Optimized Participant Assignment for an Online Experimental Framework

A Dissertation Presented
by

Ceyhun Efe Karbeyaz

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Engineering

Northeastern University
Boston, Massachusetts

Apr 2015

to my mother and father...

Contents

List of Figures

List of Tables

Acknowledgments

I am deeply grateful to my advisor, Prof. Waleed Meleis, for his guidance, patience and support over the years we have worked together. His knowledge and ideas have shaped the course of my research.

I would like to thank my dissertation committee members, Asst. Prof. Ningfang Mi, Prof. David Lazer and Assoc. Prof. Magy Seif El-Nasr for their insightful comments and contributions.

I would like to thank to Prof. David Lazer and his research group for giving me the opportunity to act as a member of their research team.

I would like to thank to my brother Ersel Karbeyaz and sister-in-law Başak Ülker Karbeyaz for their assistance during my graduate studies.

A special thanks to my family. Words cannot express how grateful I am to my mother Tülin Karbeyaz, sisters Ceylin Karbeyaz and Efsun Karbeyaz.

Abstract of the Dissertation

Optimized Participant Assignment for an Online Experimental Framework

by

Ceyhun Efe Karbeyaz

Doctor of Philosophy in Computer Engineering

Northeastern University, Apr 2015

Assoc. Prof. Waleed Meleis, Adviser

Experimental research is being transformed from being based in physical laboratories centered in research universities into web-based experimental platforms. Our group has built a web-based analog of a university's shared laboratory for behavioural research. Our platform is an experimental testbed that enables researchers to quickly design and deploy behavioural experiments. The framework is implemented to conduct very large experiments with many individuals interacting synchronously. However, large scale platforms come with drawbacks that can affect both participants and researchers. From the point of view of participants, too many participants mean excessive waiting times in the system. From the point of view of researchers, meeting the deadlines of different experiments and implementation details of the experiments can be problematic and may affect the research progress.

In this thesis, we develop and evaluate algorithms that efficiently assign incoming participants to experiments to meet experiments' deadlines. We show that participant assignment can be modeled as the problem of minimizing the weighted tardiness of a parallel schedule, an NP-complete problem. We consider varying participant arrival and experiment availability cases for the participant assignment problem and use them to evaluate the effectiveness of exhaustive, greedy, dynamic programming and integer-linear programming techniques on synthetic and real benchmarks.

The results over the synthetic and real benchmarks show that greedy algorithms perform better than the other compared algorithms for algorithm execution and solution optimality. With constant experiment release times, we observe that that the greedy ATC and ATCPA algorithms outperform other compared algorithms. For both online and offline greedy algorithms with varying experiment release times, we observe that the greedy MPRA algorithm is the best performing algorithm. The experiments over the online algorithms with varying participant arrivals and experiment release times indicate that experiments' release times and the number of experiments makes the biggest impact over minimizing the tardiness. We show that the MPRA algorithm performs well over both the synthetic datasets with different levels of burstiness and our dataset drawn from real-world data, from which we conclude that MPRA provides promising and robust results.

Chapter 1

Introduction

Physical laboratories are still widely used to conduct scientific experiments over a group of people. The use of physical laboratories is a methodological cornerstone of social and behavioural sciences. However, it has significant drawbacks such as limited pool size with unrepresentative populations, substantial organizational complexity, and cost. Web-based experimental frameworks have the potential to allow researchers to develop and deploy these experiments more quickly and more efficiently.

Such a framework typically provides an environment within which user can conduct experiments by varying one or more key parameters [1]. According to Basili et al. [2], three objectives are fulfilled by experimental frameworks. First, they let users perform experiments easily. Second, they let researchers classify and discuss the experiments. And third, they let users learn from the results, and apply this knowledge to real-world problem areas.

There are a number of common approaches for the design of experimental frameworks. Several online experimental platforms focus explicitly on a single research topic, such as ”Foldit”, a protein structure optimization environment disguised as a game where players compete for the most points [3], ”SETI”, an experimental framework that can be installed on a participant's computer and analyzes radio signals that are received from space by making use of the participants' computer resources when their computer are idle with a screensaver appearance [4], or ”FaceResearch”, which contains several experiments that seek to determine the traits in portraits that make people seem attractive to others [5]. Experimental platforms such as Games With A Purpose [6], Zooniverse [7] and OntoGames [8], contain more than one type of experiment each with a different goal. Participants in these experimental platforms are able to select the experiment in which they want to participate, and are aware that they are making a scientific contribution through their participation.

A major concern for an experimental frameworks is attracting and assigning participants to the available experiments in the framework to achieve specific goals. Such goals could include the need to conduct unbiased experiments with meaningful results, or completing certain experiments within a specified amount of time. For conducting unbiased experiments, participant assignment is often randomized to preserve the internal validity of the system. More specifically in that case we form participant groups that are assigned to experiments to be similar with each other [9]. Randomization of the participant assignment process allows the obtained experiment results to be mathematically interpretable. In some scientific studies, randomization of the participant assignment process is not preferred such that the participants may be required to have some background knowledge about the scientific experiment topic or the experiment is intended to be done within a controlled environment. In such cases assigning participants according to a proposed method to optimize the execution process of these experiments i.e. by minimizing the average completion time or minimizing the average participant waiting time would be preferable.

There are various solutions in the literature to optimize participant assignment. Examples include assigning participants to experiments by giving priority to the experiments with earlier deadlines, or minimizing the average completion time of experiments. However, these methods are not optimized to assign participants to experiments having execution priorities. Another drawback of these methods is that they do not have to provide an optimal solution to the assignment problem. In the studied problem framework experiments have priorities and some of them need to finish earlier than the others no matter what their due dates are. Hence priority (i.e. weight) of experiments is an important criterion as well as their tardiness values and this implies minimizing total weighted tardiness of experiments instead of their total tardiness value. Minimizing total weighted tardiness is often regarded as scheduling the available experiments on time as much as possible while considering their priorities which are given by the researchers. Therefore we formulate the participant assignment problem that is considered in this study as a problem of minimizing weighted tardiness and our goal is to assign incoming participants to available experiments such that the total weighted tardiness of these experiments is minimized.

Our objective in this study is to optimize the participant assignment process in online experimental environments to help researchers complete their studies on time. We consider varying participant arrival and experiment availability cases for the participant assignment problem, propose a collection of optimization algorithms, evaluate them on synthetic and real datasets, and and draw conclusions about the performance of the algorithms.

1.1 Objectives

When a large numbers of experiments are being deployed, researchers might want some experiments to finish earlier than the others. Specific experiments can be given a higher priority by preferentially assigning incoming participants to experiments. To develop solutions to this participant assignment problem, we assume that each experiment in the framework is assigned a priority value. The participant assignment methods applied to the framework use these priority values to determine optimal participant assignments. The participant assignment is done to minimize the total weighted tardiness of the experiments. The advantages of this participant assignment method can be summarized as follows. From the researchers' point of view, the completion order of the experiments will meet their demands and the total time spent on these experiments is optimized according to their assigned priorities. From participants' point of view, automatic participant assignment reduces the time they will spend waiting for an available experiment.

Assuming a specific scheme for participant arrivals and release times of experiments would provide in unrealistic results. For this reason we consider four different scenarios such as uniform participant arrivals and release times of the experiments from the beginning; varying participant arrivals and release times of the experiments from the beginning; uniform participant arrivals and varying release times of experiments and finally varying participant arrivals with varying release times of experiments. This lets us to compare the studied algorithms in different environments and assess their performance.

Having considered algorithms over experimental data, in order to evaluate the gathered performance results, we study the exeuction time of the algorithms and their optimality over the synthetic dataset and volunteer science data. We also investigate the variables which make the biggest impact over the performance of the studied algorithms. This would give us insights about the performance results of the studied algorithms and understand the outcomes of the studied data. In addition to that we also assess the robustness of the algoritms in order to check whether the studied algorithms performance results are feasible.

1.2 Summary

Participant assignment is the process of assigning incoming participants to experiments. From the researchers' point of view, assigning participants to experiments lets them control the completion order of the experiments and allows them to meet the research goals. In our work we develop and evaluate algorithms that allow researchers to achieve these goals.

There are various methods in the literature to minimize the tardiness of the scheduled jobs. The most general version of the problemem considers the weights (i.e. priorities) of the experiments since they can more accurately represent the cost of the potential losses by researchers. Researchers can use the weights to ensure that higher priority experiments finish earlier than those with lower priority.

We test the performance of our heuristics over benchmark instances by comparing their solutions to tight LP lower bounds. Our results show that the execution time of the different algorithms generally match their asymptotic running times. Although some of the studied methods, such as the exhaustive algorithm and the dynamic programming algorithm, provide optimal solutions to this problem, they require a significant amount of running time. It is also observed that the some heuristics run quickly but are not always able to find an optimal solution. The results indicate that the optimality of the greedy heuristics decreases when the number of experiments and the size of experiments increase.

In addition to exact and approximation methods considered in this study to minimize the total weighted tardiness of experiments when the participants' arrival to the framework and experiment release times are uniform, we develop algorithms that can handle different participant arrival and experiment release times scenarios. We develop a synthetic data set and evaluate algorithms' execution times and solution quality performance over the created synthetic data. Moreover, these algorithms are also tested in Volunteer Science framework data to compare their performance over the gathered data in real world environment. We compare algorithms' execution times and solution quality performance in both online and offline environments.

The rest of the thesis is organized as follows. Chapter 2 provides background information about the Volunteer Science experimental framework, other experimental framework, and participant assignment methods. Chapter 3 describes our research for the participant asssignment case based on uniform arrivals and release times of experiments. In the same way Chapter 4, Chapter 5 and Chapter 6 describe our research for the participant asssignment case based on varying arrivals and release times of experiments, uniform arrivals and varying release times of experiments and varying arrivals and varying release times of experiments respectively. These chapters also include the testing of considered algorithms and their performance evaluation. Finally we conclude our study in Chapter 7.

Chapter 2

Background

The following sections are going to give a survey of the related work for our study consisting experimental frameworks, participant assignment algorithms and Volunteer Science experimental framework which is developed during this study.

2.1 Experimental Frameworks

An experimental framework consists of a structural environment for conducting scientific experiments online within which users can change the conditions of the experiments by using the features of the framework. One of the well-known online experimental frameworks is introduced by Luis von Ahn [6]. He states that computer games can be designed to attract human beings to solve open problems from the fields like security, computer vision etc. He calls these games with a purpose and believes that such games must be proven to be accurate and efficient. In his study he introduces a collaborative picture labelling game (ESP) and image labelling game (Peekaboom). In the study, the researchers combined games and online environment to demonstrate that online experimentation is possible with a large number of participants, and can be entertaining. The disadvantage of the experimental framework used in this study is its inflexibility. Although the experiments/games provided in the framework are well designed and serve some scientific purpose, the framework is not allowed to be used by other researchers to deploy their own experiments or to collect the obtained user generated data.

Time-sharing Experiments for Social Sciences (TESS) [10] is an NSF-funded platform for conducting Internet-based experiments on general populations. TESS employs time-sharing to improve the speed, efficiency and costs of the experimental process. Different studies can use the same experimental platform to develop a population of experimental subjects, select representative samples of that population, perform the experiment over the Internet, and report the results.

TESS is an important resource for the social sciences. The strength of TESS is the high sample quality it offers and its flexible experimental platform. However, the studies performed by TESS present each subject with a fixed collection of information that cannot be varied in response to a subject's decision-making. Each subject is evaluated separately by TESS; the system does not allow subjects to interact with one another during the study. While TESS can support studies from different disciplines, little variation is allowed in the experimental format. The platform can present a subject with text, a picture, or other visual information, and solicit a response, but more complex experimental designs are not possible. Furthermore, while the high quality sample is a major asset of TESS, it also comes at a substantial cost since the sample is provided by a survey company. That is, the real cost (based on the cost of the sample) of the same experiment (where possible) run with TESS will be more expensive than on the platform we propose (but drawn from a well understood population).

The Group Experiment Environment Project (GEEP), led by Robert Goldstone [11], represented an early, ground-breaking effort to observe and explain group behaviour by enabling people to play simple games together over the internet. The project first allowed people to play games, such as guessing a hidden number, drawing a hidden picture, or locating virtual food pellets, in an online environment that allows different users to interact. They then attempt to use agent-based computational models to explain the behaviour that is observed. Virtual players based on these computational models are also introduced in the games as participants.

Rafelsberger et al. [12] develop a framework to conduct/develop experimental games with a purpose by making use of social networking platforms such as Facebook, iGoogle and Netvibes. By making use of their framework they conducted two different experiments regarding to US president election that was held in 2008. The first experiment that is done is called Election Monitor which is a user poll focusing on US 2008 election that invites users to send their votes for candidates and also asks users' opinion of how likely it is that their preferred candidate will win the election. The other experiment is called Sentiment Quiz which asks the players to evaluate the sentiment of provided sentences regarding to 2008 election by providing answers in likert-scale. In order to prevent the biased answers in the experiments, participants' identities that are collected from social platforms are hidden and each participant is assigned a trust value (determined by asking simple questions during the experiments) to determine the impact of their answers. By conducting these experiments authors were capable of optimizing the existing methods for automatically detecting the sentiment of sentences from an archive of election-related news media articles. By making use of social network visualization tools such as Rhizome, they were also able to visualize the user behaviour and disseminate information within social networks. These experiments also enabled them to reveal different perceptions and interpretations of content depending on the readers' political orientation.

Matyas et al. [13] describe the design and experimental evaluation of a location-based mobile game called CityExplorer. The goal of the game is to produce geospatial data that would be useful for location-based services. In the game, players take geographically referenced photos and categorize them semantically. CityExplorer has the same working principles of the board game Carcassonne in which at each turn, a player draws a new terrain tile from faced down tiles and places them adjacent to previously placed tiles so that roads and cities connect. Players can leave markers on the tiles, and at the end of the game the winner is decided according to the number of markers that each player has on the placed tiles. In CityExplorer, player are able to use mobile devices with gps technology and the experiment's website to send images to the system. The experiment is applied to two cities in Japan. Experimenters check the integrity of each photo by validating it against those submitted by two other participants. The results of this experiment show that it is possible to implement a location-based mobile game with a purpose is possible, and use it to collect a large quantity of geospatial data.

Siorpaes et al. [8] believe that a game that serves for a purpose should contain fun and intellectual challenge. They state that creating games with a purpose would result in challenges such as identifying relevant tasks, designing game scenarios, attractive interface, preventing cheating, fostering user participation, deriving semantic data, efficient distribution of labor, and scalability and performance. They describe ontograms, a category of games that serve to classify and label words in an entertaining way. For example, one ontotube game lets users watch a movie and then label it with a set of provided labels. If a user's label matches the one given by others, the users earn points. In the ontopronto game, users are presented with a random wikipedia article and asked to label its content. In SpotTheLink, players are presented with a target word and a set of possibly related words. Users select the word they think is most related to the target word, and receive points for making the best choice.

All of these games can either be played with a partner or in single-player mode. According to the survey results that the authors collected from participants, participants are content to provide data using these experiments and they enjoyed participating.

Jensen et al. [14] describe their findings about some existing data mining techniques that are used over social networks. They state that many of these techniques share a common set of statistical challenges and design issues and discuss these issues by making use of one of their existing techniques, PROXIMITY (a method for relational learning) as an example that they adopted to make a statistical analysis over the IMDB database. Their method is based on representing the movie database as a tree where movies, studios, people and awards are represented as nodes. Then, by making use of the query language QGraph, they are able to query this database to obtain subgraphs related to their queries. They then use relational probability trees in order to make predictions. Using this method, the authors are able to predict the probability that a movie's opening weekend box office receipts will exceed 2 million.

Chang et al. [15] used a Facebook-based questionnaire to study factors that affect a user's interest in using social network games. The researchers use concepts like personal involvement, intrinsic motivation (i.e. perceived enjoyment), and extrinsic motivation (perceived ease of use and perceived usefulness) to explain usage intentions of social network games. Using MANOVA hypothesis testing, the researchers conclude that these terms have a positive influence on participants use of social network games. They find that personal involvement has a significant effect on perceived enjoyment, perceived usefulness, and perceived ease of use. They also find that perceived ease of use has a significant role in explaining usage intention.

Brennan et al. [16] evaluate the effectiveness of two different approaches to using web based surveys to attract participants to experiments. Their method of attracting participants is based on emailing the advertisement of the experiments to the potential users by making use of newsgroup lists and promoting participation by announcing an incentive prize such as gift check. In the first survey experiment they conducted was about estimating the demand for new billing options of a local internet provider. The second experiment was about evaluating the effectiveness of sound and animation in banner ads. Questions in the experiments were created dynamically from a pool. The system ensured that the time required to complete the survey is short and every question is answered when the experimentation was completed. CGI scripts were used to create the dynamic surveys and whenever participants submit an answer their answer is emailed to the principal investigator. In order to bind a received email to a participant, authors used browser cookies to collect user identity. At the end of the experiments, authors realize that binding answers to participants by making use of cookies is not a good idea since some participants may not allow cookies, and some browsers might be incompatible. The researchers determined that participation was higher in the first days of the experiment. Overall, participation was very low and no solution to this problem was proposed.

Mason et al. [17] investigate the effects of collaboration in problem solving on the performance of individuals. In order to do this analysis, they conducted a series of web-based experiments in which groups of 16 individuals collectively solve a complex problem and share information through different communication networks. The experiments were web-based game called 'wildcat wells' and participants were compensated using the Amazon HIT system. In this game the participants' aim is to explore a realistic-looking 2D desert map in search of hidden oil fields. At the end of each round, players are able to see their scores as well as other participants' scores, and are paid in proportion to their accumulated points. The game is played under 8 different network topologies and all networks comprised n=16 nodes and k=3 neighbours.

The researchers found that higher local clustering in networks resulted in a higher tendency for players to copy their neighbors. This result was expected by the authors since local clustering allowed an individual's neighbours to see each other's locations, thus increasing their likelihood of copying.

Mason et al. [18] investigate the impact of financial incentives on the performance of experiment participants. Experiment participants are composed of people that are gathered together by making use of web-based crowd-sourcing. The researchers use the Amazon Mechanical Turk (AMT) web-based experimental environment which rewards participants with a small amount of money as compensation for their efforts. They state that AMT has advantages over real-life laboratory experiments such as affordability of the conducted web experiments and elimination of undesired effects such as unwanted group interaction. In order to understand the outcomes of this financial rewarding system authors conducted two different experiments. In both of the experiments, they measured performance in terms of quantity of work performed and the quality of work.

The first experiment is about ordering images that are taken from a traffic camera. Participants' output performance is measured in terms of number of experiments they successfully completed and their accuracy is measured as proportion of correctly ordered images. Participants initially go through a trial experiment and based on their performance, they work on real experiments with varying cash rewards changing from 0.01 to 0.1. At the end of the experiments it is seen that the participants who earn more in experiments are eager to do more experiments, thus increasing their output. However, authors find that accuracy is not related to the size of the reward and is nearly same for low- and high-earning participants. In order to ensure that these findings are not affected by the specific experiments, the authors conduct a second experiment based on a word puzzle. The results of this experiment are similar; participants who earn more; complete more experiments but accuracy is nearly unchanged across all levels of earners. At the end of the experiments, participants are asked if the reward they received was fair. Based on the responses, the authors conclude that anchoring effect may be the reason of this unexpected accuracy result. In other words authors think that participants set their mind to initial low payments they earn from experiments and do not take the participation as a serious job.

Pastor et al. [19] created an XML-based framework called RELATED for the development of internet experiments in an easy manner. According to their approach, XML files are used to connect the laboratory elements, conduct the remote system access, and define different experiments within the framework. Therefore, the system lets teachers to create and publish new internet based experiments regardless of their location. The framework is mainly based on Java to parse and execute the appropriate XML file. The details of the experiments (modules, parameters, GUI details) and variables are defined in these XML files. Similar to the approach used in Volunteer Science they are making use of static variables (that are defined prior to start of experiments) and dynamic variables which can be modified during the experiment. Although the framework they devised makes use of internet for laboratory experiments, it is not developed for public use, therefore the user interface is poor and users of the system fill in the XML files manually without making use of any web site/interface for teachers. Authors list the benefits of the developed framework as its flexibility since they design it solely for experimentation on XML and Java, the increase in the available set of activities that the teachers can use for training the students, and the special XML-based mark-up language that they are using to define the standards for running online experiments.

Reips et al. [20] present a web-based experiment generation framework, WEXTOR, implemented using Javascript. Based on inputs from the user, the framework dynamically creates both the customized web pages that are needed for the experimental procedure and the experimental user interface. Authors believe WEXTOR would be a useful tool for experienced experimenters who know less about internet technology or for experienced web experimenters who seek a fast way of creating the experiments. Experimental design is implemented using a step by step process within the framework. Users are prompted to give detailed information regarding experiments that they are planning to conduct, such as the name of the experiment, the number of questions, and whether it contains quasi-experimental (natural) factors (e.g. participant's age) etc. The drawback of the system is that it is solely designed for and experimentation process of the researchers. It does not have a friendly user interface and thus lacking entertaining features to attract online participants over internet. Moreover the framework is old and is not able to accept participants over popular social networks.

Keller et al. [21] created a Java-based web experiment automation tool called WebExp which consists of two main parts. The first part, the WebExp server, is located at the web server that hosts the experiment. The second part, the WebExp client, is located on participant's machine and it is responsible for connecting to the server, downloading experiment details, and storing participants' answers to questions Participants are authorized by email validation and experiment material can be randomized for each participant. The tool is also able to create multiple variants of the experiment and ensure that each variant is delivered to an equal number of participants. The framework records the response times of the participants in the experiment. At the end of each online experiment, the WebExp client connects to the WebExp server and sends each participant's response file. Webexp can also create offline questionnaires which can be downloaded in latex format and printed out to form paper-based questionnaires.

