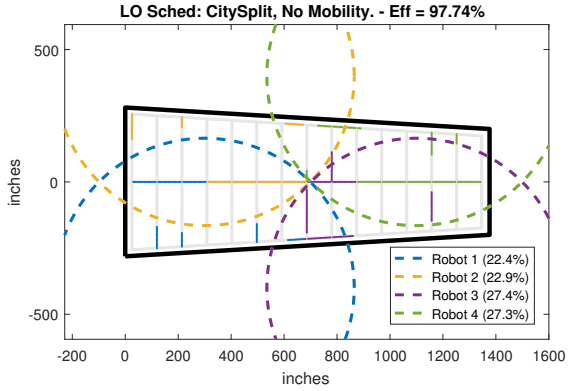
Four cases that did not benefit from robot mobility are given here. We found that these cases had less cities in the overlap region than the cases that benefited from mobility.

B.2 Results for all COAs

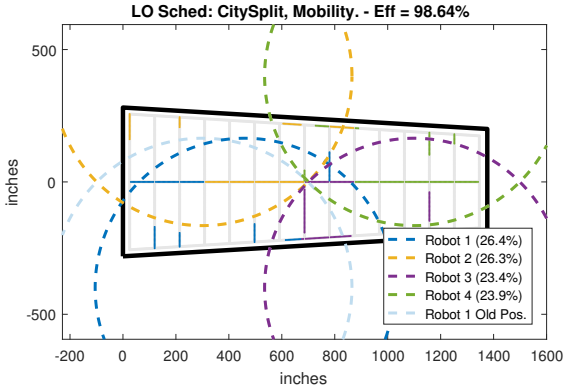
The efficiency results from stationary robot base placement and mobile robots for all COAs are given here. The results for the first COA are provided in Chapter 4.



(a) Baseline

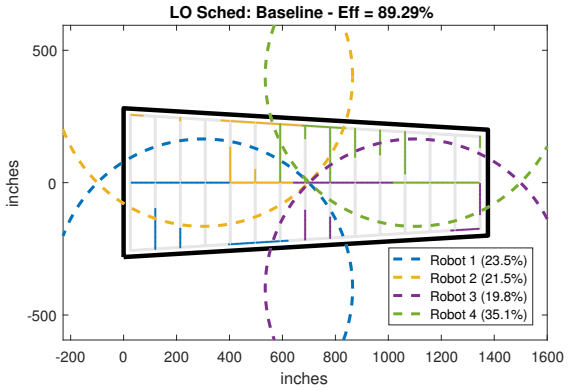


(b) City Split Only

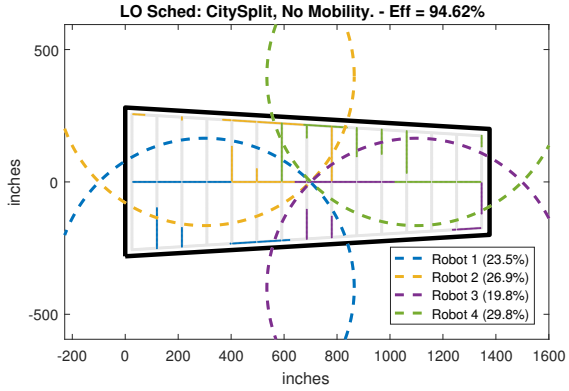


(c) With Mobility

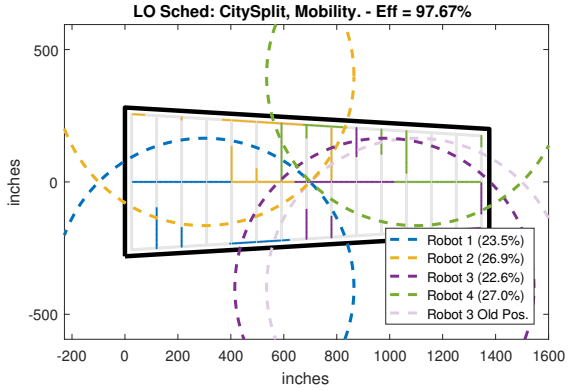
Figure B.1: First example case that benefited from mobility. Shown are (a) the baseline allocation, (b) allocation using only CitySplit, and (c) allocation with CitySplit and mobility.



(a) Baseline

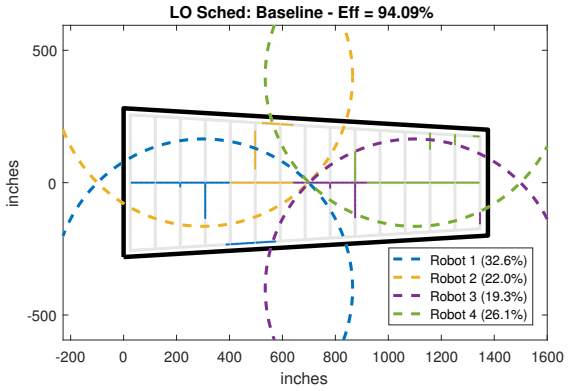


(b) City Split Only

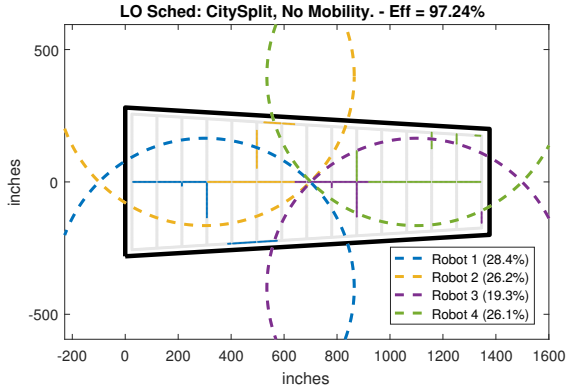


(c) With Mobility

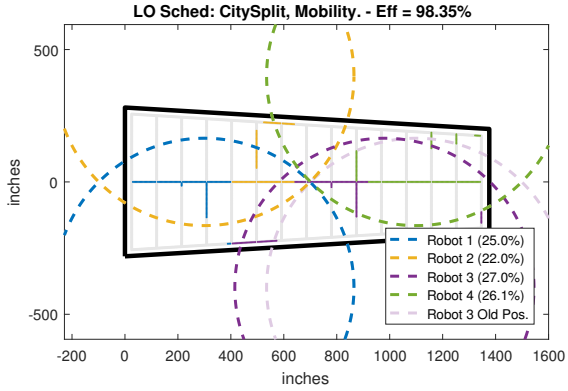
Figure B.2: Second example case that benefited from mobility. Shown are (a) the baseline allocation, (b) allocation using only CitySplit, and (c) allocation with CitySplit and mobility.