2.2 Participant Assignment

Assigning participants to groups or experiments is an important stage of the research progress and can be accomplished in many different ways. According to Clifford et al. [22], participant assignment process is important for maintaining the internal validity of studies. They state there are two main ways of assigning participants to groups. First, participants can be assigned to groups randomly. They state by using random assignment, researchers can establish group equivalence. Assignment by chance can be done in two ways. If the total sample of participants is already identified by the researchers prior to assignment process, participants are 'captive', and this case of random assignment is called captive assignment since from the beginning it is known which participant will be assigned to each group. If the researchers do not know the participants in advance, the random assignment method is called sequential assignment. In this case, researchers do not know which participants will be selected from the participant pool and will contribute to the studies. The second way of assigning participants is using controlled methods. In this case, the researcher applies certain non-random heuristics to assign participants to experiments. One approach to controlled assignment is experimental matching, in which researchers assign participants to groups according to data that is available prior to the experiment. For example, conducting different experiments for different age groups can be accomplished with experimental matching. According to the authors, another approach to controlled participant assignment is assigning participants to experiments according to their prior experimental task performances. In this way, researchers can observe effects of different experiments on the performance results of each individual participant.

Evans et al. [23] indicates that the participant selection (i.e. sampling) process is helpful in representing the collective behaviour. They state that this sampling process can be done in several ways. Along with random sampling, where each participant has equal chance to be selected, participants can be assigned using systematic sampling in which for a selected variable k, every kth individual is assigned to a selected experiment. In stratified sampling, the population is divided into different groups prior to the sampling process according to their characteristics, and participants are then randomly selected from each group. In cluster sampling, the population is clustered and then participants within clusters are all assigned to a specified experiment. In multi-stage sampling, a multi-stage clustering process takes place where smaller clusters are formed from larger ones and randomly selected participants are assigned to experiments. In convenience sampling, participants which are readily available are selected. In quota sampling, the available participants who fit a particular criterion are chosen and in referral sampling, participants are selected according to other participants' referrals.

Web-based experimental frameworks could be a good way of conducting large-scale experiments. However, recruiting participants to make experiments start can be a very time-consuming part of the process. To speed things up, researchers usually use a range of approaches for attracting people. One approach is to support participation by making use of financial incentives either directly, or by using other web sites such as Amazon Mechanical Turk. Another approach is pre-study promotion of the framework, including presentations, special arranged meetings etc. From the point of view of experimental design, as Fisher [24] notes in his study, participant assignment in experiments can be done randomly to eliminate the bias within the experiments, allowing the samples to truly represent the target population. Although random assignment of participants can be a good way creating a representative target population sample, it does not guarantee that all available experiments will be completed as quickly as possible.

The problems of minimizing the total tardiness and total weighted tardiness problems in single machine and multi-machine environments are shown to be $NP - hard$ [25, 26]. There are various solutions in the history for solving minimum tardiness problem such as [27] and [28] that are based on TABU search and simulated annealing. The methods for solving the minimum weighted tardiness are limited in the literature and the available ones are heuristic approximation algorithms such as [29] and [30] which uses branch and bound and ant colony optimization methods for the solution of this problem. In the literature minimizing total weighted tardiness problem is often regarded as a scheduling problem and there are many real life areas where it is used such as industrial facilities and printing factories [31]. However, none of these studies considered the topic as a problem of participant assignment as it is studied in this research. The methods considered for this problem in this study are based on exhaustive enumeration, greedy, dynamic programming, and integer linear programming. While some of these methods provide guaranteed optimal solutions to this problem with the long execution time drawback, some quite fast approximation algorithms are also provided. The algorithms are then successfully tested on the created synthetic data.

2.3 Volunteer Science

Collective behaviour has become much more popular due to recent technological advancements. Humanity's increasing connectedness and internet have significant affected this popularity. Individuals and groups interact with each other, often through cooperation or competition. The classical approach to conducting experiments in the cognitive and social sciences is lab and field experiments. Many universities have dedicated shared laboratory space that allows the spreading the benefits of a physical lab among the community of experimental scholars. Conducting scientific experiments in a university laboratory environment has disadvantages, such as unrepresentative samples, the integrity of collected data, institutional limitations, control factors, and experimenter bias. The World Wide Web offers a good alternative to these laboratory-based scientific experiments by providing a vast amount of participants to the experiments.

The Volunteer Science experimental framework was developed as a response to the increasing need for online social experiments. The objective of the framework was to provide a framework to the researchers where they can easily deploy experiments and obtain the analyzed user-generated data for their scientific studies. From the participants' point of view, volunteer science will let them to participate in entertaining scientific experiments quickly without dealing with registration details and long experiment waiting times. Moreover Volunteer Science allows its users to earn awards according to their performance in the experiments. Completion time of the experiments and the number of users participated in the experiments are two important criteria that affect the outcome of experiments. Volunteer Science [32] experimental environment is designed to cope with such important aspects of the experiments by moving these experiments onto the internet.

Volunteer Science is a web based experimental environment that makes use of the current popular web development methods and social networking environments to conduct and gather experimental data. The platform is intended to enable researchers to design and manage a variety of experiments without worrying about the implementation details. Volunteer Science lets researchers easily conduct experiments within a web environment. The platform supports experiments over individuals or groups, making it a good environment for conducting experiments that are based on cognitive science to investigate individual-group behaviour and collective outcomes.

Since well-functioning groups are work best when motivated by their own self interests [33], we are working to make Volunteer Science an attractive destination for participants. We use financial incentives for better crowd sourcing, as described by Mason et al. [18]. As described above, examples of internet-based experimental frameworks exist, such as TESS [10], GWAP [6], and GEEP [11]. These online experimental frameworks provide certain advantages over the laboratory-based experimental platforms. Online frameworks more easily support larger pools of participants, and experiments can be deployed more easily and quickly. Although these platforms have successfully ported laboratory experiments to internet environment, they are less flexible and do not connect to social networking sites. In Volunteer Science, researchers form study groups, conduct experiments with the other researchers and they are able to make real-time tracking of each collective decision-making experiment. In addition, subjects can exchange messages and instantly see decisions being made by their neighbours. Another difference is that Volunteer Science allows experiments to be easily customizable by researchers. Researchers are active users of the system and they are able to introduce new experiments in the framework quickly and easily.

Chapter 3

Participant Assignment with Uniform Arrivals and Release Times

Web-based experimental frameworks such as Volunteer Science are designed to handle large numbers of participants. This raises logistical problems, such as how to efficiently assign participants to available experiments to reduce participants' average waiting times, and how to complete higher priority experiments earlier than others. In this study, we focus on optimizing the participant assignment process in such a way that the total weighted tardiness as a result of assignment of participants to the experiments is minimized.

In the following sections, we formulate this problem and describe our proposed solution techniques. We also describe our preliminary experimental results.

3.1 Formulation of Optimized Participant Assignment

We assume that a sequence of participants join the framework one at a time in some order and we know the sequence in advance for the algorithms defined in this proposal. Participants can be assigned to an available experimental instances one at a time. We assume we are given a set of N experimental instances with a total instance size of K, an experiment instance processing time which refers to the execution time of an experimental instance, an experimental instance due date which refers to the desired latest completion time of an experimental instance, and an integer priority between 1 and 9 for each experiment instance which refers to the importance of each experiment instance to the researcher.

We assume that each participant is assigned to an experimental instance in the participants' arrival order. We assume that the number of participants currently in the system increases by one for every time period t. If at least one experimental instance is not started when a participant is being assigned, that participant will be successfully assigned to some experimental instance. Each participant can be assigned to no more than one experimental instance. If a participant is assigned to an experimental instance, s/he will never be unassigned or assigned to another experimental instance. If all experimental instances are complete when a participant is being assigned, that participant and all subsequent participants remain unassigned. A solution to the assigment problem exists only if the total number of arriving participants is enough to make all experiments start.

Each experimental instance is available at the same time. However, an instance only starts when the number of participants assigned to that instance equals the instance size. An instance's completion time is time at which the instance has finished processing, which is the start time plus the processing time. The tardiness of an experimental instance is the positive difference between the instance's completion time and due date, i.e.

$$Tardiness_i(t) = \begin{cases} 0 & if \quad t + PT_i \leq DD_i \\ t + PT_i - DD_i & Otherwise \end{cases}$$

Here, $Tardiness_i(t)$ is the tardiness value for some experiment i that starts at time t, assuming it is completed. Experiment i starts at time t, runs for a period of time that is equal to its processing time (PT), and it has a due date (DD). If the experiment is completed before the due date, then the tardiness of that experimental instance is zero.

The weighted tardiness of an experimental instance i that starts at time t can be calculated by multiplying $Tardiness_i(t)$ by the priority value (PR) of experiment i, i.e.

$$WT_i(t) = Tardiness_i(t) \times PR_i$$

A participant assignment is a tuple (x, y) where x is a participant, y is an experiment for $x \leq K$ and $y \leq N$. An assignment is $complete$ when the correct number of participants is assigned to experimental instances to make them start. Our goal is then to find an $optimal$ assignment such that it is $complete$ and the total weighted tardiness of the experiments is minimized. An optimal solution G that we are looking for is therefore a set of tuples which are formed by considering the priorities of the experiments while calculating the total tardiness.

3.2 Background

The single machine total minimum tardiness problem has been studied extensively and has been proven to be NP-hard [34]. Similarly, for the case of single machine total minimum weighted tardiness problem, its complexity is shown to be NP-hard [25, 26]. For the cases of parallel machines, total minimum tardiness and total minimum weighted tardiness problems are also known to be NP-hard [35, 36, 26]. Nevertheless, the exact complexity of this problem is not known and remains open [36] .

For minimizing total tardiness in a multimachine environment, a significant amount of work is done in the literature. Minimizing the total tardiness in parallel machines is also called the hybrid job shop problem which is an extension of the classical job scheduling problem and allows an operation to be processed by any of the available machines from the provided set to minimize total tardiness [31]. Scrich et al. [27] proposed two heuristic methods for minimizing total tardiness in a multimachine environment. The algorithm they adopted in their method was tabu search. One of their methods was based on a hierarchical procedure and another was based on a multiple start procedure. These procedures use dispatching rules to find an initial solution and then search for improved solutions. The authors reported diversification strategies and the test results of their proposed algorithm.

Loukil et al. [28] describe a heuristic based on simulated annealing for minimizing total tardiness in multimachine environments. Their study was based on a real-world case study concerned withminimizing total tardiness for effective product scheduling. They considered many constraints in their method and their method was composed of two steps: production of several sub-products and assembly of a final product. At the end they considered different objectives simulataneously such as total tardiness, mean tardiness, and mean completion time.

Alvarez-Valdes et al [37] describe the design and implementation of a scheduling system for a glass factory. Their goal was minimizing tardiness using a heuristic method based on a two-step integer-linear programming process. In the first step they find an initial feasible solution, and then this solution is improved by minimizing the primary cost function and the secondary objectives. According to their experiment results their heuristic works fast and provides results in very short computing times.

Although there are many studies of the problem of minimizing total tardiness on parallel machines, as stated above, fewer studies consider minimizing the total weighted tardiness on parallel machines. Liaw et al. [29] consider the problem of scheduling a set of independent jobs on unrelated parallel machines to minimize the total weighted tardiness. They propose a branch and bound based heuristic for the problem. According to their algorithm, each node in the search tree is a partial schedule and a node in level r is a partial schedule with r jobs scheduled. The algorithm they propose uses dominance rules to eliminate unpromising partial schedules during the search process. They found that their branch and bound algorithm performs well on up to 18 jobs and 4 machines. The algorithm is a fast heuristic due to its heuristic nature. It uses dominance rules to eliminate subtrees while finding the order of assignments to shorten the execution time. Therefore, unlike the many algorithms presented in this study it does n t necessarily find an optimal solution for minimizing the total weighted tardiness.

Zhou and Hong [30] proposed a heuristic solution to the problem of scheduling given set of independent jobs on unrelated parallel machines to minimize the total weighted tardiness by using a modified ant colony optimization algorithm. The computational experiments that they conducted show that their proposed method outperforms the general ant colony optimization algorithm. Although the method explained here seems a fast alternative to the minimizing weighted tardiness problem, the proposed algorithm uses ant dispatching rules to find the total minimum weighted tardiness in reasonable time periods and it does not guarantee to find the optimal solution to the problem.

Monch [38] describe a heuristic based on ant colony optimization to solve the total weighted tardiness problem on unrelated parallel machines. In this study, a colony of ants is used to construct iterative solutions of the scheduling problem using heuristic information. For computation of the heuristic information, apparent tardiness cost dispatching rules are used. The test results based on the stochastically generated test instances show that the algorithm performs well over the data set. The researchers compared their method to a genetic algorithm method for finding minimized total weighted tardiness. They conclude that their method is faster than a genetic algorithm. However, since both of the compared methods are heuristics, optimal solutions were not found in eithercase.

Souayah et al. [39] describe a branch and bound based algorithm to minimize total weighted tardiness on parallel machines. Their method is a heuristic based on Lagrangian relaxation. According to their experimental results they find optimal solutions in some cases within a reasonable amount of CPU time. Although the algorithm they proposed is a heuristic, their method is slow and their tests were conducted over 24 jobs and 3 machines. Moreover, the accuracy of their proposed method seems worse than the many other proposed heuristics for this problem in the literature.

3.3 Algorithms for Optimized Participant Assignment

To minimize total weighted tardiness, four methods are proposed in this study that are based on exhaustive enumeration, dynamic programming, greedy approaches, and integer-linear programming. In the rest of this section, the working principles of these proposed solution approaches are explained in detail. In Table 3.1 we provide the acronyms that we will be using throughout this thesis.

For each scenario considered throughout the thesis, we validated the correctness of our algorithms. For exact algorithms, we created small instances with known answers and checked whether our algorithms find the correct results. For heuristics, we created small instances, found the answers ourselves by applying the rules of heuristics, and checked whether our algorithms find those answers.

Table 3.1: List of algorithm acronyms, names and behavior.

Acronym	Name	Summary
Exhaustive	Exhaustive enumeration	Finding all the possible solutions with brute force approach.
DP	Dynamic Programming	Finding the exact solutions by making use of partial solutions.
EDD	Earliest Due Date	A heuristic that assigns the participants to the experiments with earliest due date first.
LCL	Least-Cost-Last	A heuristic that assigns the participants to the experiments with smallest weighted tardiness as last.
ATC	Apparent Tardiness Cost	A composite heuristic that makes use of Minimum Slack and Weighted Shortest Processing Time algorithms to find the minimum weighted tardiness.
ATCPA	Apparent Tardiness Cost for Participant Arrival	A composite heuristic that makes use of Apparent Tardiness Cost, and Earliest Due Date algorithms to find the minimum weighted tardiness.
ILP	Integer Linear Programming	An exact method in which the objective function and the constraints are linear.
MS	Minimum Slack	It is a scheduling heuristic which assigns priority based on the slack time of a process.
MSP	Minimum Slack with Priority	It is the division of slack to experiment's priority.
NF	NextFit	Assigning participant to the next available experiment.
SCR	Smallest Critical Ratio	It implies that the next experiment to be processed is the one that has the lowest critical ratio.
SCRP	Smallest Critical Ratio with Priority	It is the division of critical ratio to experiment's priority.
SIRT	Smallest Instance Size with Remaining Time	It is the multiplication of experiment's instance size with the remaining time.
MPRA	Maximum Priority with Remaining Attributes	It is the division of priority to remaining time times experiment's processing time times remaining experiment instance size.

3.3.1 Exhaustive enumeration

The first algorithm we consider is exhaustive enumeration (i.e. brute force). This approach goes through every possible experimental assignment for each incoming participant in order to find the optimal permutation of participant-experiment assignments while satisfying the contraints. Pseudo code of this approach is shown in Algorithm 1 below.

The recursive exhaustive algorithm takes as input the number of completed experiments, an array of experiments, instance size, processing time, due date and priority. The algorithm also takes an array containing the permuted order or participant-experiment assignments, a globally defined value to keep the found minimum weighted tardiness value and a globally defined array to keep the correct permutation that gives the minimum weighted tardiness value.

In the given pseudo code, line 1 ensures that the recursion runs until all experiments are completed. Lines 2-8 traverse the experiment array and recursively allocates each experiment with the current participant as long as we can allocate the experiment. In lines 10-11 for each completed permutation of participant-experiment assignment, the weighted tardiness of the $complete$ assignments is calculated and checked to see if it is the minimum weighted tardiness that has been found so far.

Algorithm 1 Exhaustive

```
Require: count of started experiments, experiments array
 1: if count of started experiments $<$ number of $experiments$ then
 2:     for every experiment $i$ do
 3:         Assign the next participant to the any empty slot of experiment $i$
 4:         if Assignment process results in the experiment to start then
 5:             Increment the count of started experiments
 6:         end if
 7:         $Exhaustive$(count of started experiments, $experiments$)
 8:     end for
 9: else
10:     Calculate the tardiness value of the found assignment order
11:     Store the minimum found tardiness value
12: end if
```

The base case occurs when all the experiments are completed. Then, if K is the number of assigned participants, the overall function runs within a polynomial factor of $O(K!)$ in the worst case, the factorial of the number of total assigned participants, assuming every experiment requires at least one participant to start. Since the worst-case complexity is based on factorial of K, this algorithm is very inefficient.

3.3.2 Dynamic Programming Algorithm

The exact solution for the problem of minimum weighted tardiness can be found with an improved worst-case running time if we use a dynamic programming approach. The solution that will be described here is a modified version of the Held-Karp algorithm [40] which was is originally used to solve the well-known permutation-based problem called travelling salesman problem.

Structure of the Optimal Solution: Let S be a sorted sequence of experiment numbers, and where the number n_s of consecutive duplicate values of s is less than or equal to the size of experiment s. In the initial experiment sequence, n_s equals to the size of each experiment. Given an optimal solution G for the minimum weighted tardiness problem, we can select value i such that in this solution, the first participant is assigned to experiment i and the remaining participants are assigned to experiment numbers from S. For a given a sequence of experiment numbers E, we define $S - E$ or a $subsequence$ to be the sequence S with the experiment numbers in E removed. Experiments included in a $subsequence$ are not necessarily in a sequence in S, but should be in sorted experiment order. We can select another value j from S such that the second participant is assigned to experiment j and the participants that come after the second participant according to arrival order are assigned to experiments from $S-E$ with $E = (j)$. If we remove the first participant which is assigned to experiment i from the optimal solution, $G - \{(1, i)\}$ will be an optimal solution to the resulting subproblem starting from the second participant which is assigned to experiment j by assigning the rest of participants to the experiment numbers from $S - E$ with $E = (j)$. If the removed participant was the only remaining participant for that experiment to start, the value of the overall solution will then become $WT_i(0)$ plus the optimized total weighted tardiness value of the sub problem. Otherwise the value of the overall solution will be equal to the optimized total weighted tardiness value of the sub problem. Proof of this statement can be shown by contradiction. If we assume there is a better way of assigning participants to available experiments starting from second participant, then we could select that assignment permutation and this would result in a better solution than G to the participant assignment problem which is a contradiction.

Recursive Definition of the Optimal Solution: Given S, experiment number i and time t; $G(i, S, t)$ is the total minimum weighted tardiness assuming that the participant arriving at time t is assigned to experiment i and the rest of participants are assigned to experiment numbers from S. Then we can recursively define the optimal solution as:

$$G(i,S,t) = \begin{cases} WT_i(t) & \text{if} \quad S \text{ is empty} \\ min_j\{G(j, S-(j), t+1)\} & \text{if} \quad i \text{ in } S \\ WT_i(t) + min_j\{G(j, S-(j), t+1)\} & \text{if} \quad i \text{ not in } S \text{ and } S \text{ is not empty} \end{cases}$$

The above statement states that there are three cases to be considered to determine the total minimum weighted tardiness of all experiments where the first participant is assigned to experiment i. The base case is $WT_i(t)$ if the assignment of the incoming participant at time t to experiment i

is the very last assignment so that S is empty. If the incoming participant at time t does not result in experiment i starting, it means experiment i still needs participants to be assigned and hence the value of $G(i, S, t)$ equals the solution value of total minimum weighted tardiness problem starting by assigning the incoming participant at time $t + 1$ to experiment j and by assigning the rest of participants to the experiment numbers from $S - (j)$. If the incoming participant at time t causes experiment i to start (i.e. the number of participants needed for experiment i is satisfied), then the optimal solution value equals to addition of the weighted tardiness value of experiment i plus the solution of the total weighted tardiness problem starting by assigning the incoming participant at time $t + 1$ to experiment j and by assigning the rest of participants to the experiment numbers from $S - (j)$.

Computing the Value of the Optimal Solution Bottom Up: Pseudo code for the dynamic programming solution to minimizing total weighted tardiness is provided as Algorithm 2 below. Given the experiment numbers, our goal is to find

$$min\{G(i, S, 0)\} \ \forall \text{ experiment number } i$$

The base case is handled by lines 3-5. Then the rest of the cases where $|S| \geq 1$ are handled in lines 6-15. When $|S| \geq 1$, for every experiment j within each subsequence S which does not involve the investigated experiment j, we assume the incoming participant is assigned to j and the rest is assigned from the current investigated subsequence of S at time t. Then if experiment j is included in the current investigated subsequence of S, we iterate over each experiment k of investigated subsequence to find the total minimum weighted tardiness value that starts from time $t+1$ and set it to be the same value for the participant assignment starting from time t. If experiment j is not included in the subsequence, then it means experiment j is launched and the total minimum weighted tardiness at time t is the weighted tardiness of experiment j starting at time t plus the total minimum weighted tardiness value that starts from time $t + 1$.