(a) Baseline

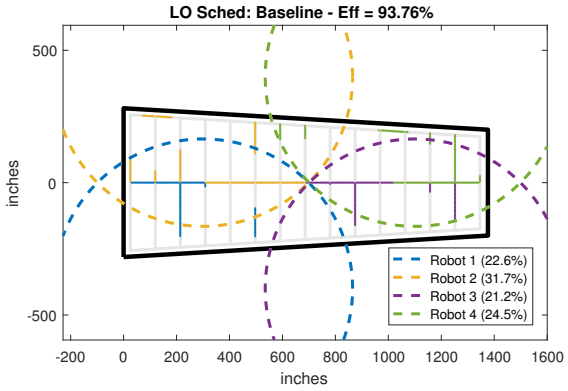


(b) City Split Only

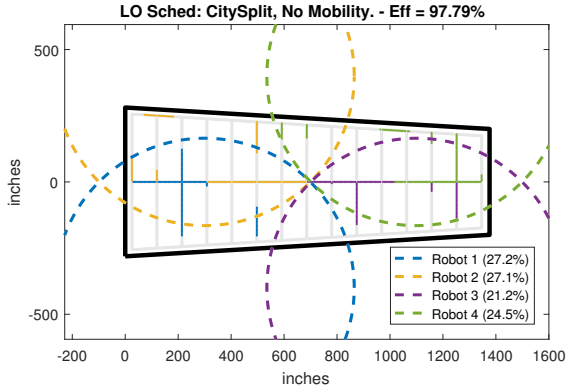


(c) With Mobility

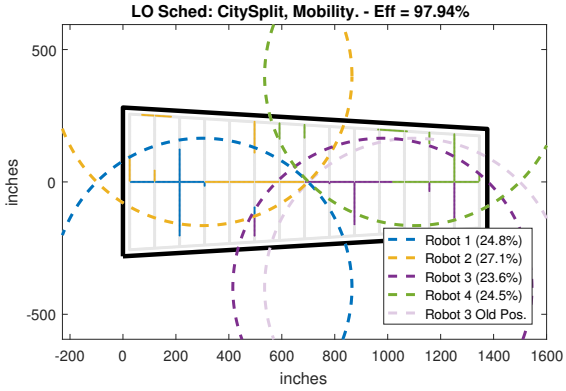
Figure B.3: Third example case that benefited from mobility. Shown are (a) the baseline allocation, (b) allocation using only CitySplit, and (c) allocation with CitySplit and mobility.



(a) Baseline



(b) City Split Only



(c) With Mobility

Figure B.4: Fourth example case that benefited from mobility. Shown are (a) the baseline allocation, (b) allocation using only CitySplit, and (c) allocation with CitySplit and mobility.

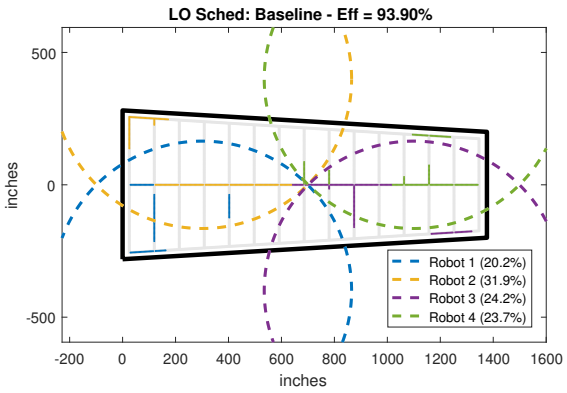


Figure B.5: First example case that did not benefit from mobility.

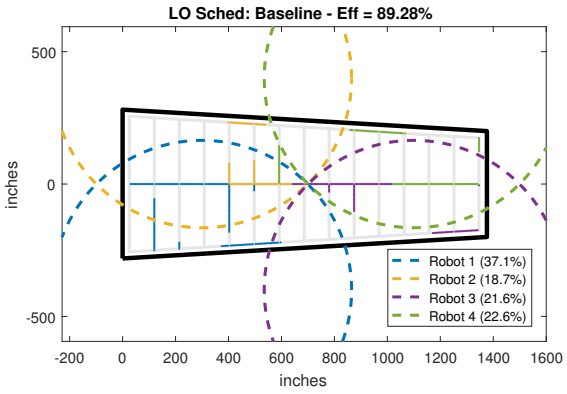


Figure B.6: Second example case that did not benefit from mobility.

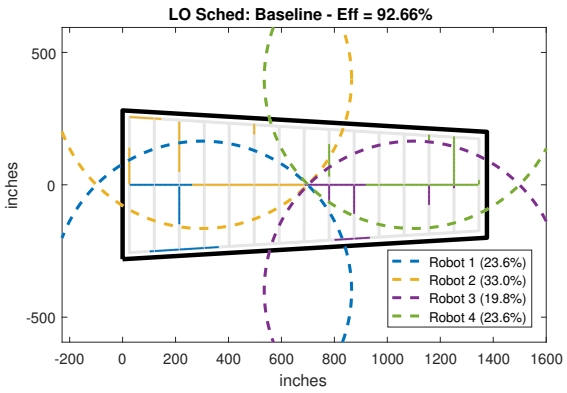


Figure B.7: Third example case that did not benefit from mobility.