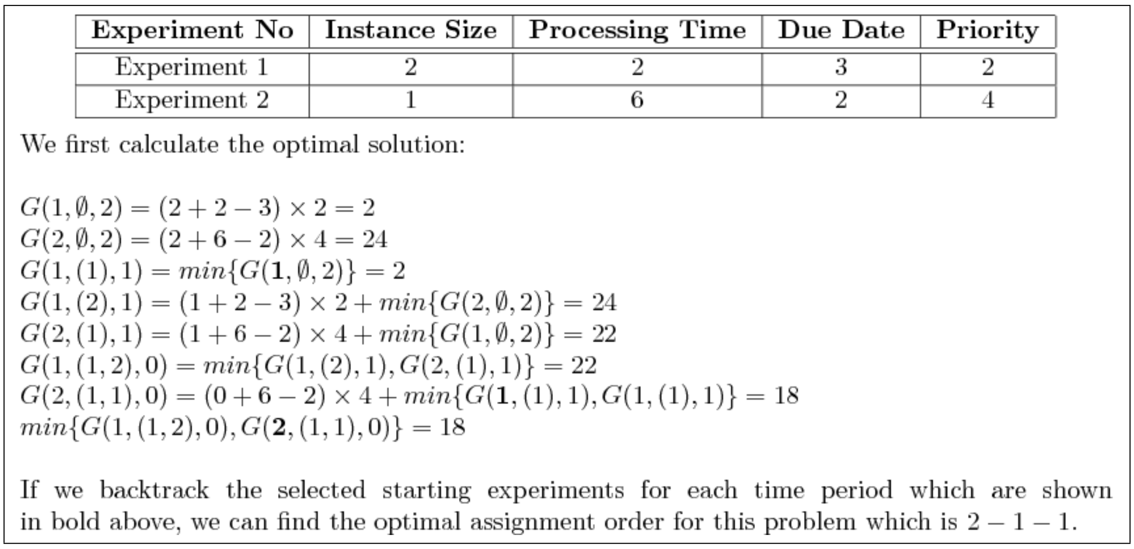
Constructing Optimal Solution from Computed Information: Having identified $(1, i)$ at time $t = 0$ that gives the optimal tardiness value for every experiment i, we can print the optimal participant assignment order by finding the value of experiment j that minimizes the expression and gives the next assignment for each participant assignment and time period t.

A sample calculation of the optimal participant assignment order is given in Figure 3.1 and the pseudo code of the printing optimal assignment scheme is given in Algorithm 3. The algorithm makes use of $optimalPath$ which is a globally defined array of size K that contains the optimal

Algorithm 2 Dynamic

```
Require: experiments array, S sequence
 1: Initialize $K$ as the number of participants needed to make all experiments start
 2: Initialize $G$ as empty to hold weighted tardiness values
 3: for every experiment $i$ do
 4:     Set $G(i, \emptyset, t) =$ weighted tardiness of experiment $i$ assuming it finishes at time $t = K - 1$
 5: end for
 6: for every subsequence of $S$ do
 7:     for every experiment $j$ from $S$ do
 8:         Set time $t = K - 1-$size of subsequence
 9:         if $j$ is in subsequence then
10:             $G(j, subsequence, t) = min\{G(k, subsequence - (k), t + 1)\}$ $\forall$ $k$ in subsequence
11:         else
12:             $G(j, subsequence, t) =$weighted tardiness of $j + min\{G(k, subsequence - (k), t + 1)\}$ $\forall$ $k$ in
                subsequence
13:         end if
14:     end for
15: end for
```

assigned experiment number values for each time period t.



Experiment No	Instance Size	Processing Time	Due Date	Priority
Experiment 1	2	2	3	2
Experiment 2	1	6	2	4

We first calculate the optimal solution:

$G(1, \emptyset, 2) = (2 + 2 - 3) \times 2 = 2$
$G(2, \emptyset, 2) = (2 + 6 - 2) \times 4 = 24$
$G(1, (1), 1) = min\{G(\mathbf{1}, \emptyset, 2)\} = 2$
$G(1, (2), 1) = (1 + 2 - 3) \times 2 + min\{G(2, \emptyset, 2)\} = 24$
$G(2, (1), 1) = (1 + 6 - 2) \times 4 + min\{G(1, \emptyset, 2)\} = 22$
$G(1, (1, 2), 0) = min\{G(1, (2), 1), G(2, (1), 1)\} = 22$
$G(2, (1, 1), 0) = (0 + 6 - 2) \times 4 + min\{G(\mathbf{1}, (1), 1), G(1, (1), 1)\} = 18$
$min\{G(1, (1, 2), 0), G(\mathbf{2}, (1, 1), 0)\} = 18$

If we backtrack the selected starting experiments for each time period which are shown in bold above, we can find the optimal assignment order for this problem which is $2 - 1 - 1$.

Figure 3.1: Calculation of optimal participant assignment order.

Running Time and Space Requirements: The "Dynamic" procedure has a main loop that runs K times where K is the summation of experiments' instance sizes. In each iteration of i, the algorithm finds sequence combinations of S with size i which is an $O(K \times 2^K)$ operation.

Algorithm 3 printOptimalAssignment

Require: G, S sequence, t, K

```
 1: if $t \geq K$ then
 2:     return
 3: end if
 4: Initialize $minVal = \infty$
 5: Initialize $minSubSequence$ list
 6: for $i = 1 \rightarrow$ size of $S$ do
 7:     Initialize $subSequence$ list
 8:     for $j = 1 \rightarrow$ size of $S$ do
 9:         if $i \neq j$ then
10:             Add $S[j]$ to $subSequence$
11:         end if
12:     end for
13:     if $G(S[i], subSequence, t) < minVal$ then
14:         $minVal = G(S[i], subSequence, t)$
15:         Clear $minSubSequence$
16:         Fill in $minSubSequence$ with contents of $subSequence$
17:         $optimalPath[t] = S[i]$
18:     end if
19: end for
20: if $t = 0$ then
21:     Print ”Minimum total weighted tardiness is ”$+minVal$
22: end if
23: Print $optimalPath[t]$
24: $printOptimalAssignment(minSubSequence, t + 1, K)$
```

Thus the overall procedure has a running time of $O(K^2 \times 2^K)$. Compared to the brute-force solution of this problem, which was shown to have a worst case running time of $O(K!)$, a significant improvement is achieved. The function needs $O(K \times 2^K)$ space to store the structure of G. Function "printOptimalAssignment" is a recursive procedure and it runs recursively K times doing $O(K^2)$ work in the worst case in each recursion. Thus it, has a running time of $O(K^3)$. The procedure fills in $optimalPath$ array which has size K, hence its space requirement is $O(K)$. Thus, the total running time of the dynamic programming approach is $O(K^2 \times 2^K)$ and the total space requirement is $O(K \times 2^K)$.

3.3.3 Greedy Algorithms

The problem of minimizing total weighted tardiness is not a subset selection problem and it does not possess the greedy-choice property [41]. As a result, finding a local minimum does not necessarily lead to finding a global minimum and a greedy algorithm may not result in an optimal solution. Greedy algorithms can be used as heuristics. This problem looks similar to the well known Activity Selection problem [42] which can be solved by sorting due dates of activities in increasing order. In contrast, our problem has more properties to be considered such as start times of experiments, processing times, and instance sizes. We will therefore describe a collection of greedy solutions to this problem.

Greedy Algorithm based on Earliest Due Date: One greedy approach assigns incoming participants to the experiment with the earliest due date (EDD) which makes it more likely that experiments with earlier due dates will run earlier than the other experiments. Since the algorithm is designed for a multi-machine environment, it assigns each new participant to the available experiment with the earliest due date.

Pseudo code for the algorithm is shown in Algorithm 4. The algorithm requires an array of experiments that lists each experiment's instance size, processing time, due date, priority, and experiment number. In case two experiments have the same due date, the algorithm assigns incoming participants to the experiment with the highest priority.

The algorithm has two main steps: sorting the experiments and outputting the final assignment order with its weighted tardiness value. Sorting takes $O(N \times log(N))$ time for N experiments with a sorting algorithm such as heap sort [42].

The second step runs in $O(K)$ time to fill in the $K-$sized $orderArr$ which keeps the found participant assignment permutation, and to calculate the total weighted tardiness value. There-

Algorithm 4 EDD

```
Require: experiments array
1: Sort $experiments$ according to due date in ascending order
2: Sort $experiments$ with same due dates according to priority in descending order
3: Sort $experiments$ with same due dates and same priorities according to instance size in ascending order

4: Assign the next participants to the next experiment in sorted order that has not been started
5: When all experiments finish, calculate the total tardiness as total minimum tardiness
```

fore the overall procedure runs in $O(N \times log(N) + K)$ time with a space requirement of $O(K)$.

Greedy Algorithm based on Least-Cost-Last: The greedy algorithm based on least-cost-last (LCL) follows the idea that since one of the experiments will start last, we can select the experiment with the lowest weighted tardiness when scheduled last. Pseudo code for the algorithm is given in Algorithm 5. Similar to the previous greedy algorithm, this algorithm requires an array of experiments where each experiment has instance size, processing time, due date, priority, and experiment number. The algorithm determines the processing order of the experiments by comparing their potential weighted tardiness costs. The algorithm finds the experiment with the lowest weighted tardiness cost among all remaining experiments which have not yet been processed and assigns it to be processed last among unprocessed experiments.

Algorithm 5 LCL

```
Require: experiments array
1: while $experiments$ is not empty do
2:     for every experiment in $experiments$ do
3:         Assume the experiment is the last processed experiment and calculate its weighted tardiness.
4:         Record the experiment with the smallest weighted tardiness value.
5:     end for
6:     Set the found experiment with least weighted tardiness cost as the last experiment to be assigned.
7:     Remove the experiment with least tardiness cost from $experiments$.
8: end while
9: When all experiments finish, calculate the total tardiness as total minimum tardiness
```

The least-cost-last greedy algorithm follows the procedure of selecting the experiment with the least weighted tardiness value among experiments which hasn't yet started. Then the selected experiment will be processed last and the algorithm repeats the procedure. Iterating the algorithm for all N experiments requires $O(N)$ time. Similarly, the second step runs in $O(N)$ time to find the experiment with least weighted tardiness cost among remaining unprocessed experiments.

Finally the algorithm requires $O(K)$ time since N experiments require K participants to start. It fills in array $orderArr$ of size K to store participant assignment permutations that have been found, and to calculate the total weighted tardiness value. Therefore the overall procedure runs in $O(N^2 + K)$ time with a space requirement of $O(K)$.

Greedy Algorithm Based on Apparent Tardiness Cost: The apparent tardiness cost (ATC) job scheduling algorithm was proposed by Vepselainen and Morton [43] and it is known as an effective composite dispatching rule for minimizing weighted tardiness. It is a multi-machine dynamic scheduling method and it is an aggregation of weighted shortest processing time (WSPT) and minimum slack (MS) algorithms. WSPT sequences jobs in non-decreasing order of $\frac{PR_j}{PT_j}$ and it is optimal for single machine job scheduling. MS schedules the job with with the smallest difference between the due date and the sum of job start time and processing time.

Given the current time t and and experiment j, priority index $I_j(t)$ is a combination of WSPT and MS dispatching rules and it is used to determine the experiment j to be scheduled at a given time t as follows:

$$I_j(t) = \frac{PR_j}{PT_j} \times exp\left(\frac{-max(DD_j - PT_j - t, 0)}{\alpha \times avg(PT)}\right) \tag{3.1}$$

In the above formula $I_j(t)$ represents the priority index of an experiment j that is available at time t, α is a lookahead parameter and $avg(PT)$ is the average of the available experiments' processing times. This last value is used in the formula with α to scale the slack. Morton et al. [44] describe results showing that $\alpha = 2$ gives good results for static flow shop problems in which the problem is determining best sequence of jobs and $\alpha = 3$ gives good results for dynamic flow shop problems in which jobs arrive continuosly over time.

The exponential function in the formula is used to select alternate dispatching rules for the experiments. When α is very large, the method reduces to the WSPT heuristic, and when α is very small, the method reduces to the MS heuristic. When a participant arrives at time t, the ATC algorithm calculates the priority index of every available experiment, and assigns the participant to the experiment with the highest priority index value. Time-dependent index values for the available experiments are recalculated whenever a new participant arrives for assignment. pseudo code for the algorithm is given in Algorithm 6 below. The algorithm requires an array of experiments to be scheduled, α lookahead parameter and the total number of participants needed for all experiments (K).

Iterating the above algorithm for each of the K required participants requires $O(K)$ time.

Algorithm 6 ATC

```
Require: experiments array, $\alpha$, $K$
 1: for $t = 0 \to K - 1$ do
 2:     for every experiment in $experiments$ do
 3:         Calculate $WSPT$ as experiment.PR/experiment.PT.
 4:         Calculate $MS$ as $\max$(experiment.DD−experiment.PT−t,0).
 5:         Calculate average processing time of all $experiments$ as $avgPT$.
 6:         Set $I(t) = WSPT \times exp(-MS/(\alpha \times avgPT))$
 7:         Store the experiment with highest $I(t)$ value.
 8:     end for
 9:     Assign participants to the experiment with highest $I(t)$ value.
10:     Remove the experiment with highest $I(t)$ value from $experiments$.
11: end for
12: When all experiments finish, calculate the total tardiness as total minimum tardiness
```

The inner loop runs in $O(N^2)$ time to find the experiment with highest time indexed index value by calculating the average processing time of the experiments. Finally the algorithm requires $O(K)$ time to fill in the $K-$sized $orderArr$ which stores the computed participant assignment permutation, and to calculate the total weighted tardiness value. Therefore the overall procedure runs in $O(K \times N^2)$ time with $O(K)$ space requirement.

Apparent Tardiness Cost for Participant Assignment: The apparent tardiness cost algorithm is a dispatching rule designed for job scheduling and tardiness of a job is calculated when it is assigned to a machine and processed. In this study, our aim is to assign participants to experiments with different experiment sizes in a multi-machine environment. Therefore, we also need to consider experiment instance sizes while making use of the apparent tardiness cost in participant assignment problem, similar to the LCL heuristic which considers instance sizes implicity while comparing the weighted tardiness values. As stated above, the ATC algorithm is a combination of the WSPT and MS heuristics. In this study we propose a modified ATC dispatching rule for minimizing total weighted tardiness in participant assignment problem. For considering instance sizes implicitly similar to what LCL algorithm does and for benefits of the EDD heuristic, we combine the EDD and LCL heuristics with ATC heuristic as follows:

$$I_j(t) = \frac{PR_j}{PT_j} \times exp\left(\frac{-max(DD_j - PT_j - t, 0)}{\alpha \times avg(PT)}\right) \times \frac{WT_j}{DD_j} \tag{3.2}$$

The formula for the apparent tardiness cost for participant assignment (ATCPA) is similar to the formula of ATC. In addition to the WSPT and MS heuristics considered in ATC method, in ATCPA we also combined EDD and LCL heuristics. The additional operation considered in ATCPA is to include both the weighted tardiness of the experiment. As stated above, in LCL the experiment with least weighted tardiness cost is scheduled to be assigned at last, which is equivalent to scheduling the experiment with highest weighted tardiness cost to be scheduled first. In ATCPA we select the experiment with the largest time-indexed value for participant assignment. Hence, multipying the ATC result with weighted tardiness of the considered experiment will increase its chance for being selected as next experiment if its weighted tardiness value is high. Similarly, in EDD we give priority to the experiments with earliest due date. Thus, dividing the ATC result by the due date of an experiment with a lower due date value will increase the selection chance of that experiment. Pseudo code of the algorithm is given in Algorithm 7 below. Similar to ATC, the algorithm requires an array of experiments to be scheduled, α lookahead parameter and the total number of participants needed for all experiments (K).

Algorithm 7 ATCPA

```
Require: experiments array, $\alpha$, $K$
 1: for $t = 0 \to K - 1$ do
 2:     for every experiment in $experiments$ do
 3:         Calculate $WSPT$ as experiment.PR/experiment.PT.
 4:         Calculate $MS$ as $\max$(experiment.DD−experiment.PT−t,0).
 5:         Calculate average processing time of all $experiments$ as $avgPT$.
 6:         Calculate $WT$ as $\max$(experiment.PR+$t$-experiment.DD, 0) $\times$ experiment.PR
 7:         Set $I(t) = WSPT \times exp(-MS/(\alpha \times avgPT)) \times WT/experiment.DD$
 8:         Store the experiment with highest $I(t)$ value.
 9:     end for
10:     Assign participants to the experiment with highest $I(t)$ value.
11:     Remove the experiment with highest $I(t)$ value from $experiments$.
12: end for
13: When all experiments finish, calculate the total tardiness as total minimum tardiness
```

The algorithm for ATCPA has the same structure as ATC and thus it runs in $O(K \times N^2)$ time with $O(K)$ space requirement.

3.3.4 Integer-Linear Programming Solution

In this section, we describe an integer-linear programming (ILP) solution for the problem of minimizing multi-machine total weighted tardiness to solve the participant assignment problem. Integer-linear programming is widely used to solve scheduling problems, and used by [45] and [46] to minimize the weighted tardiness of jobs on a single machine with fixed processing times and instance sizes. In contrast, the jobs (i.e. experiments) in our work run on multiple machines with varying instance sizes and processing times. We developed an integer-linear programming formulation that minimizes weighted tardiness on multiple machines with varying instance sizes and processing times. The problem is formulated as a time-indexed ILP similar to that described in [45] and [46].

Since experiments have varying sizes, our formulation of the problem uses two different types of participant assignments. The first type is an assignment of participants which does not make an experiment start. This type of assignment does not directly affect the total weighted tardiness. The second type of assignment takes place when the last participant is assigned and that experiment starts. This type of assignment directly affects the total weighted tardiness. We assume that only one participant assignment takes place at a time, although at any time more than one experiment can be running. The objective function is the total weighted tardiness of each experiment.

We can now formulate the multi-machine total weighted tardiness problem as a time-indexed ILP model. In the ILP formulation, two binary variables $x_{j,t}$ and $z_{j,t}$ are used. The variables are defined for all $j \in N$ and for all $t \in K$ as:

$$x_{j,t} = \begin{cases} 1 & \text{if the participant arriving at time t is assigned to experiment } j \\ & \text{and this assignment does not make experiment } j \text{ start} \\ 0 & \text{Otherwise} \end{cases}$$

$$z_{j,t} = \begin{cases} 1 & \text{if the participant arriving at time t is assigned to experiment } j \\ & \text{and this assignment makes experiment } j \text{ start} \\ 0 & \text{Otherwise} \end{cases}$$

In the participant assignment problem, there are N experiments where the total number of participants needed to start all these experiments is K. We let parameter $C_{j,t}$ equal the completion time of experiment j if it begins executaion at time t for all $j \in N$ and for all $t \in K$ i.e.,

$$C_{j,t} = t + PT_j \tag{3.3}$$

The tardiness of an experiment indicates whether the experiment is completed before or after the specified due date of that experiment. The tardiness of experiment j if it starts at time t is then formulated as:

$$Tardiness_{j,t} = \begin{cases} 0 & if \quad PT_j \leq DD_j \\ C_{j,t} - DD_j & \text{Otherwise} \end{cases}$$

As described in section 3.1, the weighted tardiness of an experiment indicates the importance of meeting the due date for some experiment from the researcher's point of view. Weighted tardiness is computed by multiplying the assigned priority of an experiment by its tardiness value. The weighted tardiness (WT) of experiment j that starts at time t is:

$$WT_{j,t} = PR_j \times Tardiness_{j,t} \tag{3.4}$$

The total weighted tardiness (TWT) is the sum of the weighted tardiness over all experiments, i.e.,

$$TWT = \sum_{j=1}^{N} PR_j \times Tardiness_{j,t} \quad \forall\, t \,\in\, K \tag{3.5}$$

Then the complete ILP model for minimizing the total weighted tardiness is:

$$min\ TWT = \sum_{j=1}^{N} \sum_{t=0}^{K-1} PR_j \times Tardiness_{j,t} \times z_{j,t} \tag{3.6}$$

subject to

$$\sum_{t=0}^{K-2} x_{j,t} = IS_j - 1 \quad \forall\, j \,\in\, N \tag{3.7}$$

$$\sum_{j=1}^{N} x_{j,t} \leq 1 \quad \forall\, t \,\in\, K-2 \tag{3.8}$$

$$\sum_{t=0}^{K-1} z_{j,t} = 1 \quad \forall\, j \,\in\, N \tag{3.9}$$

$$\sum_{j=1}^{N} z_{j,t} \leq 1 \quad \forall\, t \,\in\, K-1 \tag{3.10}$$

$$\sum_{j=1}^{N} x_{j,t} + \sum_{k=1}^{N} z_{k,t} \leq 1 \quad \forall\, t \,\in\, K-1 \tag{3.11}$$

$$x_{j,t} \times t \leq \sum_{m=0}^{K-1} z_{j,m} \times m \quad \forall\, j \,\in\, N,\ t \,\in\, K-1 \tag{3.12}$$

$$x_{j,t} \in \{0,1\} \quad \forall\, j \,\in\, N,\ t \,\in\, K \tag{3.13}$$

$$z_{j,t} \in \{0,1\} \quad \forall j \in N, \; t \in K \tag{3.14}$$

Equation 3.7 ensures that the number of participant assignments of the first type made to experiment j does not exceed the instance size minus 1. The summation for this constraint is bounded by $K-2$ since that is the last possible time for a participant assignment which does not result in any experiment starting, and in that case the experiment can start when $t = K-1$ with the assignment of the last required participant. Equation 3.8 ensures that at most one participant assignment takes place at any time. Note that at any time, the value of $\sum\limits_{j=1}^{N} x_{j,t}$ can equal 0 if the assigned participant at time t makes one of the experiments start. Equation 3.9 states that each experiment can start only once. Equation 3.12 ensures that at most one experiment can start at any time. Equation 3.10 ensures that at most one of $x_{j,t}$ and $z_{j,t}$ can equal 1 at any time. Equation 3.11 ensures that each $x_{j,t}$ is set to 1 before setting $z_{j,t}$ to 1 for every experiment j, since the assignment times of the participants which do not result in an experiment starting must be smaller than the last participant's assignment time for each experiment.

3.4 Methodology

The performance of each algorithm is evaluated over a synthetic dataset which contains experiments with uniform arrivals and release times as well as varying instance sizes, processing times, priorities and due dates. Each benchmark instance contains a set of N experiments, and we assume that K participants are needed to make all experiments start. The size of each experiment is assigned randomly from a uniform distribution with a range of either 1 to 3, 4 to 6, or 7 to 9. A segment of the benchmark includes 50 instances that have the same value of N. The benchmark contains 30 segments whose parameters are described in Table 3.2. For each segment of the benchmark, the table gives the values of N, the experiment size (ES) interval for each of those N experiments, and the average and standard deviation of K.

Table 3.2: Statistics about the used synthetic dataset.