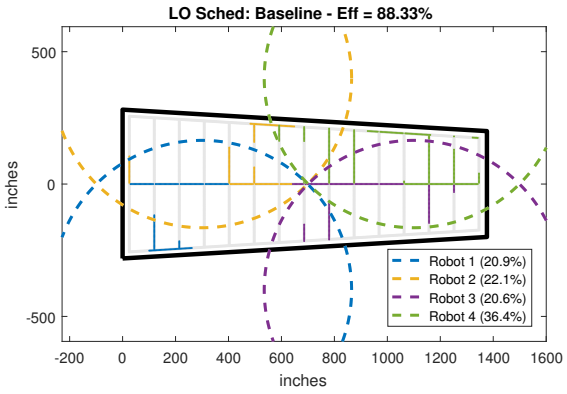
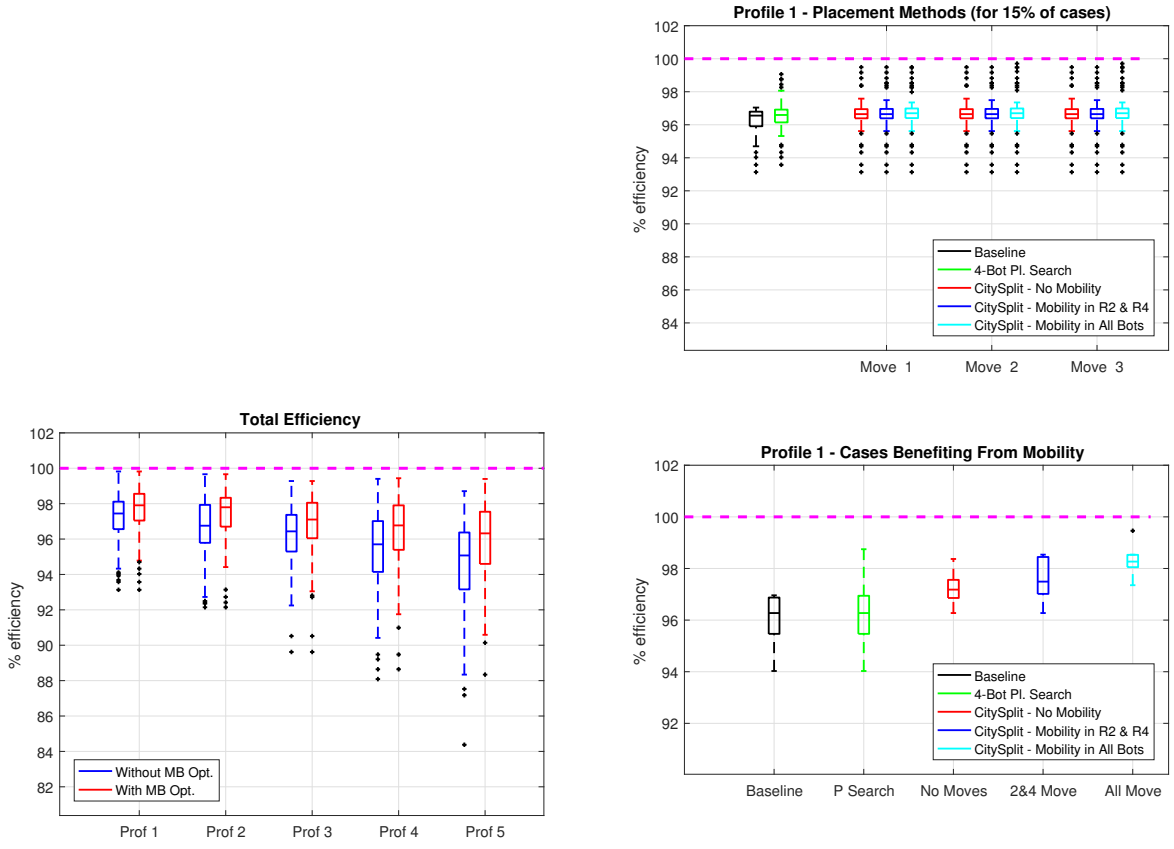


Figure B.8: Fourth example case that did not benefit from mobility.

B.2.1 COA 2 Results

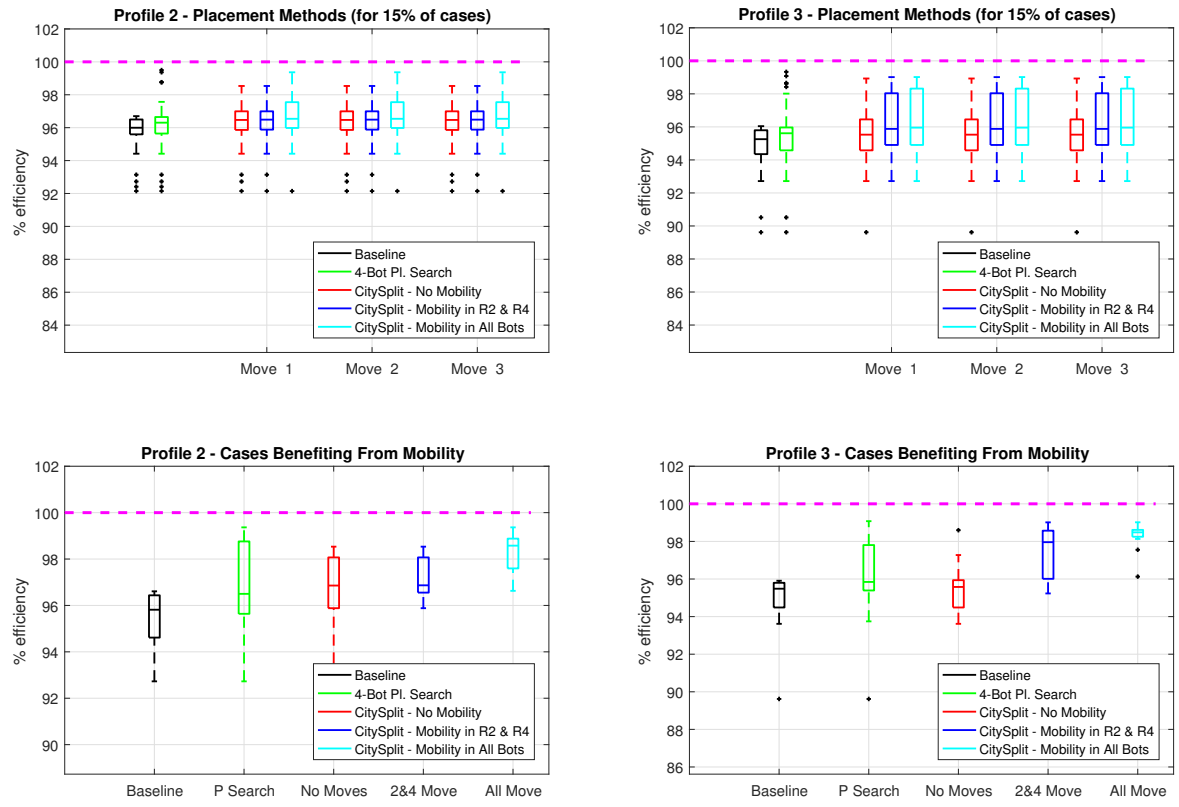
The results for COA 2 are given here. The results are consistent with the findings in Chapter 4.