	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
	μ_K	σ_K	μ_K	σ_K	μ_K	σ_K
$N = 5$	10.3	2.0	25.3	1.7	40.3	1.7
$N = 10$	19.9	2.7	50.4	3.0	79.4	2.8
$N = 15$	29.8	2.8	74.7	2.9	120.5	3.1
$N = 50$	99.6	4.9	249.6	5.4	400.0	6.0
$N = 100$	200.7	8.5	498.4	8.1	801.5	8.0
$N = 200$	300.0	13.1	998.9	9.6	1598.4	11.4
$N = 500$	999.8	20.1	2500.0	17.6	4002.8	17.4
$N = 1000$	1999.9	25.7	5003.0	25.3	8002.2	22.9
$N = 2000$	4004.0	32.8	10001.4	39.5	15998.5	41.3
$N = 5000$	9995.7	63.1	24995.5	57.3	40009.3	69.2

The running time of our algorithms is mostly a function of the number of experiments and the total number of participants required to make them run. We therefore evaluate the algorithms over bencemark instances with varying N and K values. We apply each algorithm to all 50 instances for each value of N and K.

Since the exhaustive algorithm considers all possible assignments in the process of finding the total minimum weighted tardiness, it is guaranteed to give an optimal solution to the problem but is expected to have the longest execution time. Similarly, the dynamic programming algorithm finds optimal solutions with better execution times. Greedy algorithms only consider heuristic rules for finding the participant assignment order, so this approach is very fast but does not necessarily find optimal solutions. The ILP approach includes all necessary constraints for a valid assignment and finds optimal solution to the problem. The exhaustive, DP and ILP techniques are guaranteed to find optimal solution, but may take an unreasonably large time to complete.

When comparing algorithms, we report average percentage difference between the total weighted tardiness of the assignments found by two algorithms, and the standard deviation of these values.

The benchmark instances are divided into small and large instances. An instance is small if an optimal solution can be found by one of the exact algorithms considered in this thesis. The remaining large instances can only be solved by the the heuristic algorithms considered in this thesis.

3.5 Experimental Results

In this section we describe experimental results for the total minimum weighted tardiness algorithms which we described in Section 3.3. We report and compare the performance of the exhaustive enumeration, dynamic programming, greedy and integer-linear programming solutions. We then give the size of the largest solved instance for each algorithm, and investigate how close the solutions they provide are to optimal.

The algorithms are evaluated based on the percentage of the instances they are able to process within a time limit, how close the solutions are to the optimal solution, and the amount of time needed to process the largest instance in the benchmark, for each value of N and K.

The first set of experiments investigate the percentage of the benchmark that the algorithms can process within a fixed amount of time. An instance is processed if the algorithm completes within the time limit, regardless of whether or not an optimal solution is found. The processed percentage is calculated by dividing the number of instances that each algorithm processes by the total number of instances in that segment. The execution time of the algorithms is limited to either 1 minute or 10 minutes. Results are reported in Table 3.3 for the 1-minute time restriction and in Table 3.4 for the 10-minute time restriction.

Table 3.3: The number and percentage of the instances that are processed within one minute.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
EDD, LCL	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
ATC, ATCPA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	46	92%	45	90%	44	88%
	2000	5	10%	5	10%	5	10%
	5000	0	0%	0	0%	0	0%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	7	14%	0	0%
	15	36	72%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

Table 3.4: The number and percentage of the instances that are processed within ten minutes.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
Exhaustive	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	0	0%	0	0%
	10	2	4%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
DP	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
EDD, LCL	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
ATC, ATCPA	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	3	6%	3	6%	3	6%
	5	50	100%	50	100%	50	100%
	10	50	100%	15	30%	0	0%
	15	50	100%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
ILP	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

As can be seen in Table 3.3 and Table 3.4, when the experiment size (ES) or number of experiment (N) increase, the total number of participants needed (K) for each benchmark instance increases and the number of solved instances within each segment decreases. As expected, the worst performance is observed for the exhaustive solution. The second worst results are observed with the dynamic programming algorithm. Since the dynamic programming algorithm's complexity is exponential with respect to K and quadratic with respect to N, these results are expected. Based on the results from both tables, the greedy algorithms are the fastest algorithms. This is expected since the complexity of the greedy algorithms are quadratic and mostly dependent on used sorted algorithm for sorting the due dates, priorities and experiment sizes of N experiments. The integer-linear programming solver completes more quickly than the exhaustive and dynamic programming methods. For larger values of K, the execution time of the integer-linear programming solver increases quickly.

The second set of experiments is conducted to determine how close the solutions provided by these algorithms are to optimal. The solutions provided by the dynamic programming algorithm are used as a baseline since it provides optimal solutions and runs more quickly than the exhaustive algorithm. We applied the dynamic programming algorithm for 30 minutes to every instance in a subset of a benchmark segment that was not solved by any of our exact algorithms in less than 10 minutes. The instances for this test consist of 35 instances from the benchmark segment with $N = 10$ and $4 \leq ES \leq 6$. We then applied the other algorithms to this benchmark segment for 1 minute and 10 minutes for every instance. We report the difference between the cost of the solutions found by a target algorithm and the optimal cost, divided by the optimal cost. The results from the comparison of the exhaustive, integer-linear programming and greedy algorithms are provided in Figure 3.2

As can be seen from Figure 3.2, when the execution time increases from 1 minute to 10 minutes, exhaustive and ILP algorithms find more accurate results. Accuracy results are observed to be better for ILP than the exhaustive algorithm which means ILP finds optimal results faster than the exhaustive algorithm and this result is coherent with our previous experiment based on algorithms' execution performance. No change is observed for the case of greedy algorithms, since they are very fast heuristics and find results within 1 minute. Exhaustive and ILP algorithms find solution that are on average 75.6% and 3.2% respectively from optimal for 1-minute tests, and 72.9% and 2.3% respectively from optimal for 10-minute tests. EDD, LCL, ATC and ATCPA algorithms are 43.4%, 13.3%, 25.8% and 12.1% respectively from optimal for both 1-minute and 10 minutes tests.

Intuitively these results were expected because although exhaustive is an exact algorithm like ILP, ILP executes faster by exploiting the branch and bound algorithm. This causes ILP to perform much better than exhaustive. ATCPA is performing better than LCL and EDD because similar to ATC, it makes use of a look-ahead parameter for lead time estimation which reflects possible future load in the experiments. In addition to ATC it makes use of experiment instance size and weighted tardiness of experiment. Exhaustive algorithm and ILP performs slower than the other algorithms because exhaustive algorithm needs to consider all participant assignment scenarios and ILP algorithm needs to consider many search state-space trees.

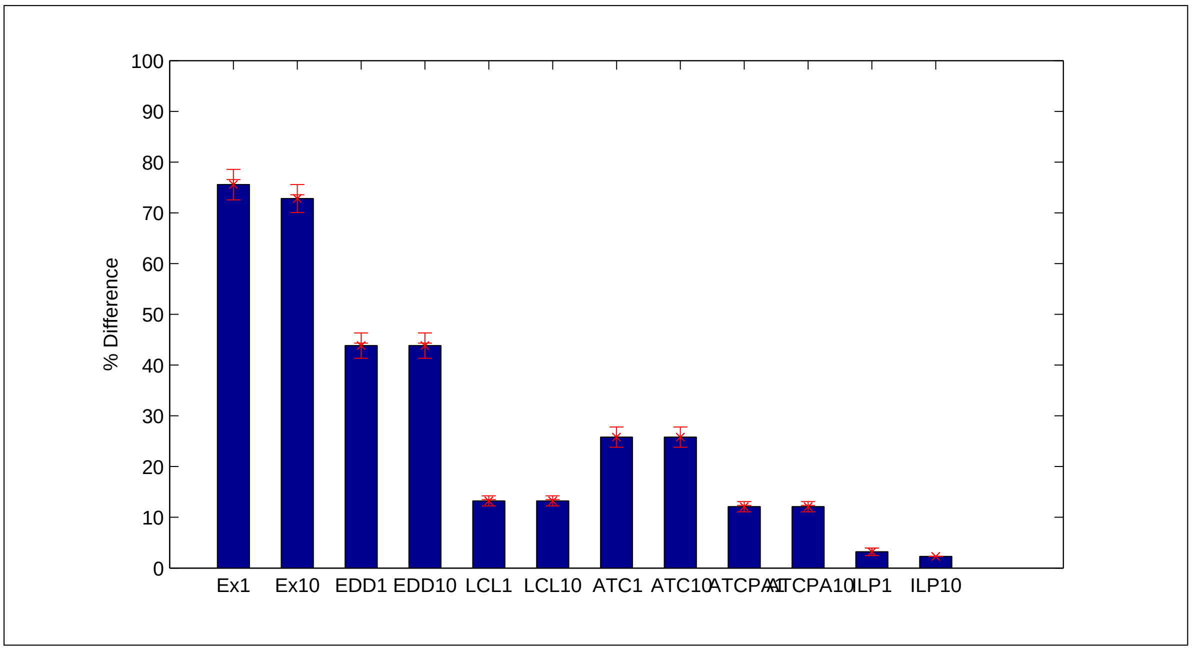


Figure 3.2: Percentage difference between the studied algorithms and dynamic programming after after running its instances with 1-minute and 10-minute time restrictions. Ex1 and Ex10 stands for 1-minute and 10-minutes execution results of the exhaustive algorithm. EDD1, EDD10, LCL1, LCL10, ATC1, ATC10, ATCPA1, ATCPA10 and ILP1, ILP10 stand for 1-minute and 10-minute execution results of the greedy and ILP algorithms respectively.

These results are based only on instances where we have been able to find exact solutions by the DP algorithm within 10 minutes. For harder instances, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. Table 3.5 shows the average difference between the LP lower bound and optimal ILP solutions for small instances where optimal results were obtained using ILP within 10 minutes. The average difference between the LP lower bound and the optimal results is 26.5%. In the same way, the average difference between the LP lower bound and the solutions computed by the EDD, LCL, ATC and ATCPA algorithms is found to be 72.0%, 60.9%, 66.2% and 60.3% respectively.

Table 3.5: Comparison of LP and ILP lower bounds for small instances as well as LP and greedy algorithms on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. ILP	12.3%	36.0%	49.7%
LP vs. EDD	59.2%	73.9%	82.7%
LP vs. LCL	38.1%	66.5%	78.1%
LP vs. ATC	46.1%	71.6%	80.7%
LP vs. ATCPA	35.5%	66.6%	78.6%

According to the 1-minute and 10-minute optimality tests and the LP comparison tests, it is observed that on the average the ATCPA heuristic performs better than the other greedy heuristics. By taking the ATCPA algorithm as baseline algorithm, we performed another test to determine if the difference between the compared greedy heuristics and the baseline greedy heuristic is statistically significant. In Table 3.6 we present the number of instances for which the baseline greedy heuristic performs better ($<$), equal ($=$) or worse ($>$) than the compared greedy heuristic. In this table we also report if the difference between the compared greedy heuristics is statistically significant. Since the compared greedy heuristics are tested on the same problems, we can conduct a paired-samples test. However, experimental conditions do not meet the hypothesis of the paired-samples t-test (i.e. in t-test dependent variable, which is weighted tardiness for our case, must be on an interval scale). For this reason we conducted nonparametric Wilcoxon test.

Table 3.6: Comparison results of other greedy heuristics with ATCPA on different benchmark segments.

	EDD				LCL				ATC			
N	$<$	$=$	$>$	$p-value$	$<$	$=$	$>$	$p-value$	$<$	$=$	$>$	$p-value$
5	89	29	36	0.0006	28	58	64	0.0083	104	10	36	0.0001
10	145	1	4	0.0001	63	45	42	0.0019	133	0	17	0.0001
15	150	0	0	0.0001	108	4	38	0.0001	135	1	14	0.0001
50	150	0	0	0.0001	150	0	0	0.0001	139	0	11	0.0001
100	150	0	0	0.0001	149	0	1	0.0001	142	1	7	0.0001
200	150	0	0	0.0001	140	0	10	0.0001	141	0	0	0.0001
500	150	0	0	0.0001	93	0	57	0.0001	146	0	4	0.0001
1000	150	0	0	0.0001	51	1	98	0.0001	107	0	43	0.0001
2000	150	0	0	0.0001	27	0	123	0.0001	106	0	44	0.001
5000	150	0	0	0.0001	11	0	139	0.0001	150	0	0	0.001

From the results obtained from Table 3.6, we conclude that ATCPA greedy heuristic outperforms the other greedy heuristics and provides better results for many benchmark instances. Wilcoxon test results also indicates that the difference between ATCPA heuristic and the other compared greedy heuristics are statistically significant.

According to the execution performance statistics of the algorithms, it is observed that greedy algorithms outperformed the other studied methods both in 1-minute and 10-minute tests. The integer-linear programming algorithm works faster than the exhaustive and dynamic programming methods at finding the exact solutions. These results imply that the greedy algorithms should be chosen to solve assignment problems if one attaches more importance to faster execution of the assignment process. If one attaches more importance to the exact solution of the given problem then integer-linear programming method should be chosen since it runs slower with respect to greedy but finds exact solutions. If one pays equal importance to execution speed of the algorithm and the exactness of the found solutions, then the greedy algorithm would be the best choice since it is much faster than the other compared algorithms and the ATCPA greedy heuristic performs reasonably well at finding the exact solutions.

Chapter 4

Participant Assignment with Varying Arrivals and Uniform Release Times

Participant assignment algorithms considered in the first scenario assume that participants arrive uniformly and that all experiments are released before starting the participant assignment process, as depicted in Figure 4.1. However, in practice, participants' arrival times will be irregular, and not all experiments will be released for participant assignment at the same time.

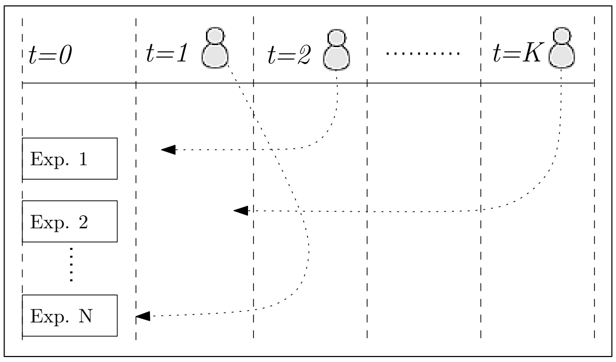


Figure 4.1: Assignment of participants when experiment release times and arrivals are uniform.

In the following sections we consider these alternative participant assignment scenarios, past work on these topics, and discuss differences between the extended models and the models we evaluated in Section 3.5.

In this section we consider one of these alternative participant assignment scenarios and present our participant assignment algorithms. These algorithms handle the realistic situation where participants' arrival times vary and experiments are all initially available. We evaluate and report the performance of these algorithms when applied to a synthetic dataset containing a wide range of instances.

4.1 Background

The problem of minimizing weighted tardiness with varying participant arrivals and uniform experiment release times is visualized in Figure 4.2. This problem is equivalent to the job scheduling problem where jobs with unequal release times are assigned to parallel machines which are all initially available. Dessouky [47] considered a similar job scheduling problem where n identical jobs with unequal release times are scheduled on m parallel machines to minimize the maximum lateness. The author describes a solution the problem using a branch and bound algorithm and six offline greedy heuristics that find the approximate solutions such as selecting the job to be scheduled next based on jobs' ready times. In this heuristic, the job to be scheduled next is the job that has been waiting the longest among the unscheduled jobs. The results indicate that the proposed method is able to find optimal solutions for small instances, and near-optimal solutions for large instances.

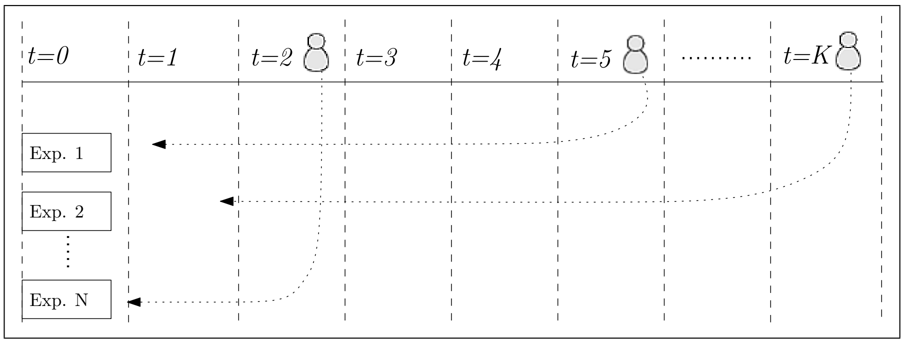


Figure 4.2: Assignment of participants with uniform experiment release times and varying arrivals.

Monch et al. [48] describe two different offline job scheduling approaches to minimize the total weighted tardiness of jobs with unequal ready times on parallel machines. Their heuristics use genetic algorithms to assign jobs to machines. Their first approach starts by assigning jobs to batches, and then assigns batches of jobs to machines using a genetic algorithm to sequence them. Their second approach starts by assigning jobs to machines using genetic algorithm and then forms batches of jobs to sequence them. In both approaches batches are created by making use of modified version of apparent tardiness cost heuristic. Their experimental results on their synthetic dataset show that their first approach outperforms their second approach with respect to solution quality and computation time. They also result in that for creating batches, larger window sizes and smaller thresholds improve the average total weighted tardiness value.

Jafari et al. [49] describe an ILP-based method to schedule parallel batch-processing machines to minimize the total tardiness of jobs. Their model considers several issues such as varying release times of experiments, varying job sizes and varying processing times. Their offline algorithm is a hybrid approximation algorithm using simulated annealing to find near-optimal solutions for large instances. The performance of their algorithm is verified for small size problems by comparing the results with the solution obtained by CPLEX, simulated annealing and random key genetic algorithm. The results showed that were able to find optimal results for up to 180 instances.

Reeves [50] modifies a previously-known heuristic used for job scheduling with unequal ready times to improve its performance. The original heuristic selects a job with the smallest weight factor, where the weight factor is a function of a job's release time and processing time. Weight factor definition contains a multiplicative constant $0 \leq \alpha \leq 1$ which is multiplied with effective release time of job j and $1 - \alpha$ is multiplied with processing time of the job. Effective release time of job j is the maximum of previous job's release time summed with its processing time and release time of the current job. By using multiplicative constant, different importance is given to effective release times and processing times and this effects the assignment order of the weighted tardines calculation. In his study, Reeves uses tabu search to find a better performing α value. The experimental results shows that the modifications of α values by making use of heuristics gives better quality solutions in certain situations where the original algorithm fails.

Mahnam et al. [51] studies job scheduling to minimize the sum of maximum job earliness and tardiness with varying job ready times. They describe a branch and bound algorithm with dominance rules that reduce the size of the search tree. They also present two dispatching rules, and two metaheuristics based on a genetic algorithm and particle swarm optimization to solve the problem for large job sizes. The modified earliest due date and modified minimum slack time heuristics derived from EDD and MST rules for the case of unequal release times have been used to obtain the upper bound in the proposed branch and bound algorithm. Experimental results show that the branch and bound algorithm is capable of solving the problems up to 20 jobs. The metaheuristics are able to solve instances up to 1000 jobs. The results show that the proposed genetic algorithm improves the solutions compared to particle swarm optimization in terms of both quality and and time efficiency. Although the heuristic is not optimal, it was also observed that in a large fraction of instances, the best solution obtained from the dispatching rules was as good as the optimal solution obtained from the branch-and-bound algorithm.

Wu et al. [52] solves the job scheduling problem with varying release times for a wafer fabrication facility. Their solution is based on branch and bound and simulated annealing. For their application with unequal ready times, it is better to wait for future job arrivals in order to increase the fullness of each batch. They also observe that the repeated processing of similar tasks improves workers' skills. The objective is to find a schedule to minimize total completion time. Branch and bound is used to solve the problems to find the optimal solution and simulated annealing is used to find the near optimal solutions. Experimental results show that branch and bound solves instances up to 24 jobs and the average error percentage of the simulated-annealing algorithm is less than 0.1482%. It is also observed that in most of the cases the simulated annealing solutions took less than a second to execute.

Akturk and Ozdemir [53] propose a new dominance rule to minimize total weighted tardiness with unequal release times. Their dominance rule considers time-dependent orderings between each pair of jobs and provides local optimality. They then describe a new algorithm based on their dominance rule, which is a modified version of the apparent tardiness cost algorithm [43].

They compare their proposed greedy heuristic with a number of greedy heuristics on problems with 50, 100, and 150 jobs over a set of randomly generated instances. Since greedy heuristics may result in solutions that are far from the optimum, they also adapt and compare their dominance rule to a local search algorithms. Experimental results indicate that the proposed algorithm dominates the competing algorithms in terms of optimality and execution performance. They also show that the improvements are statistically significant.

Job scheduling where jobs have unequal release times is known to be NP-hard [54, 55]. This problem is equivalent to the participant assignment problem with varying arrival times. Lawler [25] and Lenstra [26] show the single machine assignment problem is minimally NP hard. Lawler [25] showed that the problem is NP-complete if there are arbitrary precedence constraints and he also showed that if precedence constraints are series parallel, the problem can be solved in $O(nlogn)$ time. Cheng et al. [54] show that the scheduling problem with varying job release times on one machine is NP-hard. Monch et al. [55] confirms the problem is NP-hard for scheduling problem with varying job release times on parallel machines. Kan [56] proves that the scheduling problem with varying ready times is NP-hard and heuristic methods are essential in order to solve the problems of reasonable size in a reasonable amount of computer time.

4.2 Algorithms for Optimized Participant Assignment

In this section, we present our algorithms for the scenario that includes varying participant arrival times and uniform experiment release times. We solve this problem using exhaustive enumeration, dynamic programming, greedy algorithms and integer-linear programming. In the following sections we introduce details of these algorithms and then we give our experimental results.

4.2.1 Exhaustive enumeration

The exhaustive algorithm with varying arrival times and uniform release times is similar to the exhaustive algorithm designed for uniform arrivals of participants and uniform experiment release times. The difference between these two exhaustive algorithms is that in the second scenario, the calculation of total weighted tardiness must take into consideration the arrival time of the last arriving participant which allows the experiment to run.