(a) Baseline Efficiency

(b) Profile 1

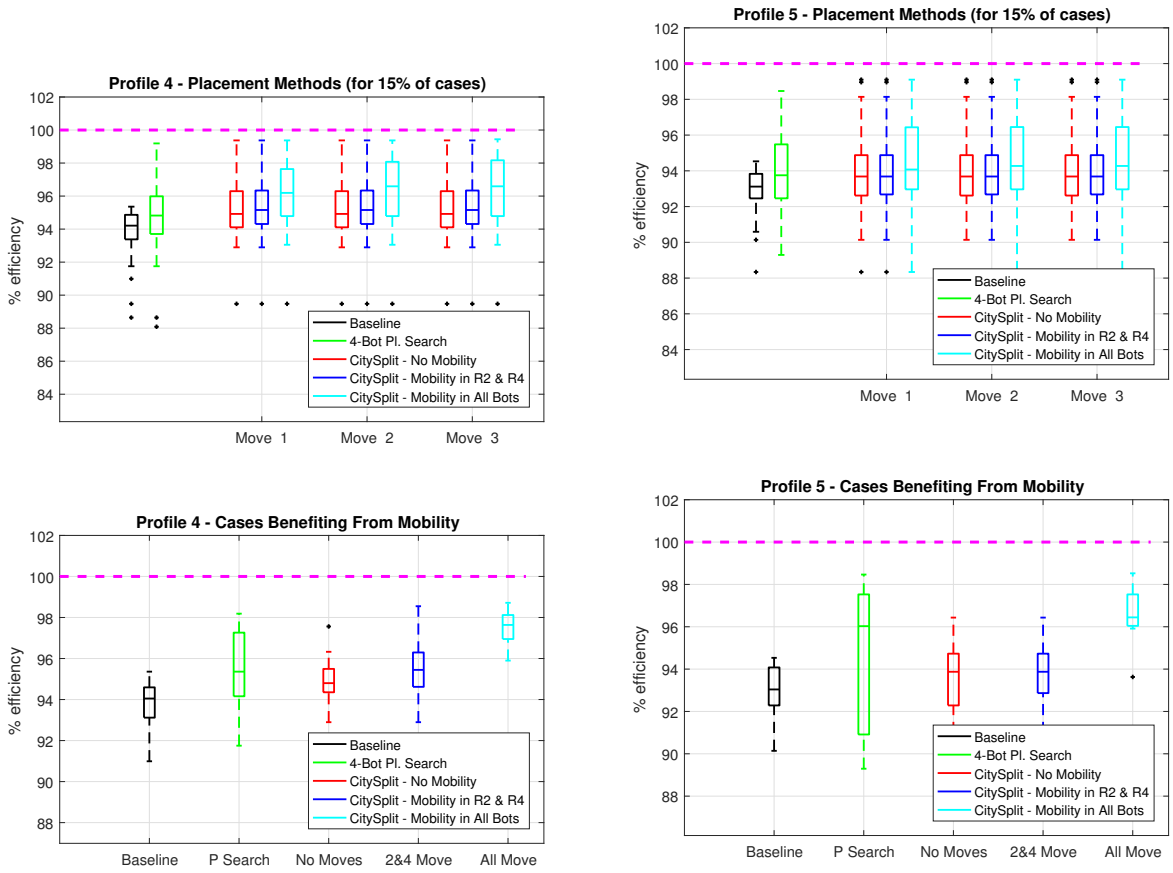
Figure B.9: Efficiency results for COA 2 showing (a) baseline efficiency for all profiles, and (b) efficiency results using the methods in Chapter 4 for failure Profile 1.



(a) Profile 2

(b) Profile 3

Figure B.10: Efficiency results for COA 2 using (a) failure Profile 2 and (b) failure Profile 3.



(a) Profile 4

(b) Profile 5

Figure B.11: Efficiency results for COA 2 using (a) failure Profile 4 and (b) failure Profile 5.

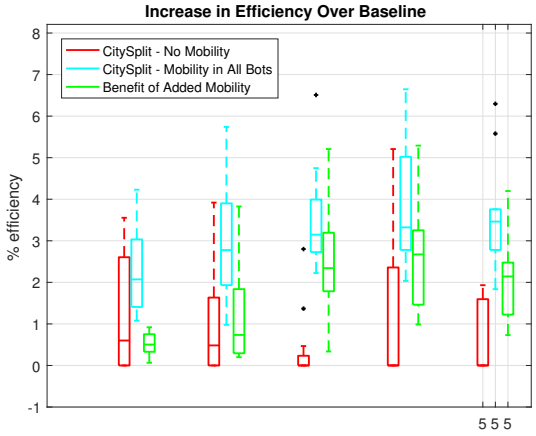


Figure B.12: Efficiency results for COA 2 showing the increase in efficiency over baseline for CitySplit with and without mobility.

B.2.2 COA 3 Results

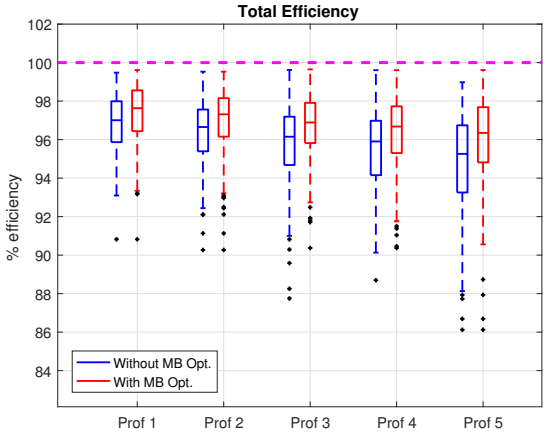
The results for COA 3 are given here. The results are consistent with the findings in Chapter 4.

B.2.3 COA 4 Results

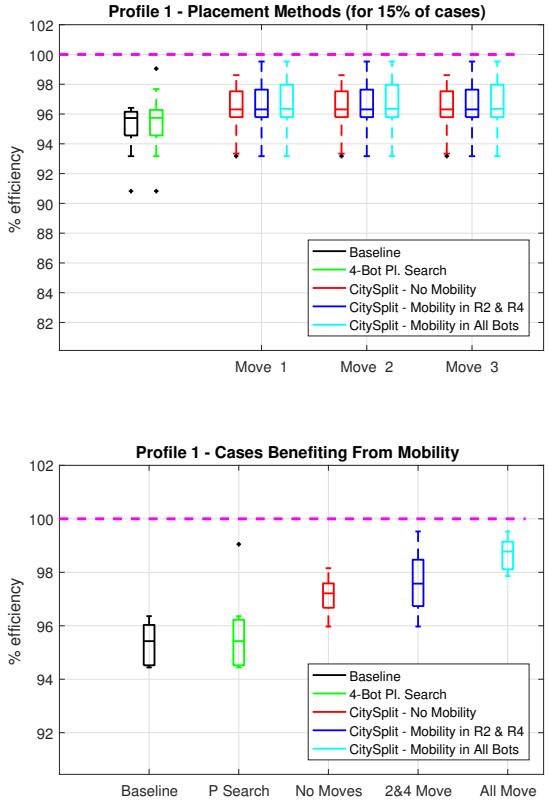
The results for COA 4 are given here. The results are consistent with the findings in Chapter 4.

B.2.4 COA 5 Results

The results for COA 5 are given here. The results are consistent with the findings in Chapter 4.