Similar to the first scenario, the recursive exhaustive algorithm takes as input the number of completed experiments, an array of experiments where each experiment contains parameters such as instance size, processing time, due date and priority. Unlike the first scenario, there is an arrival array that contains the arrival time of each participant.

The pseudo code shown in Algorithm 8 is similar to the algorithm shown in 8, with the following exception. In lines 10-11 for each completed permutation of participant-experiment assignment, the weighted tardiness of the complete assignments is calculated by making use of participants' varying arrival times and checked to see if it is the minimum weighted tardiness that has been found so far.

The base case occurs when all the experiments are completed. The pseudo code is similar to algorithm shown in 8, with the following exception. The arrival array that contains the varying arrival time of each participant helps us to calculate the weighted tardiness of the experiments with varying participant arrival times.

Similar to the first scenario, if K is the number of assigned participants, the overall function runs within a polynomial factor of O(K!) in the worst case, assuming every experiment requires at least one participant to start.

Algorithm 8 Exhaustive

```
Require: count of started experiments, experiments array, arrivals array
 1: if count of started experiments < number of $experiments$ then
 2:     for every experiment $i$ do
 3:         Assign the next participant to the any empty slot of experiment $i$
 4:         if Assignment process results in the experiment to start then
 5:             Increment the count of started experiments
 6:         end if
 7:         $Exhaustive($count of started experiments, $experiments, arrivals)$
 8:     end for
 9: else
10:     Calculate the tardiness value of the found assignment order
11:     Store the minimum found tardiness value
12: end if
```

4.2.2 Dynamic Programming Algorithm

The recursive algorithm for the scenario of varying participant arrivals and uniform experiment release times is similar to the recursive algorithm used in the first scenario.

Structure of the Optimal Solution:

The definitions of the sorted sequence of experiment numbers S, optimal solution G, subsequence $S - E$ where E is a given sequence of experiment numbers are also the same. Participants are assigned one by one starting from the first incoming participant and the assignment of a participant to an experiment may result in a tardiness value of zero or more depending on the instance size of the experiment. Similar to the first scenario as described in Section 3.3.2, we demonstrate the structure of the optimal solution by contradiction. The overall solution is the weighted tardiness value result in first participant's assignment plus the optimized total weighted tardiness value of the subproblem where the varying arrivals of the incoming participants are also taken into consideration. Otherwise the value of the overall solution will be equal to the optimized total weighted tardiness value of the subproblem. If we assume there is a better way of assigning participants to available experiments starting from second participant, then we could select that assignment permutation and this would result in a better solution than the optimal solution G to the participant assignment problem which is a contradiction.

Recursive Definition of the Optimal Solution:

Since the dynamic programming algorithm that is used for the second scenario is similar to the dynamic programming algorithm of the first scenario, the recursive solution to the problem is also similar. The recursive algorithm solves the problem top down, and the dynamic problem solution solves the minimizing weighted tardiness problem bottom up by making use of the recurrence relation.

The calculation difference of weighted tardiness values between the scenarios does not significantly affect the recusive formulation of the dynamic programming and this results in the similar recursive formulation of the problem.

Given the sorted sequence of experiment numbers S, experiment number i, participant's arrival time array A and arrival order number k; $G(i, S, A(k))$ is the total minimum weighted tardiness assuming that the participant arriving at time $A(k)$ is assigned to experiment i and the rest of participants are assigned to experiment numbers from S. Then we can recursively define the optimal solution as:

$$G(i,S,A(k)) = \begin{cases} WT_i(A(k)) & \text{if} \quad S \text{ is empty} \\ min_j\{G(j,S-(j),A(k+1))\} & \text{if} \quad i \text{ in } S \\ WT_i(A(k)) + min_j\{G(j,S-(j),A(k+1))\} & \text{if} \quad i \text{ not in } S \text{ and } S \text{ is not empty} \end{cases} \tag{4.1}$$

As described in Algorithm 3.3.2, this statement indicates that there are three cases to be considered to determine the total minimum weighted tardiness. Unlike the algorithm in Section 3.3.2, instead of taking sequential time periods (t values) we are keeping an array of participant arrivals and we consider each participant arrival in the array one by one.

Computing the Value of the Optimal Solution Bottom Up:

Similar to the Algorithm 2, the optimal solution is computed bottom up by making use of experiment numbers and S sequence. The total tardiness value is calculated by making use of the arrival values of the participants. Pseudo code for the dynamic programming solution is given in Algorithm 9. Given the experiment numbers and arrival array, our goal is to find

$$min\{G(i,S,A(0))\} \ \forall \text{ experiment number } i$$

In the Algorithm 9, lines 4, 12 and 14 differ from the Algorithm 2 in the sense that these lines compute the weighted tardiness value by making use of the provided A arrivals array.

Constructing Optimal Solution from Computed Information:

Similar to the optimal solution construction approach of Algorithm 3.3.2, we find the optimal total weighted tardiness value by making use of the tuples. Having identified $(1, i)$ at time $t = A(0)$, we can backtrack our optimal solution by finding the experiment j for each tuple that minimizes the expression.

Algorithm 9 Dynamic

```
Require: experiments array, S sequence, A
 1: Initialize $K$ as the number of participants needed to make all experiments start
 2: Initialize $G$ as empty to hold weighted tardiness values
 3: for every experiment $i$ do
 4:     Set $G(i, \emptyset, A(m))$ = weighted tardiness of experiment $i$ assuming it finishes at time $A(m)$ for $m = K - 1$
 5: end for
 6: for every subsequence of $S$ do
 7:     for every experiment $j$ from $S$ do
 8:         Set $m = K - 1-$ size of subsequence
 9:         if $j$ is in subsequence then
10:             $G(j, subsequence, A(m)) = min\{G(k, subsequence - (k), A(m + 1))\}$ $\forall$ $k$ in subsequence
11:         else
12:             $G(j, subsequence, A(m))$ =weighted tardiness of $j + min\{G(k, subsequence - (k), A(m + 1))\}$ $\forall$ $k$ in subsequence
13:         end if
14:     end for
15: end for
```

Running Time and Space Requirements:

The "Dynamic" algorithm that is used for computing the optimal solution has the same running time complexity as the dynamic programming algorithm for scenario 1 and it is a $O(K \times 2^K)$ time operation where K is the summation of experiments' instance sizes. The procedure that is used to construct the optimal solution takes $O(K^3)$ time and thus the overall procedure has a running time of $O(K^2 \times 2^K)$. Similar to the scenario 1, the total space requirement is also $O(K \times 2^K)$.

4.2.3 Greedy Algorithms

In the scenario with varying participant arrivals and uniform experiment release times, we describe 4 greedy algorithms: earliest due date (EDD), least cost last (LCL), apparent tardiness cost (ATC) and apparent tardiness cost for participant arrival (ATCPA). These algorithms are adapted from the algorithms described in Section 3.3.3. Instead of assuming participants arrive sequentially, participants have varying arrivals times. Similar to the first scenario, all algorithms are offline and is assumed that the participant arrival sequence is known prior to running the algorithms.

Greedy Algorithm based on Earliest Due Date: The first greedy algorithm that we describe assigns incoming participants to the experiment with the earliest due date. However, now the varying arrivals of participants are also considered for the second scenario. Pseudo code for the algorithm is shown in Algorithm 10. Similar to the first scenario, the algorithm requires an array of experiments that lists each experiment's instance size, processing time, due date, priority, and experiment number. Unlike the first scenario, an arrival array is provided that contains participant's varying arrival times.

Algorithm 10 EDD

```
Require: experiments array, arrivals array
1: Sort $experiments$ according to due date in ascending order
2: Sort $experiments$ with same due dates according to priority in descending order
3: Sort $experiments$ with same due dates and same priorities according to instance size in ascending order

4: Assign the next participant to the next experiment in sorted order that has not been started
5: When all experiments finish, calculate the total minimum tardiness using $arrivals$
```

In the above pseudo code, sorting takes $O(N \times log(N))$ time for N experiments and $O(K)$ time to fill in the $K-$sized $orderArr$. Therefore the running time is $O(N \times log(N) + K)$ time with a space requirement of $O(K)$.

Greedy Algorithm based on Least-Cost-Last:

The LCL algorithm selects the experiment with the lowest weighted tardiness to be scheduled last. Pseudo code for the algorithm is given in Algorithm 11. Similar to the previous LCL algorithm from scenario 1, this algorithm requires an array of experiments where each experiment has instance size, processing time, due date, priority, and experiment number. Unlike the other algorithm from scenario 1, this algorithm additionally makes use of an arrival array

Algorithm 11 LCL

```
Require: experiments array, arrivals array
1: while $experiments$ is not empty do
2:     for every experiment in $experiments$ do
3:         Assume the experiment is the last processed experiment and calculate its weighted tardiness.
4:         Record the experiment with the smallest weighted tardiness value.
5:     end for
6:     Set the found experiment with least weighted tardiness cost as the last experiment to be assigned.
7:     Remove the experiment with least tardiness cost from $experiments$.
8: end while
9: When all experiments finish, calculate the total minimum tardiness using $arrivals$
```

The algorithm has a nested loop that runs for $O(N^2)$ time and it also requires $O(K)$ time since N experiments require K participants to start. Therefore the overall procedure runs in

$O(N^2 + K)$ time with a space requirement of $O(K)$.

Greedy Algorithm Based on Apparent Tardiness Cost:

The ATC algorithm calculates the priority index of every available experiment, and assigns the participant to the experiment with the highest priority index value. Time-dependent index values for the available experiments are recalculated whenever a new participant arrives for assignment. The inputs for all the introduced greedy algorithms are the same and include an array of experiments to be scheduled and an arrivals array giving the arrival time of each participant. The greedy algorithm based on Apparent Tardiness Cost and Apparent Tardiness Cost for Participant assignment also use a lookahead parameter α and the total number of participants needed for all experiments (K). Unlike the first scenario, in the calculation of total weighted tardiness varying arrivals of the participants are also considered. Pseudo code for the algorithm is given in Algorithm 12 below.

Algorithm 12 ATC

```
Require: experiments array, $\alpha$, $K$, arrivals array
 1: for $t = 0 \to K - 1$ do
 2:     for every experiment in $experiments$ do
 3:         Calculate $WSPT$ as experiment.PR/experiment.PT.
 4:         Calculate $MS$ as $\max$(experiment.DD−experiment.PT−t,0).
 5:         Calculate average processing time of all $experiments$ as $avgPT$.
 6:         Set $I(t) = WSPT \times exp(-MS/(\alpha \times avgPT))$
 7:         Store the experiment with highest $I(t)$ value.
 8:     end for
 9:     Assign participants to the experiment with highest $I(t)$ value.
10:     Remove the experiment with highest $I(t)$ value from $experiments$.
11: end for
12: When all experiments finish, calculate the total minimum tardiness using $arrivals$
```

The inner loop runs in $O(N^2)$ time and K participants requires $O(K)$ time. Therefore the overall procedure runs in $O(K \times N^2)$ time with $O(K)$ space requirement.

Apparent Tardiness Cost for Participant Assignment:

Apparent tardiness cost for participant assignment is a mixture of weighted shortest processing time (WSPT), minimum slack (MS), earliest due date (EDD) and least cost last (LCL) algorithms. For scenario 2, the working principle of the algorithm remains same but an arrivals array is introduced to consider the varying arrivals of the participants. Pseudo code for the algorithm is given in Algorithm 13 below. Similar to ATC, the algorithm requires an array of experiments to be scheduled, a lookahead paramter α, the total number of participants needed for all experiments

(K) and an arrivals array to keep the varying arrivals of participants.

Algorithm 13 ATCPA

```
Require: experiments array, $\alpha$, $K$, arrivals array
 1: for $t = 0 \to K - 1$ do
 2:     for every experiment in $experiments$ do
 3:         Calculate $WSPT$ as experiment.PR/experiment.PT.
 4:         Calculate $MS$ as $\max$(experiment.DD−experiment.PT−t,0).
 5:         Calculate average processing time of all $experiments$ as $avgPT$.
 6:         Calculate $WT$ as $\max$(experiment.PR+$t$-experiment.DD, 0) $\times$ experiment.PR
 7:         Set $I(t) = WSPT \times exp(-MS/(\alpha \times avgPT)) \times WT/experiment.DD$
 8:         Store the experiment with highest $I(t)$ value.
 9:     end for
10:     Assign participants to the experiment with highest $I(t)$ value.
11:     Remove the experiment with highest $I(t)$ value from $experiments$.
12: end for
13: When all experiments finish, calculate the total minimum tardiness using $arrivals$
```

The ATCPA algorithm for the second scenario has the same structure as the the algorithm for the first scenario, and therefore the running time of the algorithm is again in $O(K \times N^2)$ time with $O(K)$ space requirement.

4.2.4 Integer-Linear Programming Solution

We describe an ILP solution to the problem of minimizing weighted lateness of experiments with varying participant arrivals and uniform experiment release times problem. Similar to the ILP formulation described in Section 3.3.4, there are two types of binary variables $x_{j,t}$ and $z_{j,t}$ used in the formula where $x_{j,t}$ is set to 1 if the participant at time t is assigned to experiment j and does not make the experiment start, and 0 otherwise. Similarly $z_{j,t}$ is set to 1 if the participant arriving at time t is assigned to experiment j makes the experiment start and 0 otherwise. Unlike the previous ILP formulation, now we introduce a vector $arrivals$ that contains the varying arrival times of the participants. We also introduce $noeffectarrivals$ sequence which contains the varying arrivals of participants and these arrivals is a subset of $arrivals$ that contain the last possible time for a participant assignment which does not result in any experiment starting. Since this is an offline algorithm, we know the number of experiments, number of participants needed and their varying arrival times. If the $arrivals$ contain K arrival times, $noeffectarrivals$ contain the first $K - 1$ arrivals. The array $arrivals$ contains arrival time t_a for every arrival time t_{nea} in

$noeffectarrivals$ where $t_a = tnea$ and this ensures that algorithm does not start the calculation of total weighted tardiness of any experiments because the last experiment does not start its execution.

The complete ILP model for minimizing the total weighted tardiness is:

$$min\ TWT = \sum_{j=1}^{N} \sum_{t=0}^{arrivals} PR_j \times Tardiness_{j,t} \times z_{j,t} \tag{4.2}$$

subject to

$$\sum_{t=0}^{noeffectarrivals} x_{j,t} = IS_j - 1 \quad \forall j \in N \tag{4.3}$$

$$\sum_{j=1}^{N} x_{j,t} \leq 1 \quad \forall t \in noeffectarrivals \tag{4.4}$$

$$\sum_{t=0}^{arrivals} z_{j,t} = 1 \quad \forall j \in N \tag{4.5}$$

$$\sum_{j=1}^{N} z_{j,t} \leq 1 \quad \forall t \in arrivals \tag{4.6}$$

$$\sum_{j=1}^{N} x_{j,t} + \sum_{k=1}^{N} z_{k,t} \leq 1 \quad \forall t \in arrivals \tag{4.7}$$

$$x_{j,t} \times t \leq \sum_{m=0}^{arrivals} z_{j,m} \times m \quad \forall j \in N,\ t \in arrivals \tag{4.8}$$

$$x_{j,t} \in \{0,1\} \quad \forall j \in N,\ t \in arrivals \tag{4.9}$$

$$z_{j,t} \in \{0,1\} \quad \forall j \in N,\ t \in arrivals \tag{4.10}$$

Similar to scenario 1, equation 4.3 ensures that the number of participant assignments of the first type made to experiment j does not exceed the instance size minus 1. Equation 4.4 ensures that at most one participant assignment takes place at any participant arrival time from $noeffectarrivals$. Equation 4.5 states that each experiment can start only once. Equation 4.6 ensures that at most one experiment can start at any participant arrival time from $arrivals$. Equation 4.7 ensures that at most one of $x_{j,t}$ and $z_{j,t}$ can equal 1 at any participant arrival time from

$arrivals$. Equation 4.8 ensures that each $x_{j,t}$ is set to 1 before setting $z_{j,t}$ to 1 for every experiment j, since the assignment times of the participants from $arrivals$ which do not result in an experiment starting must be smaller than the last participant's assignment time for each experiment.

4.3 Experimental Results

In this section we report our experimental evaluation results for the algorithms described in Section 4.2. Similar to 3.5, we report and compare the performance of the exhaustive enumeration, dynamic programming, greedy and integer-linear programming solutions. We then give the size of the largest solved instance for each algorithm, and investigate how close the solutions they provide are to optimal. We follow the same experimental methodology as we did in 3.5. The performance of each algorithm is evaluated over a synthetic dataset which contains experiments with varying participant arrivals and release times as well as varying instance sizes, processing times, priorities and due dates. Each benchmark instance contains a set of N experiments, and we let K be the number of participants needed to make all experiments start. We use the same benchmark properties as in 3.5 in terms of experiment sizes, segment sizes and benchmark size.

Similar to Section 3.5 the first set of experiments investigate the percentage of the benchmark that the algorithms can process within a fixed amount of time. The execution time of the algorithms is limited to either 1 minute or 10 minutes. Results are reported in Table 4.1 for the 1-minute time restriction and in Table 4.2 for the 10-minute time restriction.

Table 4.1: The number and percentage of the instances that are processed within one minute.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
	5	0	0%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
Exhaustive	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
DP	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	0	0%	0	0%
EDD, LCL	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	0	0%	0	0%
ATC, ATCPA	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
	5	50	100%	50	100%	50	100%
	10	50	100%	7	14%	0	0%
	15	36	72%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
ILP	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

Table 4.2: The number and percentage of the instances that are processed within ten minutes.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	2	4%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
EDD, LCL	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
ATC, ATCPA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	3	6%	3	6%	3	6%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	15	30%	0	0%
	15	50	100%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

As can be seen in Tables 4.1 and 4.2, when the experiment size (ES) or number of experiment (N) increase, the total number of participants needed (K) for each benchmark instance increases and the number of solved instances within each segment decreases. As expected, the worst performance is observed for the exhaustive solution. The second worst results are observed with the dynamic programming algorithm. Since the dynamic programming algorithm's complexity is exponential with respect to K and quadratic with respect to N, these results are expected. ILP performs better than dynamic programming for both 1-minute and 10-minute tests yet it is slower than the greedy algorithms. Based on the results from both tables, the greedy algorithms are the fastest algorithms.

Similar to Section 3.5 the second set of experiments is conducted to determine how close the solutions provided by these algorithms are to optimal. The solutions provided by the dynamic programming algorithm are used as a baseline since it provides optimal solutions and runs more quickly than the exhaustive algorithm. We applied the dynamic programming algorithm for 30 minutes to every instance in a subset of a benchmark segment that was not solved by any of our exact algorithms in less than 10 minutes. The instances for this test consist of 35 instances from the benchmark segment with $N = 10$ and $4 \leq ES \leq 6$. We then applied the other algorithms to this benchmark segment for 1 minute and 10 minutes for every instance. We report the difference between the cost of the solutions found by a target algorithm and the optimal cost, divided by the optimal cost. The results from the comparison of the exhaustive, integer-linear programming and greedy algorithms are provided in Figure 4.3

As can be seen from Figure 4.3, when the execution time increases from 1 minute to 10 minutes, exhaustive and ILP algorithms find more accurate results. In both the 1-minute and 10-minute tests, the ILP algorithm is shown to be more accurate than the exhaustive algorithm. Unlike the optimality tests for scenario 1, ILP performs worse than greedy algorithms in the 1-minute tests. This is an expected result since the ILP algorithm improves the results by time but no change is observed for the case of greedy algorithms because they are very fast heuristics and find results within 1 minute.

The exhaustive and ILP algorithms find solutions that are on average 11.07% and 9.05%, respectively, from optimal in the 1-minute tests, and 10.96% and 0.01%, respectively, from optimal in the 10-minute tests. The EDD, LCL, ATC and ATCPA algorithms are 6.29%, 3.36%, 0.09% and 3.59%, respectively, from optimal for both the 1-minute and 10 minutes tests.

Intuitively, we would expect that the exhaustive and ILP algorithms would give the best solutions. However, these algorithms are very slow and they are not able to finish their execution within 10 minutes. ATC performs better than other greedy algorithms because it makes use of a look-ahead parameter. This parameter enables lead time estimation which reflects possible future load in the experiments.

For harder instances where we could not find exact solutions within 10 minutes, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. Table 4.3 shows the average difference between the LP lower bound and optimal ILP solutions for small instances where optimal results were obtained

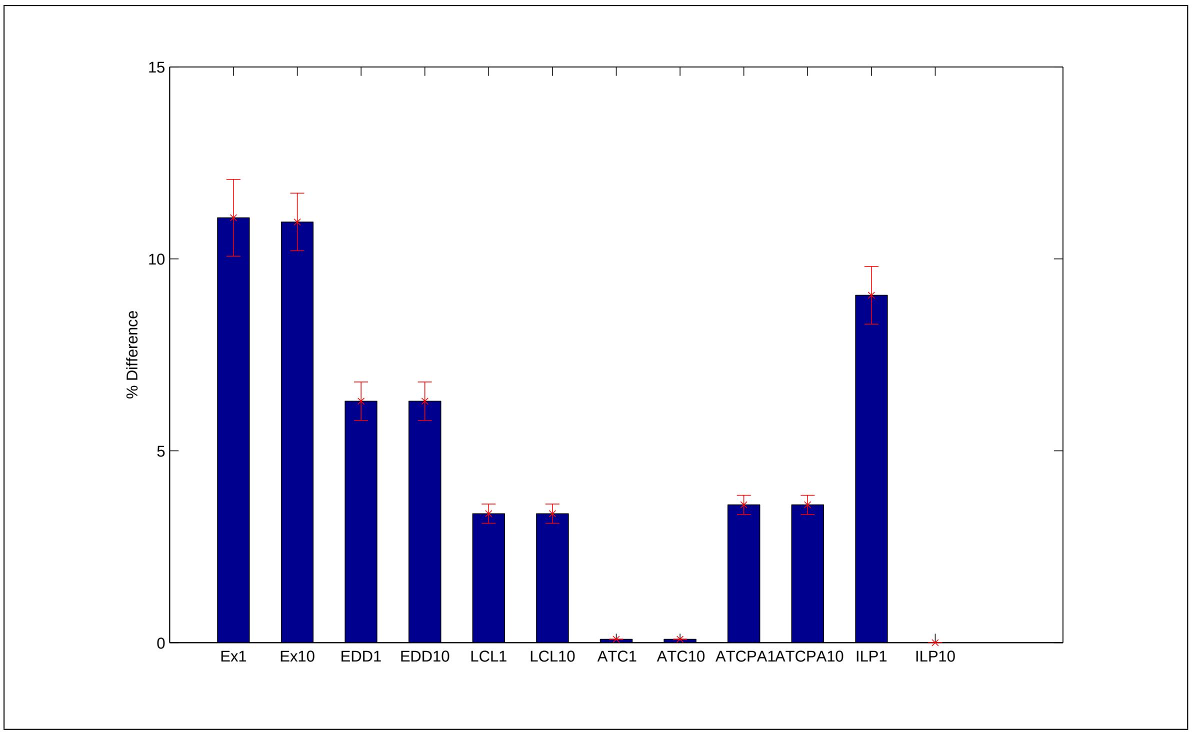


Figure 4.3: Percentage difference between the studied algorithms and dynamic programming after after running its instances with 1-minute and 10-minute time restrictions. Ex1 and Ex10 stands for 1-minute and 10-minutes execution results of the exhaustive algorithm. EDD1, EDD10, LCL1, LCL10, ATC1, ATC10, ATCPA1, ATCPA10 and ILP1, ILP10 stand for 1-minute and 10-minute execution results of the greedy and ILP algorithms respectively.

using ILP within 10 minutes.