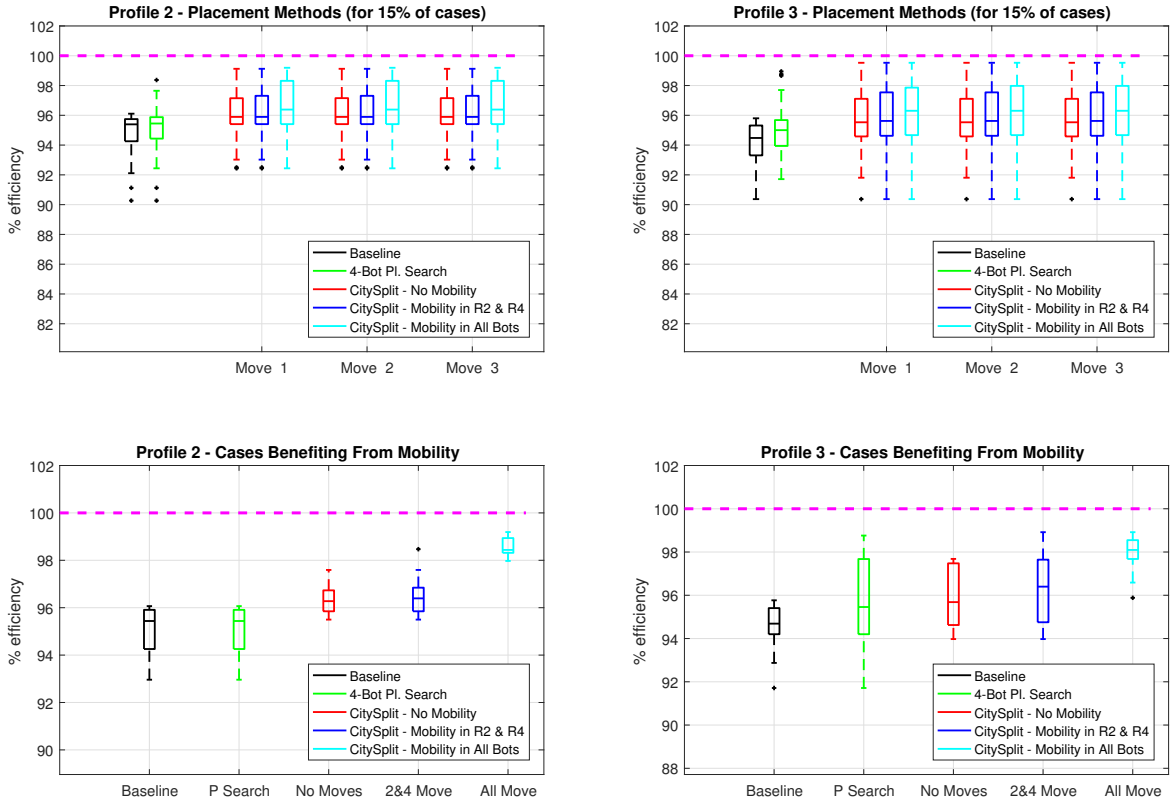


(a) Baseline Efficiency



(b) Profile 1

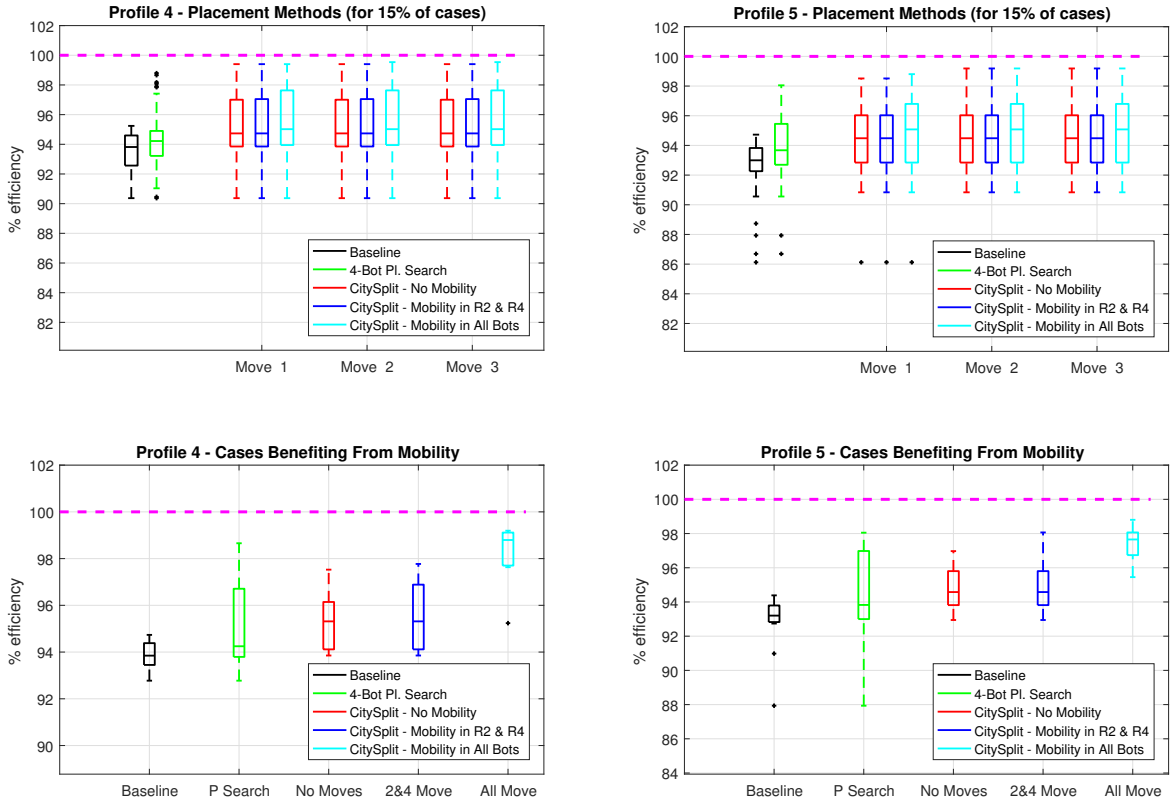
Figure B.13: Efficiency results for COA 3 showing (a) baseline efficiency for all profiles, and (b) efficiency results using the methods in Chapter 4 for failure Profile 1.



(a) Profile 2

(b) Profile 3

Figure B.14: Efficiency results for COA 3 using (a) failure Profile 2 and (b) failure Profile 3.



(a) Profile 4

(b) Profile 5

Figure B.15: Efficiency results for COA 3 using (a) failure Profile 4 and (b) failure Profile 5.

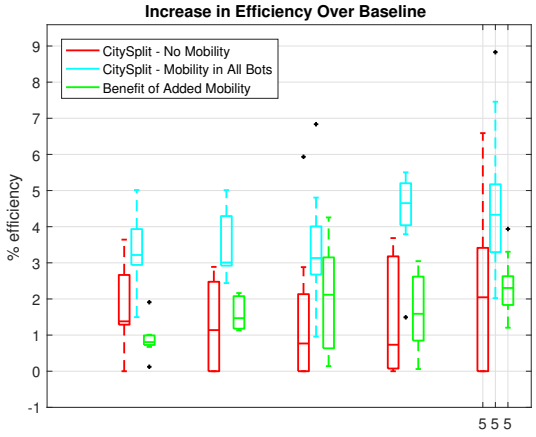
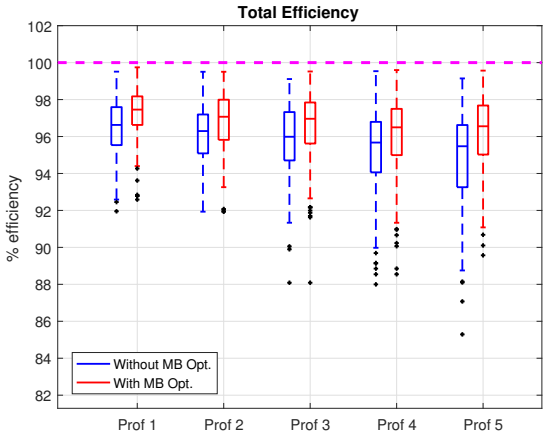
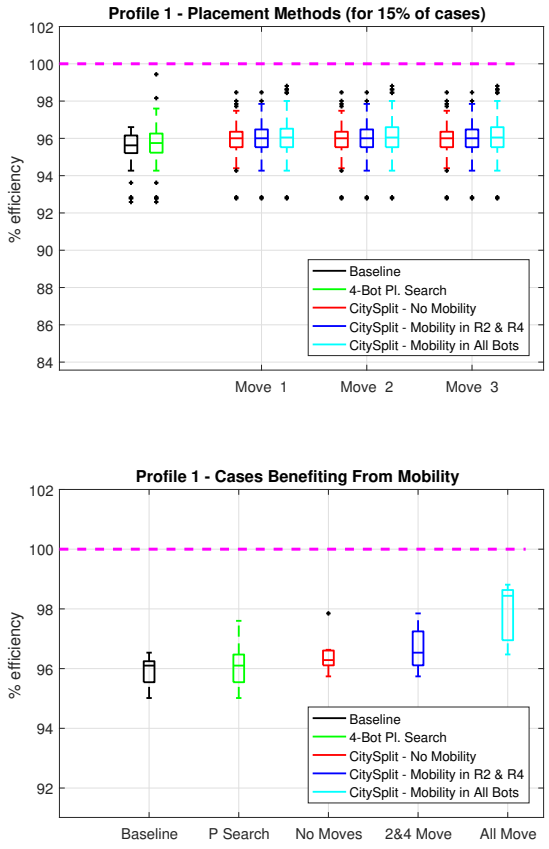


Figure B.16: Efficiency results for COA 3 showing the increase in efficiency over baseline for CitySplit with and without mobility.

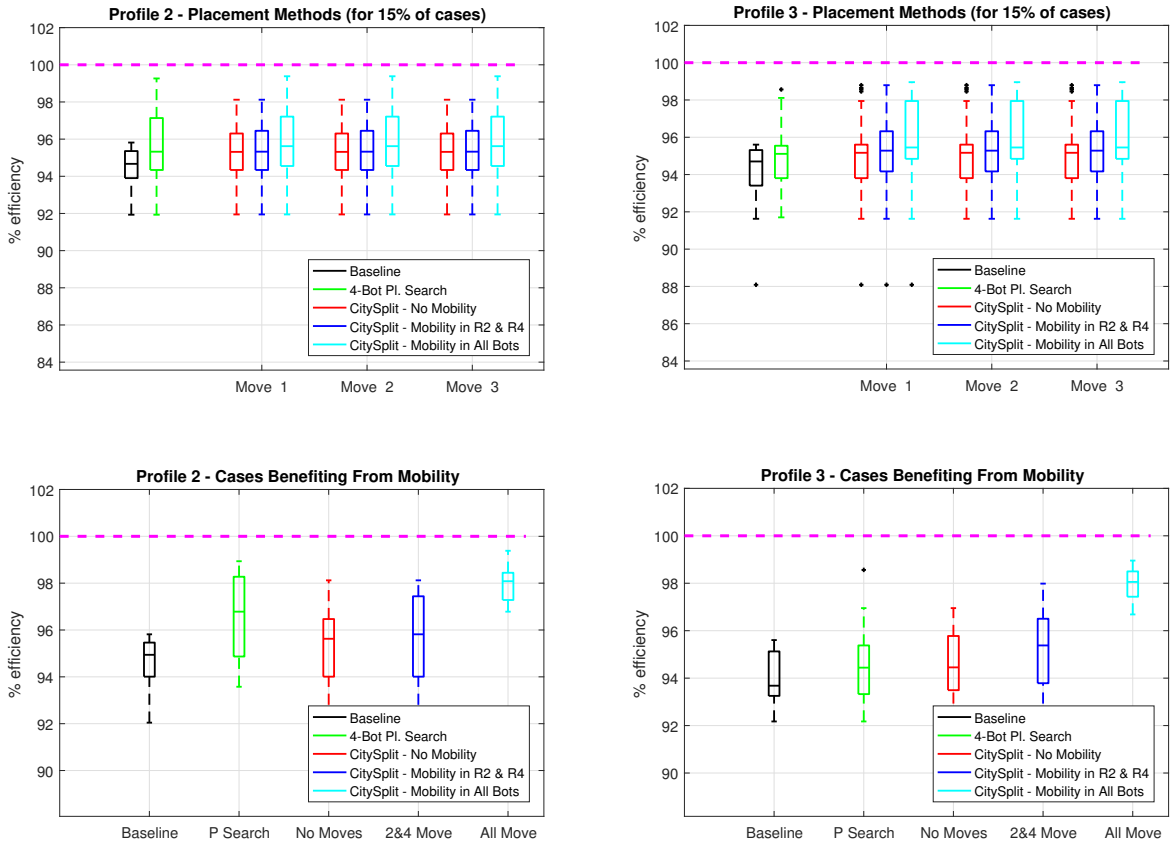


(a) Baseline Efficiency



(b) Profile 1

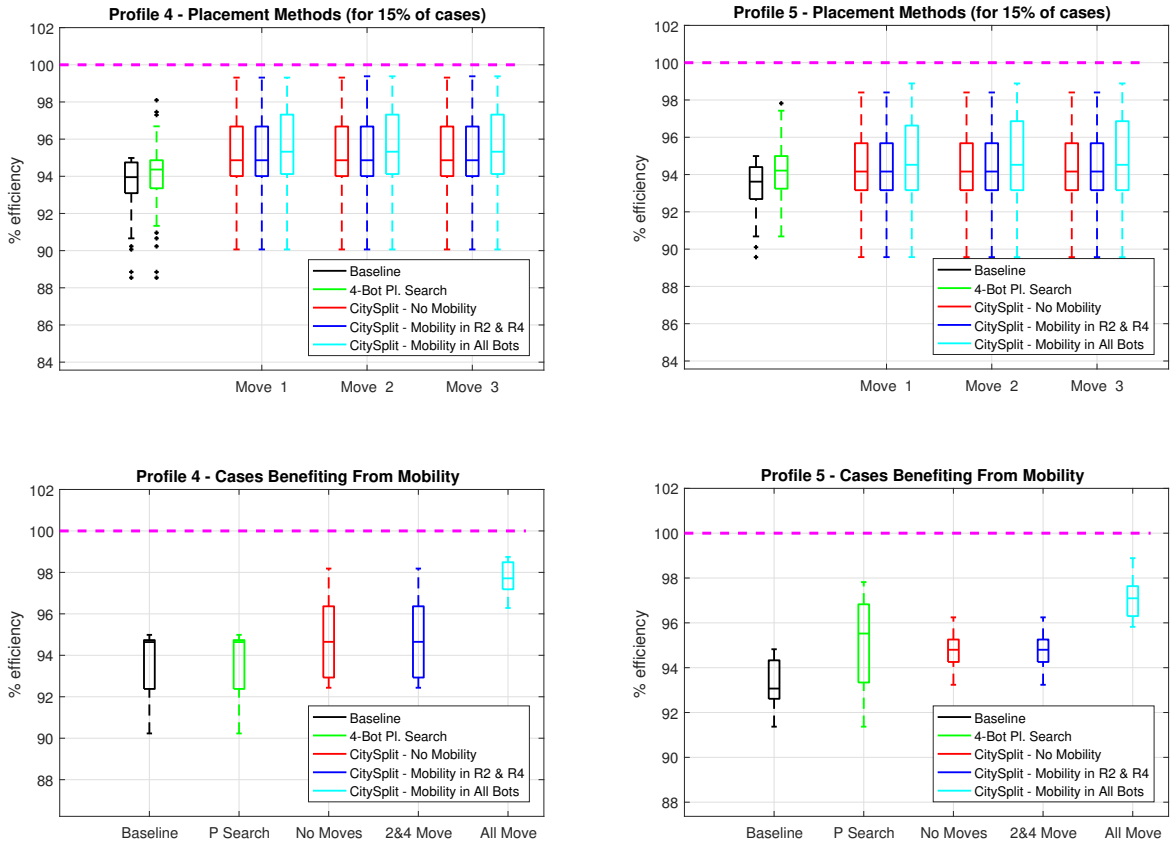
Figure B.17: Efficiency results for COA 4 showing (a) baseline efficiency for all profiles, and (b) efficiency results using the methods in Chapter 4 for failure Profile 1.