The average difference between the LP lower bound and the optimal results is 72.4%. In the same way, the average difference between the LP lower bound and the solutions computed by the EDD, LCL, ATC and ATCPA algorithms is found to be 77.2%, 74.7%, 72.0%, and 74.6% respectively.

Table 4.3: Comparison of LP and ILP lower bounds for small instances as well as LP and greedy algorithms on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. ILP	70.7%	70.5%	76.0%
LP vs. EDD	82.4%	81.2%	66.2%
LP vs. LCL	81.0%	77.4%	65.6%
LP vs. ATC	77.2%	74.8%	64.0%
LP vs. ATCPA	80.6%	77.8%	65.4%

According to the 1-minute and 10-minute optimality tests and the LP comparison tests, we observe that on average the ATC heuristic performs better than the other greedy heuristics. By taking the ATC algorithm as a baseline algorithm, we performed another test to determine if the difference between the compared greedy heuristics and the baseline greedy heuristic is statistically significant. In Table 4.4 we present the number of instances for which the baseline greedy heuristic performs better (<), equal (=) or worse (>) than the compared greedy heuristic. In this table we also report if the difference between the compared greedy heuristics is statistically significant by applying the nonparametric Wilcoxon test, as we did for scenario 1.

Table 4.4: Comparison results of other greedy heuristics with ATCPA on different benchmark segments.

	EDD				LCL				$ATCPA$			
N	$<$	$=$	$>$	$p-value$	$<$	$=$	$>$	$p-value$	$<$	$=$	$>$	$p-value$
5	89	29	36	0.0006	28	58	64	0.0083	104	10	36	0.0001
10	145	1	4	0.0001	63	45	42	0.0019	133	0	17	0.0001
15	150	0	0	0.0001	108	4	38	0.0001	135	1	14	0.0001
50	150	0	0	0.0001	150	0	0	0.0001	139	0	11	0.0001
100	150	0	0	0.0001	149	0	1	0.0001	142	1	7	0.0001
200	150	0	0	0.0001	140	0	10	0.0001	141	0	0	0.0001
500	150	0	0	0.0001	93	0	57	0.0001	146	0	4	0.0001
1000	150	0	0	0.0001	51	1	98	0.0001	107	0	43	0.0001
2000	150	0	0	0.0001	27	0	123	0.0001	106	0	44	0.001
5000	150	0	0	0.0001	11	0	139	0.0001	150	0	0	0.001

From the results obtained from Table 4.4, we conclude that the ATC greedy heuristic outperforms the other greedy heuristics and provides significantly better results for benchmark instances up to 100 experiments which is the highest number of experiments that we managed to run the optimality tests. Similar to scenario 1, we observe that greedy algorithms outperformed the other studied methods in both the 1-minute and 10-minute tests. The ILP algorithm is the best performing algorithm in the 10-minute tests, whereas ATC gives the best average for both execution time and optimality performance. These results imply that the greedy algorithms should be chosen to solve assignment problems based on varying participant arrivals and uniform experiment release times and ATC can be a good choice for this purpose.

Chapter 5

Participant Assignment with Uniform Arrivals and Varying Release Times

In this section we consider the third alternative participant assignment scenario and present our participant assignment algorithms. These algorithms are designed for the scenario where participants arrival times are uniform, and experiments' release times vary. We evaluate the performance of these algorithms over a synthetic dataset for a wide range of instances and report the results.

5.1 Background

The problem of minimizing weighted tardiness with uniform participant arrivals and varying experiment release times is visualized in Figure 5.1. If we consider the multi-machine participant assignment problem as a job scheduling problem on parallel machines, we can assume that each job corresponds to a participant and each parallel machine represents one experiment. In this varying assignment problem we assume that participants can be assigned to experiments that start at the same time and in multi-machine job scheduling problem, there are more than one machines that can be used for job scheduling. If parallel machines are not fixed, they start to be considered at different times during the scheduling and this serves our purpose since we assume experiments are represented by the parallel machines. Therefore this variation of the participant assignment problem is similar to the job scheduling problem on parallel machines where the number of parallel machines is not fixed (i.e. machines have unequal ready times).

In the further sections, this problem is considered as an offline problem where everything about the assignment problem is known. The problem is also considered in online environments where the experiments' properties and time and number of arriving participants are unknown.

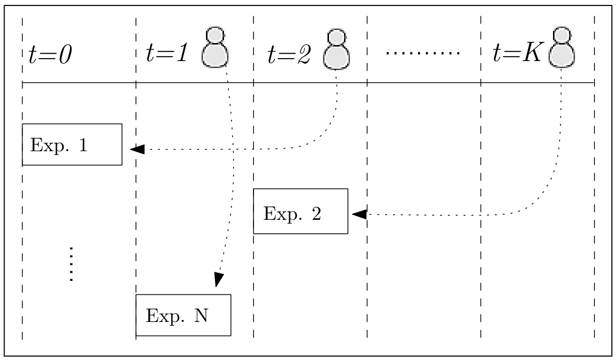


Figure 5.1: Assignment of participants with varying experiment release times and uniform arrivals.

Lenstra et al. [57] propose an offline approximation algorithm for scheduling n jobs to parallel machines where the number of parallel machines is not fixed. Their goal was to find a schedule to minimize the makespan and they come up with a 2-approximation algorithm that solves this problem.

Arkin and Silverberg [58] study the parallel machine scheduling problem where the number of available parallel machines is not fixed. They describe a dynamic programming-based exact offline algorithm to solve this problem. However, their algorithm becomes very inefficient as the number of used parallel machines increases.

Gabrel [59] describe offline scheduling algorithms to schedule jobs on identical parallel machines where the number of machines is not fixed and the number of existing machines depends on the number of considered jobs. The goal was to define heuristics to solve this problem. The experimental results indicate that these heuristics can achieve good results.

Xi and Jang [60] describe an Apparent Tardiness Cost based dispatching rule to minimize the total weighted tardiness on identical parallel machines with unequal ready times. Experimental results show that two of their proposed modified apparent tardiness cost algorithms outperform the existing apparent tardiness cost algorithm for reducing the total minimum weighted tardiness.

Yang-Kuei et al. [61] consider the problem of scheduling n jobs on m unrelated parallel machines with release dates to minimize total weighted tardiness. They develop several mixed integer programming models for these scheduling problems to find the optimal solutions for small problem instances which contain 20 jobs. They also describe several dispatching rules to find good solutions quickly for large problem instances. They compare these dispatching rules with other existing dispatching rules. According to their experimental results the proposed dispatching rules outperform other existing dispatching rules for problem instances of all sizes.

According to Lawler et al. [62] parallel machine job scheduling problem for minimizing the makespan when the number of parallel machines is not fixed is NP-hard.

5.2 Algorithms for Optimized Participant Assignment

In this section, we present our algorithms for the scenario of uniform arrivals of participants to the experiments with varying release times. We solve this problem using exhaustive enumeration, dynamic programming, greedy algorithms, branch and bound and integer-linear programming. In the following sections we introduce details of these algorithms and then we give our experimental results.

5.2.1 Exhaustive enumeration

Our exhaustive algorithm for the scenario of uniform arrivals of participants to the experiments with varying release times is similar to the exhaustive algorithm designed for the previous 2 scenarios. The difference between the exhaustive algorithm for scenario 3 and the other two exhaustive algorithms is the fact that for the third scenario experiments' release times are different from one each other, which changes the calculation total weighted tardiness.

As in scenario 1 and 2 the inputs to each algorithm includes an array of experiments where each experiment contains parameters such as the number of participants assigned to each experiment, instance size, processing time, due date and priority. However in scenario 3, the number of completed experiments is an additional input. Unlike the other scenarios scenario 3, experiments also contain release times to calculate the total weighted tardiness value by considering experiments with varying release times and uniform arrivals of the participants.

In the given pseudo code below, line 1 ensures that the recursion runs until all experiments are completed. Lines 2-8 traverse the experiment array and recursively allocates the current participant to each experiment as long as we can allocate the experiment. In lines 10-11 for each completed permutation of participant-experiment assignment, the weighted tardiness of the complete assignments is calculated by making use of experiments' varying release times and checked to see if it is the minimum weighted tardiness that has been found so far.

The base case occurs when all the experiments are completed. Similar to the first two exhaustive algorithms, if K is the number of assigned participants, the overall function runs within a polynomial factor of O(K!) in the worst case, the factorial of the number of total assigned participants, assuming every experiment requires at least one participant to start.

Algorithm 14 Exhaustive

```
Require: count of started experiments, experiments array
 1: if count of started experiments $<$ number of $experiments$ then
 2:     for every experiment $i$ do
 3:         Assign the next participant to the any empty slot of experiment $i$
 4:         if Assignment process results in the experiment to start then
 5:             Increment the count of started experiments
 6:         end if
 7:         $Exhaustive$(count of started experiments, $experiments, arrivals$)
 8:     end for
 9: else
10:     Calculate the tardiness value of the found assignment order by considering varying release times of
        the experiments
11:     Store the minimum found tardiness value
12: end if
```

5.2.2 Dynamic Programming Algorithm

The recursive algorithm for the scenario of uniform participant arrivals and varying experiment release times is similar to the recursive algorithm of the previous scenarios. Therefore, the obtained recursion tree and the exact solution approach with dynamic programming to solve the problem is also similar.

Structure of the Optimal Solution:

The definitions of the sorted sequence of experiment numbers S, the optimal solution G, the subsequence $S - E$ where E is a given sequence of experiment numbers are also the same as the definitions for the first two scenarios. The structure of the optimal solution's structure is also the same as those in the first and second scenarios, as described in 3.3.2 and 4.2.2. The existence of optimal substrucutre is based on contradiction. The overall solution is first participant's weighted tardiness value plus the optimized total weighted tardiness value of the sub-problem where the various release times of the experiments are also taken into consideration.

Recursive Definition of the Optimal Solution: The recursive solution of the problem is also similar to the recursive solutions of the first two scenarios. Given S, experiment number i, the experiments' release time array rel; $G(i, S, rel(i) + t)$ is the total minimum weighted tardiness assuming that the participant arriving at time $rel(i)+t$ is assigned to experiment i and the remaining participants are assigned to experiment numbers from S. Then we can recursively define the optimal solution as:

$$G(i, S, rel(i) + t) = \begin{cases} WT_i(rel(i) + t) & \text{if } S \text{ is empty} \\ min_j\{G(j, S - (j), rel(j) + t + 1)\} & \text{if } i \text{ in } S \\ WT_i(rel(i) + t) + min_j\{G(j, S - (j), rel(j) + t + 1)\} & \text{if } i \text{ not in } S \text{ and } S \text{ is not empty} \end{cases} \tag{5.1}$$

Similar to 3.3.2 and 4.2.2 the above statement states that there are three cases to be considered to determine the total minimum weighted tardiness. Unlike the 3.3.2 and 4.2.2 instead of considering time sequentially or arriving in varying time from the beginning as time t, we consider the time to be sequentially after the release date as . Instead of considering time periods as t, we consider time periods that vary t.

For this reason we maintain an array of experiment release times as an input and we are considering each experiment release date in the array one by one.

Computing the Value of the Optimal Solution Bottom Up:

Similar to Algorithms 2 and 9, the optimal solution for varying arrivals and uniform release times is computed bottom up by making use of experiments array and the S sequence. Total tardiness value in the algorithm is calculated by making use of the release time values of the experiments. Pseudo code for the dynamic programming solution to the problem of minimizing total weighted tardiness for uniform arrivals of the participants and varying release times experiments is provided as Algorithm 15 below. Given the experiment numbers and arrival array, our goal is to find

$$min\{G(i, S, rel(i) + 0)\} \ \forall \text{ experiment number } i$$

In the Algorithm 15, lines 4, 10 and 12 differ from the Algorithm 2 and 9 in the sense that these lines compute the weighted tardiness value by making use of the provided rel release times array.

Constructing Optimal Solution from Computed Information:

Similar to the optimal solution construction approach of the first two scenarios' dynamic programming algorithms, we find the optimal total weighted tardiness value by making use of the tuples. Having identified $(1, i)$ at time $t = rel(i) + 0$, we can backtrack our optimal solution by finding the experiment j for each tuple that minimizes the expression.

Running Time and Space Requirements:

Algorithm 15 Dynamic

```
Require: experiments array, S sequence, A
 1: Initialize $K$ as the number of participants needed to make all experiments start
 2: Initialize $G$ as empty to hold weighted tardiness values
 3: for every experiment $i$ do
 4:     Set $G(i, \emptyset, rel(i)+t)$ = weighted tardiness of experiment $i$ assuming it finishes at time $t$ for $m = K-1$
 5: end for
 6: for every subsequence of $S$ do
 7:     for every experiment $j$ from $S$ do
 8:         Set $m = K - 1 -$ size of subsequence
 9:         if $j$ is in subsequence then
10:             $G(j, subsequence, rel(j) + t) = min\{G(k, subsequence - (k), rel(k) + t + 1)\}$ $\forall$ $k$ in subsequence
11:         else
12:             $G(j, subsequence, rel(j) + t)$ =weighted tardiness of $j$ + $min\{G(k, subsequence - (k), rel(k) + t + 1)\}$ $\forall$ $k$ in subsequence
13:         end if
14:     end for
15: end for
```

The "Dynamic" algorithm that is used for computing the optimal solution has the same runtime complexity as the dynamic programming algorithm for scenarios 1 and 2, $O(K \times 2^K)$ where K is the summation of all experiments' instance sizes. The procedure that is used to construct the optimal solution takes $O(K^3)$ time and thus the overall procedure has a running time of $O(K^2 \times 2^K)$. Similar to scenarios 1 and 2, the total space requirement is also $O(K \times 2^K)$.

5.2.3 Offline Greedy Algorithms

For the case of uniform arrivals of participants and varying experiment release times, we considered both online and offline greedy algorithms. In the offline greedy algorithms, all the problem input data including participants arrival times, the complete participant arrival sequence, the number of experiments, their instance size, release time, processing time and due date are known in advance. The algorithms considered in this section are minimum slack (MS), minimum slack with priority (MSP), next fit (NF), smallest critical ratio (SCR), smallest critical ratio with priority (SCRP), smallest instance size with remaining time (SIRT) and maximum priority with remaining attributes (MPRA). Since this scenario considers varying experiment release times, we pay attention to how algorithms make use of varying experiment release times while assigning participants to experiments. The nextfit algorithm does not consider varying experiment release times and is used to compare performance with other algorithms. We consider algorithms like minimum slack and smallest critical ratio because these algorithms use real time in their formulation which is easy to tune and thus we can convert these algorithms to be offline by replacing time with a experiment release times within the formula.

5.2.3.1 Minimum Slack

The minimum slack algorithm assigns participants to experiments by giving priority to the slack time of the assignment process. Slack time is the difference between an experiment's due date and the current time, minus the experiment's processing time. However, for the offline case of the algorithm, we consider minimum slack to be the difference between an experiment's due date and its release date minus its processing time.

In Algorithm 16 the algorithm uses an experiment array to calculate minimum slack values of the experiments. Lines 1 through 7 calculate the slack for every experiment in the $experiments$ array and assign participants to the experiment with minimum slack at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 16 Minimum Slack

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $slack$
4:         Find the experiment with minimum $slack$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ due to two for loops and the total space requirement is $O(1)$.

5.2.3.2 Minimum Slack with Priority

Similar to minimum slack algorithm, this algorithm considers slack values of the experiments. Unlike minimum slack, this algorithm gives priority to the experiments with higher priority. In order to do that we divide the slack value of each experiment by its priority, select the experiment with minimum priority, and assign the next incoming participants to that experiment. Thus the slack/priority values of the experiments with higher priority will be less than that for experiments with lower priority. The algorithm for minimum slack with priority is given in Algorithm 17.

The algorithm uses an experiments array to calculate an experiment's minimum slack values with priority. Lines 1 through 7 are used to calculate the $slack \div$ experiment's priority for every experiment in the $experiments$ array and assign participants to the experiment with minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 17 Minimum Slack

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $slack$
4:         Find the experiment with minimum $slack \div$ experiment's priority
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

Similar to offline minimum slack algorithm, for N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.3.3 NextFit

In the NextFit algorithm for participant assignment, an incoming participant is assigned to the next available experiment. This eliminates the need to consider experiments that were already allocated participants while assigning the next incoming participant. In order to do that we keep track of the experiment that was mostly recently assigned. Pseudo code for this approach is given in Algorithm 18.

The algorithm uses a global current experiment number to keep track of the current experiment to which a participant has been assigned, an experiments array, and a participants array to assign each incoming participant to a suitable experiment. Lines 1 through 7 assign incoming participant to the next suitable experiment. Line 8 calculates the total minimum weighted tardiness.

For K incoming participants, the run time complexity of the algorithm is $O(K)$ due to one for loop and the total space requirement is $O(1)$.

Algorithm 18 NextFit

```
Require: experiments array, participants array, current experiment number
1: for every participant $i$ do
2:     if current experiment didn't start then
3:         Assign participant $i$ to the current experiment
4:         Set current experiment to be the next experiment
5:     end if
6: end for
7: Calculate the total minimum weighted tardiness by making use of experiment release times
```

5.2.3.4 Smallest Critical Ratio

Similar to minimum slack, smallest critical ratio algorithm assigns a participant to the experiment with the minimum critical ratio, but arranges the slack in ratio form. The critical ratio is defined as time remaining $\div$ work remaining where time remaining is the difference between an experiment's due date minus the current time, and work remaining is the remaining processing time. However for the offline algorithms we use the current time as the experiment's release time.

Algorithm 19 uses an experiment array to calculate critical ratio values of the experiments. Lines 1 through 7 calculate the critical ratio for every experiment in the $experiments$ array and assign participants to the experiment with minimum critical ratio at every time. Finally, we calculate total minimum weighted tardiness in line 8.

Algorithm 19 Smallest Critical Ratio

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $critical\,ratio$
4:         Find the experiment with minimum $critical\,ratio$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ due to two for loops and the total space requirement is $O(1)$.

5.2.3.5 Smallest Critical Ratio with Priority

Similar to the smallest critical ratio algorithm, this algorithm considers the critical ratio values of the experiments. Unlike the smallest critical ratio algorithm, and similar to minimum slack with priority algorithm, this algorithm gives priority to the experiments with higher priority. In order to do that priority of the experiment is added to denominator of the critical ratio and multiplied by the remaining processing time. Adding priority to the denominator gives smaller values for experiments with higher priority and thus this approach is meaningful. The algorithm for the smallest critical ratio with priority is given in Algorithm 20.

Algorithm 20 uses an experiments array to calculate critical ratio $\times \frac{1}{priority}$ values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 20 Smallest Critical Ratio With Priority

caption

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $critical\,ratio \times \frac{1}{priority}$
4:         Find the experiment with minimum $critical\,ratio \times \frac{1}{priority}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.3.6 Smallest Instance Size with Remaining Time

In this algorithm, we consider the instances sizes as well as the remaining time in experiments. Since this is an offline algorithm, the remaining time is the difference between an experiment's due date and its release date. In order to consider instance size and remaining time together, we multiply these two values and select the experiment with the smallest resulting value. Since we want to start smaller experiments with less remaining time sooner, this approach is meaningful. The pseudo code of this approach is given at Algorithm 21.

Algorithm 21 uses an experiment array to calculate an experiment's instance size $\times$ (experiment's due date $-$ experiment's release time) values. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 21 Smallest Instance Size with Remaining Time

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's instance size $\times$ (experiment's due date $-$ experiment's release time)
4:         Find the experiment with minimum experiment's instance size $\times$ (experiment's due date $-$ experi-
           ment's release time)
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.3.7 Maximum Priority with Remaining Attributes

This algorithm considers all of an experiment's attributes and gives precedence to experiments with higher priority. Since the priority of the experiment has positive effect on maximizing the overall value, it is in the numerator and the other attributes are in the denominator. In the denominator, we multiply the remaining time, processing time and remaining instance size. When remaining time or processing time or remaining instance size is small, an experiment can start early and may finish early. Therefore placing this product in the denominator is meaningful. Pseudo code for this approach is given in Algorithm 22.