(a) Profile 2

(b) Profile 3

Figure B.18: Efficiency results for COA 4 using (a) failure Profile 2 and (b) failure Profile 3.



(a) Profile 4

(b) Profile 5

Figure B.19: Efficiency results for COA 4 using (a) failure Profile 4 and (b) failure Profile 5.

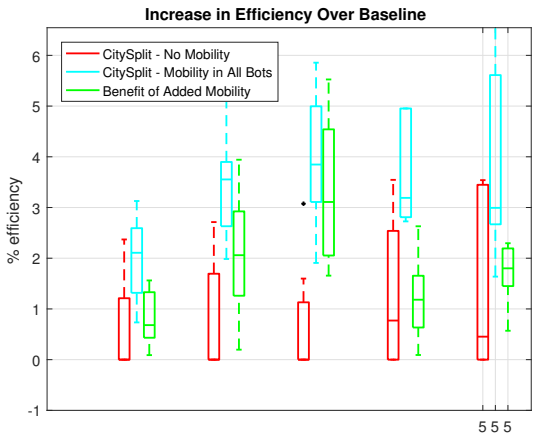
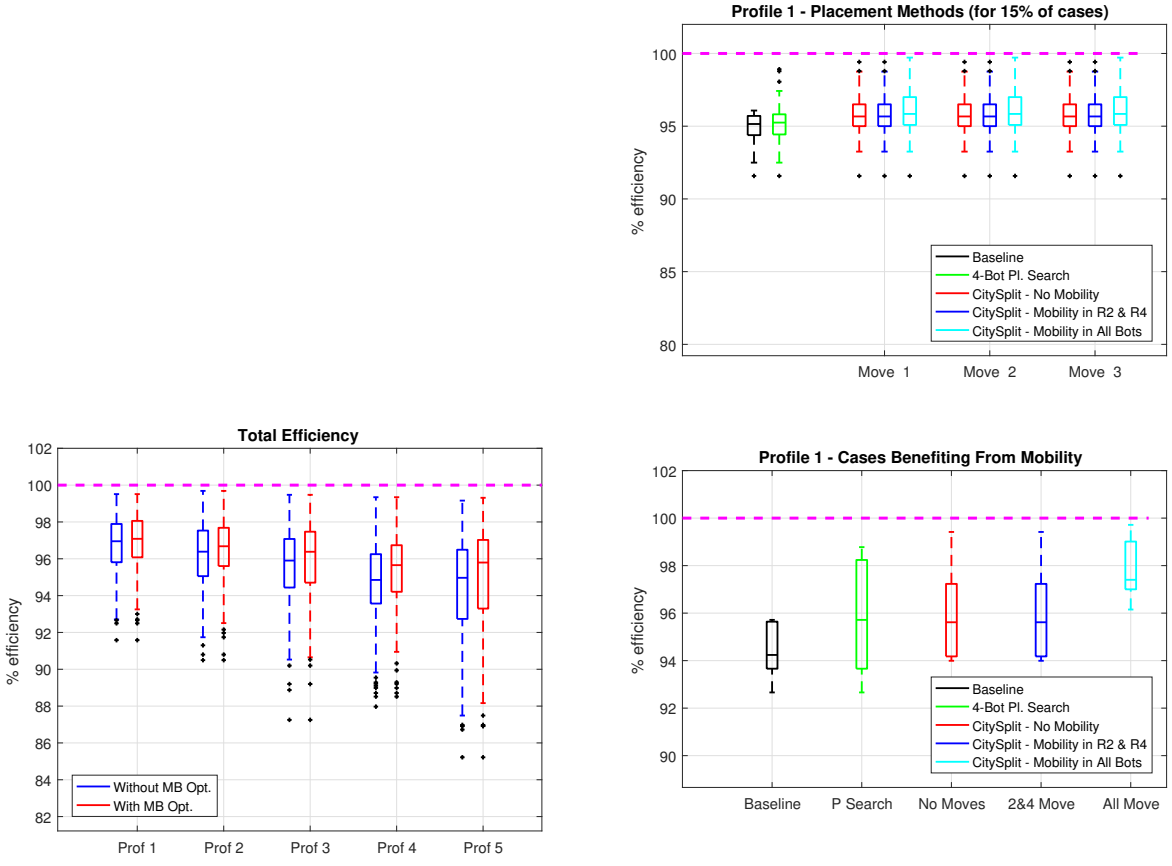


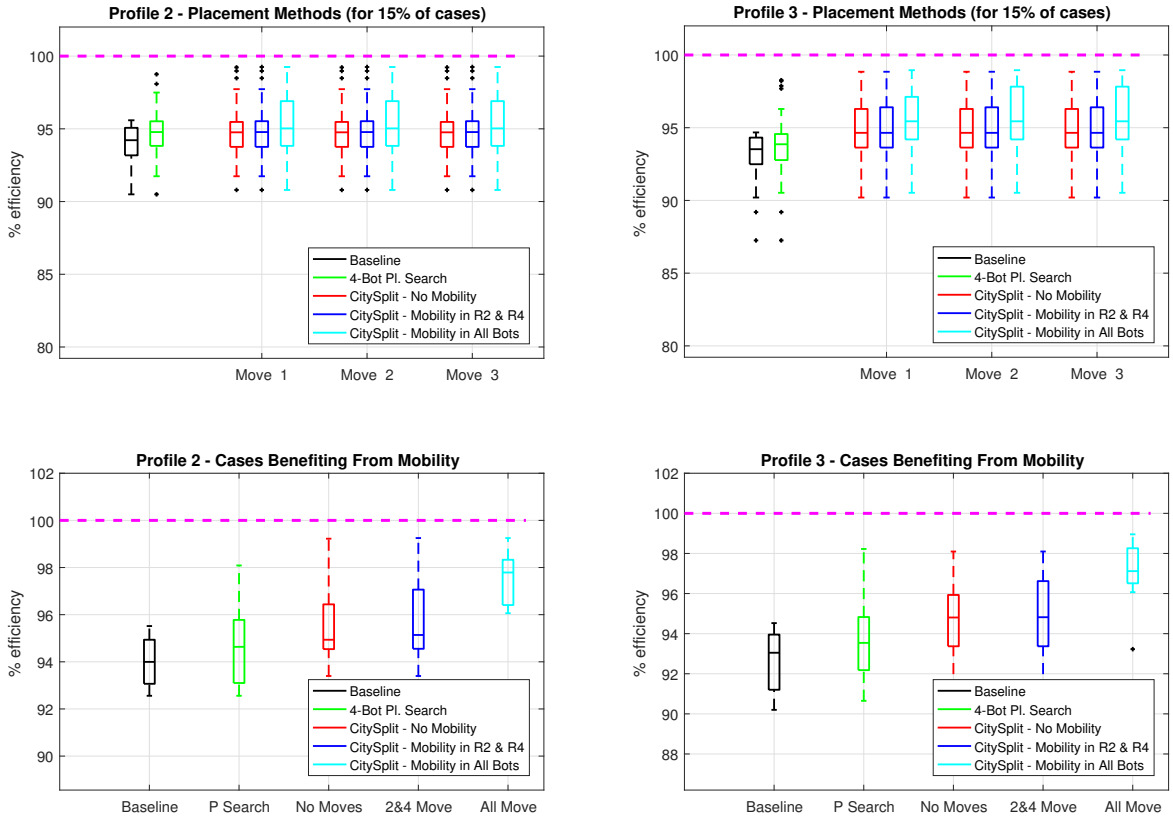
Figure B.20: Efficiency results for COA 4 showing the increase in efficiency over baseline for CitySplit with and without mobility.



(a) Baseline Efficiency

(b) Profile 1

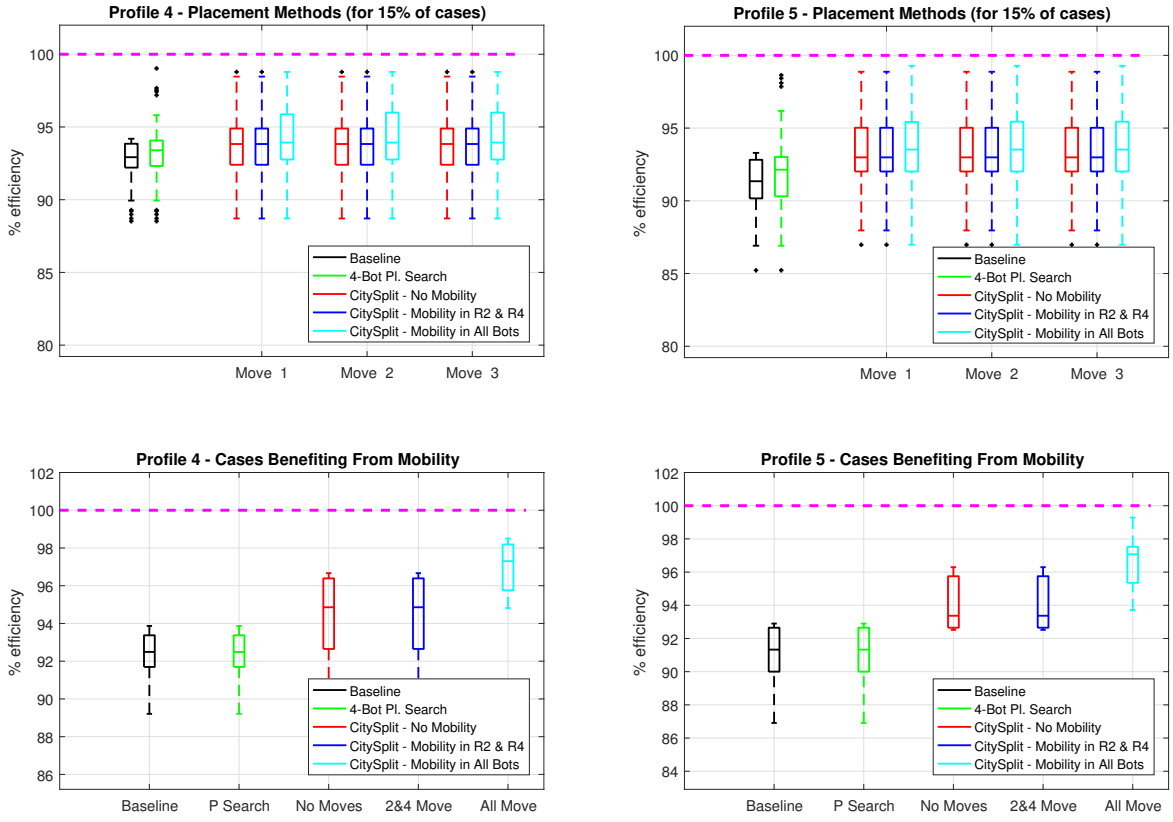
Figure B.21: Efficiency results for COA 5 showing (a) baseline efficiency for all profiles, and (b) efficiency results using the methods in Chapter 4 for failure Profile 1.



(a) Profile 2

(b) Profile 3

Figure B.22: Efficiency results for COA 5 using (a) failure Profile 2 and (b) failure Profile 3.



(a) Profile 4

(b) Profile 5

Figure B.23: Efficiency results for COA 5 using (a) failure Profile 4 and (b) failure Profile 5.

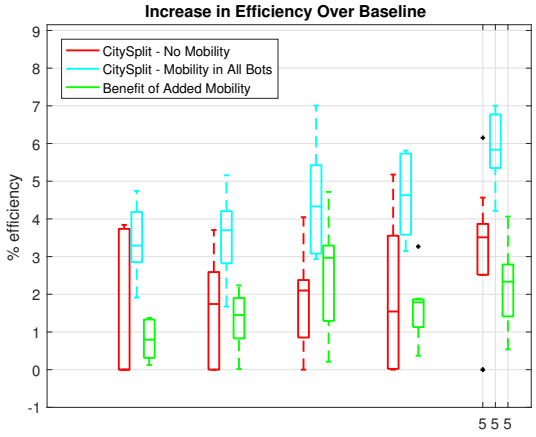


Figure B.24: Efficiency results for COA 5 showing the increase in efficiency over baseline for CitySplit with and without mobility.