Algorithm 22 uses an experiments array to calculate $priority \div ((due\,date - release\,time) \times processing\,time \times remaining\,instance\,size)$ values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

Algorithm 22 Maximum Priority with Remaining Attributes

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's $\frac{priority}{(due\,date - release\,time) \times processing\,time \times remaining\,instance\,size}$
4:         Find the experiment with minimum $\frac{priority}{(due\,date - release\,time) \times processing\,time \times remaining\,instance\,size}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

5.2.4 Online Greedy Algorithms

For the case of online greedy algorithms, the entire problem input data is not available at the beginning and participants are processed one by one. Their arrival time, participant arrival sequence and the experiment properties such as the number of experiments, their instance size, release time, processing time and due date are unknown prior to execution of the algorithm. Incoming participants are only assigned to released experiments for which we can obtain experiment properties. On the other hand, if there are no available experiments at the time of a participant's arrival, the participant waits until the release of a new experiment. Similar to offline greedy algorithms we consider seven online greedy algorithms and they are minimum slack (MS), minimum slack with priority (MSP), next fit (NF), smallest critical ratio (SCR), smallest critical ratio with priority (SCRP), smallest instance size with remaining time (SIRT) and maximum priority with remaining attributes (MPRA).

5.2.4.1 Minimum Slack

Similar to the offline minimum slack algorithm we calculate the slack for each experiment and assign participants to the experiment with the smallest slack. Unlike the offline algorithm, slack time is the amount of time between an experiment's due date and today's date minus the experiment's processing time. To differentiate these two versions of slacks, we call the online version as $real\,slack$. In Algorithm 23 we give the pseudo code for the online minimum slack algorithm.

The algorithm uses an experiments array to calculate minimum slack values of the experiments. Lines 1 through 7 calculate the real slack for every experiment in the $experiments$ array and assign participants to the experiment with minimum real slack at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 23 Minimum Slack

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\ slack$
4:         Find the experiment with minimum $real\ slack$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.4.2 Minimum Slack with Priority

Similar to offline minimum slack algorithm with priority we consider both slack and priority in this algorithm. The difference is that instead of slack, we consider the $real\ slack$ which is defined in the previous section. We divide the experiment's real slack by its priority and assign participants to the experiment that results in smallest value. Pseduo code for this approach is given in Algorithm 24.

The algorithm uses an experiments array to calculate experiments' minimum slack values with priority. Lines 1 through 7 are used to calculate the $real\ slack \div$ experiment's priority for every experiment in the $experiments$ array and assign participants to the experiment with minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 24 Minimum Slack

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\ slack$
4:         Find the experiment with minimum $real\ slack \div$ experiment's priority
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

Similar to online minimum slack algorithm, for N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.4.3 NextFit

As with the offline version, in the online nextfit algorithm each incoming participant is assigned to the next available experiment. In the offline version of the algorithm, we calculate weighted tardiness by making use of the experiments' release times. However, in the online version, we consider the time when the experiment is ready to launch to calculate the weighted tardiness. Pseudo code for this approach is given in Algorithm 25.

In the pseudo code the algorithm uses a global current experiment number to keep track of the current experiment to which the participant has assigned, an experiments array and a participants array to assign each incoming participant to a suitable experiment. Lines 1 through 7 are responsible for assigning incoming participant to the next suitable experiment. Finally, line 8 calculates the total minimum weighted tardiness.

Algorithm 25 NextFit

```
Require: experiments array, participants array, current experiment number
1: for every participant $i$ do
2:     if current experiment didn't start then
3:         Assign participant $i$ to the current experiment
4:         Set current experiment to be the next experiment
5:     end if
6: end for
7: Calculate the total minimum weighted tardiness by making use of experiment launch times
```

For K incoming participants, the run time complexity of the algorithm is $O(K)$ and the total space requirement is $O(1)$.

5.2.4.4 Smallest Critical Ratio

Similar to the smallest critical ratio offline algorithm, this algorithm selects the experiment with the smallest critical ratio value. For this online version of the algorithm, the critical ratio is changed to be the time remaining $\div$ work remaining, where time remaining is the difference between experiment's due date and the current time. Work remaining is the remaining processing time. In Algorithm 26 we gave the pseudo code for the online version of the smallest critical ratio algorithm. Note that to differ online and offline versions of the critical ratio definition, we called the online critical ratio as $real\,critical\,ratio$ in Algorithm 26.

Similar to the minimum slack algorithm, the smallest critical ratio algorithm assigns the participants to the experiment with the minimum critical ratio, but arranges the slack in ratio form. The critical ratio is defined as the time remaining $\div$ work remaining where time remaining is the difference between an experiment's due date and the current time, and work remaining is the remaining processing time. However for the offline algorithms we let the current time be the experiment's release time.

Algorithm 26 uses an experiments array to calculate the $real\,critical\,ratio$ values for each experiment.

Lines 1 through 7 calculate the real critical ratio for every experiment in the $experiments$ array and assign participants to the experiment with minimum real critical ratio at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 26 Smallest Critical Ratio

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\,critical\,ratio$
4:         Find the experiment with minimum $real\,critical\,ratio$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ due to two for-loops, and the total space requirement is $O(1)$.

5.2.4.5 Smallest Critical Ratio with Priority

Similar to offline smallest critical ratio algorithm, this algorithm calculates experiment's critical ratio divided by experiment's priority. Unlike the offline smallest critical ratio algorithm, in this case $real\,critical\,ratio$ is used. The detailed pseudo code of this approach is given in Algorithm 37.

Algorithm 37 uses an experiments array to calculate real critical ratio $\times \frac{1}{priority}$ values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$

array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

Algorithm 27 Smallest Critical Ratio With Priority

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\,critical\,ratio \times \frac{1}{priority}$
4:         Find the experiment with minimum $real\,critical\,ratio \times \frac{1}{priority}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.4.6 Smallest Instance Size with Remaining Time

Similar to offline version of the smallest instance size with remaining time algorithm, we consider the instance sizes and remaining time of experiments by multiplying them and selecting the experiment with the minimum resulting value. Unlike the offline version of the algorithm, remaining time is now the difference between experiment's due date and current time. The pseudo code of this approach is given in Algorithm 28.

Algorithm 28 requires an experiments array to calculate smallest instance size with remaining time value by making use of an experiment's instance size $\times$ (experiment's due date $-$ current time). Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8.

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.4.7 Maximum Priority with Remaining Attributes

Similar to the offline version of the maximum priority with remaining attributes algorithm, we consider all of an experiment's attributes in the online version of the algorithm. Unlike the offline version of the algorithm, we consider remaining time as the difference between an exper-

Algorithm 28 Smallest Instance Size with Remaining Time

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's instance size $\times$ (experiment's due date $-$ current time)
4:         Find the experiment with minimum experiment's instance size $\times$ (experiment's due date $-$ current time)
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

iment's due date and the current time and multiply it by the processing time and remaining instance size in the denominator. Thus the calculation turns into priority $\div((due\,date\ -\ current\,time) \times processing\,time\ \times\ remaining\,instance\,size)$ Pseudo code for this approach is given in Algorithm 29.

Algorithm 29 uses an experiments array to calculate the Maximum Priority with Remaining Attributes values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally, we calculate the total minimum weighted tardiness in line 8.

Algorithm 29 Maximum Priority with Remaining Attributes

```
Require: experiments array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's $\frac{priority}{(due\,date - current\,time) \times processing\,time \times remaining\,instance\,size}$
4:         Find the experiment with maximum $\frac{priority}{(due\,date - current\,time) \times processing\,time \times remaining\,instance\,size}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

5.2.5 Branch and Bound Solution

The branch and bound method was first proposed by A. H. Land and A. G. Doig [63]. The algorithm solves combinatorial optimization problems by enumerating candidate solutions. The algorithm explores branches of a tree, where each node represents a subset of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and it is ignored if it cannot produce a better solution than the best one found so far by the algorithm.

The principle of the branch and bound algorithm in this study is systematically solving the participant assignment problem by making use of the experiments and participant arrival times. The algorithm does a BFS-like search for the optimal solution, but not all nodes get expanded. To minimize total weighted tardiness, we find a lower bound by finding the least cost of total weighted tardiness even if it is infeasible.

We then search the space from the active node which results in smallest total weighted tardiness value with respect to other children of the parent and we ignore the inferior solutions. For solving the branch and bound problem for participant assignment problem, we were in need of considering more than one state-space tree because of the experiments' varying release times. Since experiment release times may differ for this scenario, more than one state-space tree needs to be considered.

Pseudo code for the branch and bound algorithm is given in Algorithm 30. The algorithm considers all possible experiment-participant arrival time tables. For each row of that table it finds the participant arrival time that gives the smallest total weighted tardiness and continues to expand the state-space tree in this way. The algorithm finds the minimum total weighted tardiness value and the corresponding participant arrival order can be easily found from the state-space tree.

Algorithm 30 Branch and Bound

```
Require: experiments array
 1: Find all possible experiment-participant arrival time tables
 2: for every experiment-participant arrival time table do
 3:     for every experiment $i$ do
 4:         for every participant arrival time $t$ do
 5:             Assign the participant coming at time $t$ to the experiment $i$
 6:             For the remaining experiments consider the remaining experiment arrival times which result in
                the smallest possible total weighted tardiness value
 7:             if Assigned participant to the experiment $i$ at time $t$ results in smallest total weighted tardiness
                value with respect to other $t$ values for that experiment then
 8:                 Accept that participant coming at time $t$ is assigned to experiment $i$
 9:             end if
10:         end for
11:     end for
12: end for
```

For N experiments and K participants, the run time complexity of the algorithm is $O(N \times 2^K)$ since finding all possible experiment-participant arrival time tables takes $O(N \times 2^K)$ time in the worst case. The total space requirement of the algorithm is also $O(N \times 2^K)$ to hold different participant arrival time tables.

5.2.6 Integer-Linear Programming Solution

Similar to scenarios 1 and 2, we solved the assigment problem with integer-linear programming for scenario 3. Unlike the first two scenarios, the solution now considers the varying release times of the experiments. Unlike scenarios 1 and 2, we now use a modified definition of $Tardiness_{i,t}$ in the formulation of the objective function. As it can be seen in the formulation below, we sum the processing time with varying release times of the experiments while calculating the tardiness and select the maximum of 0 and calculated tardiness value since tardiness can not have negative value. Note that instead of summing processing time (PT_i) with the time t that makes an experiment start as in the first scenario, now we sum the processing time with the maximum of the time t that makes an experiment start or experiment's release time since the experiment may start right after its release date if there are existing participants or it may start at a later time t after the release date. This is shown as $max(REL_i, t)$ in the below formulation.

$$Tardiness_i(t) = \begin{cases} 0 & if \quad max(REL_i, t) + PT_i \leq DD_i \\ max(REL_i, t) + PT_i - DD_i & Otherwise \end{cases}$$

Other than the modified version of the objective function, integer-linear programming formulation for uniform arrivals of participant and varying release times of the experiments has no difference with the integer-linear programming solution of scenario 1. Therefore we do not give the formulation details of the integer-linear programming formulation for this scenario again in this section.

5.3 Experimental Results

In this section we report our experimental results for the algorithms we described in Section 5.2. Similar to 3.5, we report and compare the performance of the exhaustive enumeration, dynamic programming, greedy and integer-linear programming solutions. Unlike 3.5, we also report our findings for our branch and bound solution. We then give the size of the largest solved instance foreach algorithm, and investigate how close the solutions they provide are to optimal.

Similar to 3.5 and 4.3 the first set of experiments investigate the percentage of the benchmark that the algorithms can process within a fixed amount of time. The execution time of the algorithms is limited to either 1 minute or 10 minutes. Results are reported in Table 5.1 for the 1-minute time restriction and in Table 5.2 for the 10-minute time restriction. Note that since online and offline greedy algorithms show the same execution performance, they are considered together in the following tables.

Table 5.1: The number and percentage of the instances that are processed within one minute.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	3	6%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
MS, MSP, NF, SCR SCRP, SIRT, MPRA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
B&B	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	6	12%	0	0%
	15	28	56%	5	10%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

Table 5.2: The number and percentage of the instances that are processed within ten minutes.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	6	12%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
MS, MSP, NF, SCR SCRP, SIRT, MPRA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
B&B	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	3	6%
	15	50	100%	24	48%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

According to tables 5.1 and 5.2, when the experiment size (ES) or number of experiment (N) increase, the total number of participants needed (K) for each benchmark instance increases and the number of solved instances within each segment decreases. The worst performance is observed using the exhaustive algorithm, as expected. The second-worst results are observed with the dynamic programming and branch and bound algorithms. Since the runtimes of these algorithms are exponential with respect to the number of experiments, these are the expected results. ILP performs better than dynamic programming and branch and bound for both 1-minute and 10-minute tests, yet it is slower than the greedy algorithms. Based on the results from both tables, the greedy algorithms are the fastest algorithms.

Similar to 3.5 and 4.3 the second set of experiments is conducted to observe how close the solutions found by the algorithms are to optimal. As before, solutions provided by the dynamic programming algorithm are used as a baseline since it provides optimal solutions and runs more quickly than the exhaustive algorithm. The instances for this test consist of 26 instances from the benchmark segment with $N = 15$ and $4 \leq ES \leq 6$. We then applied the other algorithms to this benchmark segment for 1 minute and 10 minutes for every instance. The results from the comparison of the exhaustive, branch and bound, integer-linear programming and online and offline greedy algorithms are provided in Figure 5.2, Figure 5.3, Figure 5.4.

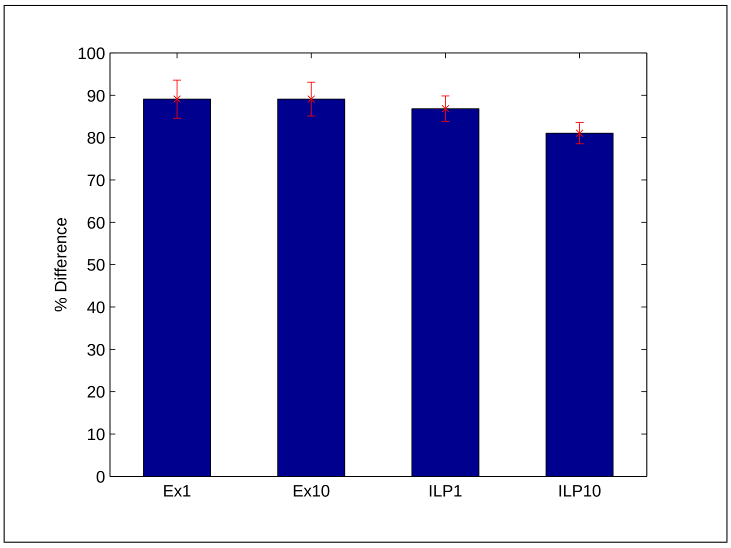


Figure 5.2: Percentage difference between exact algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. Ex1 and Ex10 stands for 1-minute and 10-minutes execution results of the exhaustive algorithm. B&B1, B&B10 and ILP1, ILP10 stand for 1-minute and 10-minute execution results of the branch and bound and ILP algorithms respectively.

As can be seen from Figure 5.2, when the execution time increases from 1 minute to 10 minutes, the exhaustive and branch and bound algorithms remain same in terms of percentage of difference with the dynamic programming algorithm. However, when the time increases from 1 minute to 10 minutes, ILP algorithms find more accurate results.

The intuition behind the optimality performance of exact algorithms is that although these algorithms are slow since they consider the all or partial solutions to find the exact results. This makes the solution time of the algorithms to be very slow and the optimal results couldn't be found within 1 minutes or 10 minutes of time.

We conducted the two-sample t-test to determine whether the results of exact algorithms are significantly different. In Table 5.3 we provide the significance results (p-values) for this test. The results there is a significant difference between EX 1, EX 10, ILP 1 minute and ILP 10 minutes results.

Table 5.3: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	$EX1$	$EX10$	$ILP1$	$ILP10$
$EX1$	1.00	0.56	0.25	0.01
$EX10$	-	1.00	0.11	0.01
$ILP1$	-	-	1.00	0.01
$ILP10$	-	-	-	1.00

We also conduct the optimality tests for online and offline greedy algorithms. Our optimality comparison results of these algorithms for 1 minute and 10 minutes tests are provided in Figure 5.3 and Figure 5.4.

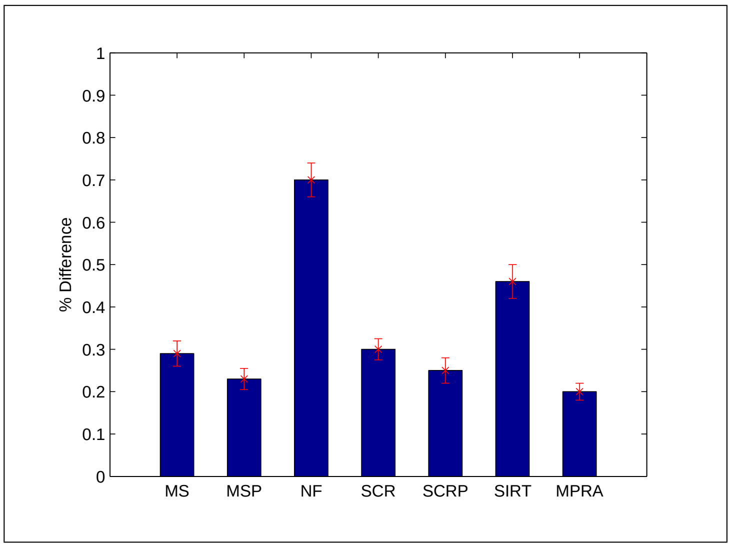


Figure 5.3: Percentage difference between offline greedy algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. MS stands for 1-minute and 10-minutes execution results of the minimum slack since they are the same value. MSP, NF, SCR, SCRP, SIRT, MPRA stand for 1-minute and 10-minute execution results of the offline greedy algorithms respectively.

According to the offline greedy algorithm comparison results shown in Figure 5.3, MPRA gives the best performance and the NF algorithm gives the worse performance. These results were expected since NF was the only algorithm which does not make use of experiment release times and MPRA considers the experiment release times and all other experimental attributes appropriately. MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms find solution that are on the average 0.29%, 0.23%, 0.7%, 0.3%, 0.25%, 0.46% and 0.2%, respectively, from optimal for both 1-minute and 10 minutes tests.

Table 6.4 below shows that the difference between the compared offline greedy algorithms are statistically significant.

Table 5.4: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	MS	MSP	$MPRA$	NF	$SIRT$	SCR	$SCRP$
MS	1.00	0.01	0.01	0.39	0.11	0.01	0.01
MSP	-	1.00	0.01	0.01	0.12	0.01	0.41
$MPRA$	-	-	1.00	0.01	0.01	0.01	0.01
NF	-	-	-	1.00	0.01	0.01	0.01
$SIRT$	-	-	-	-	1.00	0.01	0.01
SCR	-	-	-	-	-	1.00	0.01
$SCRP$	-	-	-	-	-	-	1.00

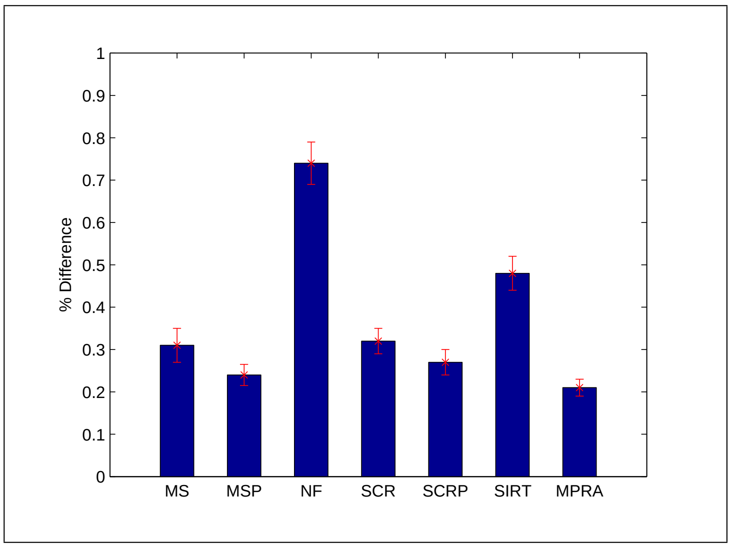


Figure 5.4: Percentage difference between online greedy algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. MS stands for 1-minute and 10-minutes execution results of the minimum slack since they are the same value. MSP, NF, SCR, SCRP, SIRT, MPRA stand for 1-minute and 10-minute execution results of the offline greedy algorithms respectively.

According to the online greedy algorithm comparison results shown in Figure 5.4, MPRA again performs best and the NF algorithm performs worst. MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms find solution that are on the average 0.31%, 0.24%, 0.74%, 0.31%, 0.27%, 0.48% and 0.21%, respectively, from optimal in both 1-minute and 10 minutes tests. Note that solutions found by the online greedy algorithms are slightly worse than the solutions found by the offline algorithms, which was expected since in the case of online algorithms the number of incoming participants and experiment release times are unknown prior to participant assignment.

The intuition behind these results is that for online algorithms, since we do not know the participant arrival sequence and experiment release times, assigning participants to experiments is a more difficult task and this affects the optimality performance results of the algorithms. MSP and SCRP performance results are better than MS and SCR because they make use of experiment priorities as well as experiment due dates and experiment processing times. MPRA is better than other algorithms because it makes use of all experiment properties appropriately while finding the solution.

We report the two sample t-test significance results for online greedy heuristics in Table 6.5. The results show they are significanly different from each other.

Table 5.5: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	MS	MSP	$MPRA$	NF	$SIRT$	SCR	$SCRP$
MS	1.00	0.01	0.01	0.34	0.06	0.01	0.01
MSP	-	1.00	0.01	0.01	0.12	0.01	0.32
$MPRA$	-	-	1.00	0.01	0.01	0.01	0.01
NF	-	-	-	1.00	0.01	0.01	0.01
$SIRT$	-	-	-	-	1.00	0.01	0.01
SCR	-	-	-	-	-	1.00	0.01
$SCRP$	-	-	-	-	-	-	1.00

Similar to previous scenarios, for harder instances where we could not find exact solutions within 10 minutes, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. Table 5.6 shows the average difference between the LP lower bound and optimal ILP solutions for small instances where optimal results were obtained using ILP within 10 minutes.

The average difference between the LP lower bound and the optimal results is 6.6%. In the same way, the average difference between the LP lower bound and the solutions computed by the MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms is found to be 0.4%, 0.3%, 0.7%, 0.4%,

0.3%, 0.5% and 0.2% respectively.

Table 5.6: Comparison of LP and ILP lower bounds for small instances as well as LP and offline greedy soluntions on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. ILP	5.4%	6.8%	7.7%
LP vs. MS	0.3%	0.3%	0.5%
LP vs. MSP	0.3%	0.3%	0.3%
LP vs. NF	0.6%	0.7%	0.8%
LP vs. SCR	0.4%	0.4%	0.4%
LP vs. SCRP	0.3%	0.3%	0.3%
LP vs. SIRT	0.4%	0.5%	0.5%
LP vs. MPRA	0.2%	0.2%	0.2%

The above results indicate that offline MPRA results are closest to the LP lower bound. Table 5.7 shows the average difference between the LP lower bound and the solutions computed by the greedy algorithm based on MS for larger instances. The average difference between the LP lower bound and the solutions computed by the online MS algorithm is 0.4%. In the same way, the average difference between the LP lower bound and the solutions computed by the online MSP, NF, SCR, SCRP, SIRT and MPRA algorithms is found to be 0.3%, 0.7%, 0.4%, 0.4%, 0.5%, 0.3% respectively.

Table 5.7: Comparison of LP lower bounds and online greedy solutions on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. MS	0.4%	0.4%	0.5%
LP vs. MSP	0.3%	0.4%	0.3%
LP vs. NF	0.7%	0.7%	0.8%
LP vs. SCR	0.4%	0.4%	0.5%
LP vs. SCRP	0.3%	0.4%	0.4%
LP vs. SIRT	0.4%	0.5%	0.6%
LP vs. MPRA	0.3%	0.3%	0.3%

The results above show that on average the MSP and MPRA heuristics perform better than the other greedy heuristics. In our final experiment we compare online and offline version of greedy algorithms and report the percentage difference between them in Table 6.8 by dividing the difference between the results obtained by the offline and online version of an algorithm to the results obtained by the online version.

Table 5.8: Comparison of online and offline greedy algorithms.

$Algorithm$	$Difference$
MS	4.9%
MSP	0.0%
NF	10.8%
SCR	8.6%
SCRP	5.8%
SIRT	7.1%
MPRA	2.3%

According to above results the differences between online and offline algorithms are not very significant. The largest difference is seen with nextfit. The results show us that for MSP and MPRA the difference is relatively low which means these algorithms have more stable performance with respect to other algorithms for both online and offline cases. We conclude that, in general, the MPRA greedy heuristic is preferable to other greedy heuristics and provides significantly better results for many benchmark instances for both online and offline cases in terms of optimality and execution performance. Overall, these results imply that the greedy algorithms should be chosen to solve assignment problems based on uniform participant arrivals and varying experiment release times and MPRA can be a good choice for this purpose.

Chapter 6

Participant Assignment with Varying Arrivals and Varying Release Times

In this section we consider the fourth participant assignment scenario and present our participant assignment algorithms. These algorithms are designed for the scenario where participants' arrival times and experiments' release times vary. We develop algorithms to solve these problems and evaluate the effectiveness of our algorithms on a synthetic benchmark.

6.1 Background

The problem of minimizing weighted tardiness with varying participant arrival times and varying experiment release times is visualized in Figure 6.1. This problem is equivalent to the problem of scheduling jobs on parallel machines where jobs have unequal release times and the number of parallel machines is not fixed (i.e. machines have unequal ready times). Note that we can consider jobs with unequal release times to be participants with varying arrival times, and jobs with unequal ready times as be experiments with varying release times.

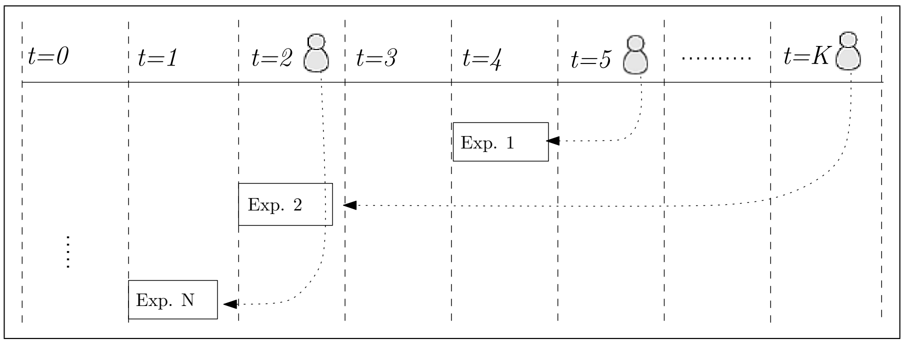


Figure 6.1: Assignment of participants with varying experiment release times and varying arrivals.

Centeno and Armacost [64] describe an offline parallel machine job scheduling approximation algorithm with non-zero job release times and machine eligibility restrictions. They describe a heuristic algorithm for minimizing maximum lateness and evaluate their algorithm using real-world data. Their results with respect to real-world data show that in experiments, completion time decreased from 48 hours to 42 hours and average lot completion time decreased from 85 hours to 19 hours.

These authors describe an offline job scheduling algorithm for minimizing makespan with varying job release times with machine eligilibility constraints [65]. They describe a new heuristic based on assigning jobs to the most restricted machine. They compare the method with other heuristics from the literature such as the heuristic that assigning the jobs with the longest processing time first. Their experimental results show that their heuristic minimizes makespan in a parallel machine environment.

The parallel job scheduling problem with an arbitrary number of machines is strongly NP-hard in [66]. Varying participant arrivals and varying release times makes this problem the hardest studied participant assignment case we consider.

6.2 Algorithms for Optimized Participant Assignment

In this section, we present our algorithms for the scenario of varying participant arrivals with varying release times. We solve this problem using exhaustive enumeration, dynamic programming, greedy algorithms, branch and bound and integer linear programming. In the following sections we introduce details of these algorithms and then give experimental results.

6.2.1 Exhaustive enumeration

The exhaustive algorithm is similar to the exhaustive algorithm designed for the previous scenarios. The difference between the exhaustive algorithm for scenario 4 and the other exhaustive algorithms is the fact that for the fourth scenario, the arrival times of the participants are different and thus the calculation of the total weighted tardiness is different.

Similar to the other scenarios, the recursive exhaustive algorithm takes as input the number of completed experiments, an array of experiments where each experiment contains parameters such as an allocation size to keep track of how many participants are assigned to that experiment, instance size, processing time, due date and priority. Unlike the first scenario and third scenario, we also provide an arrivals array that contains the varying arrival times of the participants to calculate the total weighted tardiness value.

In the given pseudo code below, line 1 ensures that the recursion runs until all experiments are completed. Lines 2-8 traverse the experiment array and recursively allocates each experiment with the current participant as long as we can allocate the experiment. In lines 10-11 for each completed permutation of participant-experiment assignment, the weighted tardiness of the complete assignments is calculated by making use of participants' varying arrival times and experiments' varying release times and checked to see if it is the minimum weighted tardiness that has been found so far.

The base case occurs when all the experiments are completed. Similar to the other exhaustive algorithms, if K is the number of assigned participants, the overall function runs within a polynomial factor of O(K!) in the worst case, the factorial of the number of total assigned participants, assuming every experiment requires at least one participant to start.

6.2.2 Dynamic Programming Algorithm

Since the recursive algorithm for the scenario of varying participant arrivals and varying experiment release times is similar to the recursive algorithm of the previous scenarios, the recursion tree and the exact solution approach with dynamic programming to solve the problem is also similar.

Structure of the Optimal Solution:

The definitions of the sorted sequence of experiment numbers S, optimal solution G, and subsequence $S - E$ where E is a given sequence of experiment numbers are the same with the first three scenarios. In the same way, the structure of an optimal solution is unchanged, as described in 3.3.2, 4.2.2 and 5.2.2. The cost of an overall solution is the first participant's weighted tardiness

Algorithm 31 Exhaustive

```
Require: count of started experiments, experiments array, arrivals array
 1: if count of started experiments < number of $experiments$ then
 2:     for every experiment $i$ do
 3:         Assign the next participant to the any empty slot of experiment $i$
 4:         if Assignment process results in the experiment to start then
 5:             Increment the count of started experiments
 6:         end if
 7:         $Exhaustive$(count of started experiments, $experiments$, $arrivals$)
 8:     end for
 9: else
10:     Calculate the tardiness value of the found assignment order by considering varying arrival times of the
        participants and varying release times of the experiments
11:     Store the minimum found tardiness value
12: end if
```

value plus the optimized total weighted tardiness value of the sub problem where the various release times of the experiments are also taken into consideration.

Recursive Definition of the Optimal Solution: The recursive solution of the problem is also the similar to the recursive solutions of the first three scenarios. Given S, experiment number i, experiment's release time array rel and experiment release order number k; $G(i, S, rel(i) + A(t))$ is the total minimum weighted tardiness assuming that the participant arriving at time $rel(i) + A(t)$ is assigned to experiment i and the rest of participants are assigned to experiment numbers from S. Then we can recursively define the optimal solution as:

$$G(i,S,rel(i)+t) = \begin{cases} WT_i(rel(i)+A(t)) & \text{if } S \text{ is empty} \\ min_j\{G(j,S-(j),rel(j)+A(t+1))\} & \text{if } i \text{ in } S \\ WT_i(rel(i)+A(t)) + min_j\{G(j,S-(j),rel(j)+A(t+1))\} & \text{if } i \text{ not in } S \text{ and } S \text{ is not empty} \end{cases} \tag{6.1}$$

Similar to 3.3.2 and 4.2.2, the above statement states that there are three cases to be considered to determine the total minimum weighted tardiness. Unlike 3.3.2, 4.2.2 and similar to 5.2.2, instead of considering time sequentially we assume participants are arriving at varying times by time and for this reason we are keeping an arrivals array A. In this scenario we also assume that participant release times differ and we keep a rel array to keep varying release times of the experiments.

Computing the Value of the Optimal Solution Bottom Up:

Similar to the Algorithm 2, 9 and 9, the optimal solution for varying arrivals and uniform release times are computed bottom up by making use of experiments and S sequence. Unlike

Algorithm 2and 15 and similar to 9, we are providing an arrivals array A to keep varying arrival times of the participants. Unlike Algorithm 2and Algorithm 9 and similar to Algorithm 15, we are providing an experiment release times array rel in the pseudo code to compute the total weighted tardiness value of the experiments with varying release dates. The total tardiness value in the algorithm is calculated by making use of the varying arrival times of the participants and varying release time values of the experiments. Pseudo code for the dynamic programming solution to the problem of minimizing total weighted tardiness for uniform arrivals of the participants and varying release times experiments is provided as Algorithm 32 below. Given the experiment numbers and arrival array, our goal is to find

$$min\{G(i, S, rel(i) + A(0))\}\ \forall \text{ experiment number } i$$

In Algorithm 32, lines 4, 10 and 12 differ from the Algorithm 2 and 9 in the sense that these lines compute the weighted tardiness value by making use of the rel release times array and differ from the Algorithm 2 and 15 in the sense that it also makes use of varying arrival times of participants by making use of A array.

Algorithm 32 Dynamic

```
Require: experiments array, S sequence, rel, A
 1: Initialize $K$ as the number of participants needed to make all experiments start
 2: Initialize $G$ as empty to hold weighted tardiness values
 3: for every experiment $i$ do
 4:     Set $G(i, \emptyset, rel(i) + A(t))$ = weighted tardiness of experiment $i$ assuming it finishes at time $A(t)$ for $t = K - 1$
 5: end for
 6: for every subsequence of $S$ do
 7:     for every experiment $j$ from $S$ do
 8:         Set $t = K - 1-$ size of subsequence
 9:         if $j$ is in subsequence then
10:             $G(j, subsequence, rel(j) + A(t)) = min\{G(k, subsequence - (k), rel(k) + A(t+1))\}\ \forall\ k$ in subsequence
11:         else
12:             $G(j, subsequence, rel(j) + A(t))$ =weighted tardiness of $j + min\{G(k, subsequence - (k), rel(k) + A(t+1))\}\ \forall\ k$ in subsequence
13:         end if
14:     end for
15: end for
```

Constructing Optimal Solution from Computed Information:

Similar to the optimal solution construction approach of the first three scenarios' dynamic programming algorithms, we find the optimal total weighted tardiness value by making use of the tuples. Having identified $(1, i)$ at time $t = rel(i) + A(0)$, we can backtrack to find our optimal solution by finding the experiment j for each tuple that minimizes the expression.

Running Time and Space Requirements:

The "Dynamic" algorithm that is used for computing the optimal solution has the same runtime complexity as the dynamic programming algorithm for the first three scenarios and it is a $O(K \times 2^K)$ time operation where K is the summation of experiments' instance sizes. The procedure that is used to construct the optimal solution takes $O(K^3)$ time and thus the overall procedure has a running time of $O(K^2 \times 2^K)$. Similar to the scenario 1, scenario 2 and scenario 3, the total space requirement is also $O(K \times 2^K)$.

6.2.3 Offline Greedy Algorithms

In scenario 3, in most of the offline greedy algorithms, we treated the current time (ie. participant arrival time) as an experiment's release date. For this reason, even if participants are arriving at varying times for scenario 4, the working principle of offline greedy algorithms does not change. Therefore, unlike scenario 3, for the case of varying arrivals of participants and varying release time of experiments, we will only explain online greedy algorithms in this section.

6.2.4 Online Greedy Algorithms

Similar to scenario 3 online greedy algorithms, we consider seven algorithms: minimum slack (MS), minimum slack with priority (MSP), next fit (NF), smallest critical ratio (SCR), smallest critical ratio with priority (SCRP), smallest instance size with remaining time (SIRT) and maximum priority with remaining attributes (MPRA).

6.2.4.1 Minimum Slack

As defined for scenario, slack time is the amount of time between an experiment's due date and the current time minus the experiment's processing time. Unlike scenario 3, participants do not have to arrive at sequential times. For this reason we use an arrivals array for the varying participant arrival times. Similar to scenario 3, we call the online version of slack $real\,slack$. In Algorithm 33, we give the pseudo code for the online minimum slack algorithm.

The algorithm uses an experiments array to calculate minimum slack values of the experiments. Lines 1 through 7 calculate real slack by making use of arrivals array for every experiment in the $experiments$ array and assign participants to the experiment with minimum real slack at every time. Finally, we calculate total minimum weighted tardiness in line 8 using the $arrivals$ array.

Algorithm 33 Minimum Slack

```
Require: experiments array, arrivals array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\ slack$ by making use of $arrivals$
4:         Find the experiment with minimum $real\ slack$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness by making use of $arrivals$
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

6.2.4.2 Minimum Slack with Priority

Similar to scenario 3, we divide an experiment's real slack by its priority and assign participants to the experiment that gives a smallest value. Unlike scenario 3, we now also consider the varying arrivals of the participants. Pseduo code for this approach is given in Algorithm 34.

The algorithm uses an experiments array to calculate experiments' minimum slack values with priority. Lines 1 through 7 are used to calculate the $real\ slack \div$ experiment's priority for every experiment in the $experiments$ array and assign participants to the experiment with minimum value at every time. Finally, we calculate total minimum weighted tardiness in line 8 by using the $arrivals$ array.

Similar to the online minimum slack algorithm, for N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

6.2.4.3 NextFit

Similar to scenario 3 online version of the nextfit algorithm, we consider the time when the experiment is ready to launch to calculate the weighted tardiness. Unlike the previous online

Algorithm 34 Minimum Slack

```
Require: experiments array, arrivals array
for every experiment do
    for every unconsidered experiment do
        Calculate $real\ slack$
        Find the experiment with minimum $real\ slack \div$ experiment's priority
    end for
    Assign participants to the experiment
end for
Calculate the total minimum weighted tardiness by making use of $arrivals$
```

nextfit algorithm, this time is now affected by the varying arrivals of participants. Pseudo code for this approach is given in Algorithm 35.

In the pseudo code the algorithm uses a global current experiment number to keep track of the current experiment to which the participant has assigned, an experiments array and a participants array to assign each incoming participant to a suitable experiment. Lines 1 through 7 are responsible for assigning incoming participant to the next suitable experiment. Finally, line 8 calculates the total minimum weighted tardiness by making use of $arrivals$.

Algorithm 35 NextFit

```
Require: experiments array, participants array, current experiment number, arrivals array
1: for every participant $i$ do
2:     if current experiment didn't start then
3:         Assign participant $i$ to the current experiment
4:         Set current experiment to be the next experiment
5:     end if
6: end for
7: Calculate the total minimum weighted tardiness by making use of $arrivals$
```

For K incoming participants, the run time complexity of the algorithm is $O(K)$ and the total space requirement is $O(1)$.

6.2.4.4 Smallest Critical Ratio

Similar to scenario 3 online version of the smallest critical ratio, the critical ratio is the time remaining $\div$ work remaining, where time remaining is the difference between experiment's due date and the current time. Unlike the scenario 3, current time is now the current participant's arrival time from $arrivals$. Work remaining is the remaining processing time. In Algorithm 36 we gave the pseudo code for the online version of the smallest critical ratio algorithm. We called the online version of the critical ratio as $real\,critical\,ratio$ in Algorithm 36.

Algorithm 36 uses an experiments array to calculate the $real\,critical\,ratio$ values for each experiment and an arrivals array to calculate the total minimum weighted tardiness.

Lines 1 through 7 calculate the real critical ratio for every experiment in the $experiments$ array and assign participants to the experiment with minimum real critical ratio at every time. Finally we calculate total minimum weighted tardiness in line 8 by making use of $arrivals$.

Algorithm 36 Smallest Critical Ratio

```
Require: experiments array, arrivals array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\,critical\,ratio$
4:         Find the experiment with minimum $real\,critical\,ratio$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ due to two for-loops, and the total space requirement is $O(1)$.

6.2.4.5 Smallest Critical Ratio with Priority

Similar to scenario 3, this algorithm calculates experiment's critical ratio divided by experiment's priority. Unlike the offline smallest critical ratio. However, varying arrivals of participants are also considered for this algorithm. The detailed pseudo code of this approach is given in Algorithm 37.

Algorithm 37 uses an experiments array and participants' arrivals array to calculate real critical ratio $\times \frac{1}{priority}$ values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8 by making use of $arrivals$.

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

Algorithm 37 Smallest Critical Ratio With Priority

Require: experiments array, arrivals array

```
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate $real\,critical\,ratio \times \frac{1}{priority}$
4:         Find the experiment with minimum $real\,critical\,ratio \times \frac{1}{priority}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness by making use of $arrivals$
```

6.2.4.6 Smallest Instance Size with Remaining Time

Similar to scenario 3, we consider the instance sizes and remaining time of experiments by multiplying them and selecting the experiment with the minimum resulting value. The pseudo code of this approach is given in Algorithm 38.

Algorithm 38 requires an experiments array to calculate smallest instance size with remaining time value by making use of an experiment's instance size $\times$ (experiment's due date $-$ current time) by making use of $arrivals$ while considering current time. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally we calculate total minimum weighted tardiness in line 8 by making use of $arrivals$.

Algorithm 38 Smallest Instance Size with Remaining Time

Require: experiments array, arrivals array

```
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's instance size $\times$ (experiment's due date $-$ current time) by making use of
           $arrivals$
4:         Find the experiment with minimum experiment's instance size $\times$ (experiment's due date $-$ current
           time)
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness by making use of $arrivals$
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

6.2.4.7 Maximum Priority with Remaining Attributes

Similar to scenario 3, we consider all of an experiment's attributes in the online version of the algorithm. Unlike the previous version of the algorithm, we consider $arrivals$ array for arrival times of participants in the algorithm. Thus the calculation of maximum priority with remaining attributes turns into $priority \div ((due\,date - current\,time) \times processing\,time \times remaining\,instance\,size)$ Unlike the scenario 3, in the formulation current time does not increase sequentially and it is equal to the current participant's arrival time from $arrivals$. Pseudo code for this approach is given in Algorithm 39.

Algorithm 39 uses an experiments array and arrivals array to calculate the Maximum Priority with Remaining Attributes values of the experiments. Lines 1 through 7 calculate this value for every experiment in the $experiments$ array and assign participants to the experiment with the minimum value at every time. Finally, we calculate the total minimum weighted tardiness in line 8 by making use of $arrivals$.

Algorithm 39 Maximum Priority with Remaining Attributes

```
Require: experiments array, arrivals array
1: for every experiment do
2:     for every unconsidered experiment do
3:         Calculate experiment's $\frac{priority}{(due\,date - current\,time) \times processing\,time \times remaining\,instance\,size}$
4:         Find the experiment with maximum $\frac{priority}{(due\,date - current\,time) \times processing\,time \times remaining\,instance\,size}$
5:     end for
6:     Assign participants to the experiment
7: end for
8: Calculate the total minimum weighted tardiness by making use of $arrivals$
```

For N experiments, the run time complexity of the algorithm is $O(N^2)$ and the total space requirement is $O(1)$.

6.2.5 Branch and Bound Solution

Similar to scenario 3, to apply branch and bound to the participant assignment problem, we consider more than one state-space tree because of the experiments' varying release times. Unlike scenario 3, in the state-space tree, participant arrival times are not sequential and varying arrival times of the participants needed to be considered.

Pseudo code for the branch and bound algorithm is given in Algorithm 40. The algorithm considers all possible experiment-participant arrival time tables. In those tables, participant arrival times are different and loaded from the arrivals array. For each row of that experiment-participant arrival time table, the algorithm finds finds the participant arrival time that gives the smallest total weighted tardiness and continues to expand the state-space tree in this way. The algorithm finds the minimum total weighted tardiness value and the corresponding participant arrival order can be easily found from the state-space tree.

Algorithm 40 Branch and Bound

```
Require: experiments array, arrivals array
 1: Find all possible experiment-participant arrival time tables by making use of $arrivals$
 2: for every experiment-participant arrival time table do
 3:     for every experiment $i$ do
 4:         for every participant arrival time $t$ do
 5:             Assign the participant coming at time $t$ to the experiment $i$
 6:             For the remaining experiments consider the remaining experiment arrival times which result in
                the smallest possible total weighted tardiness value
 7:             if Assigned participant to the experiment $i$ at time $t$ results in smallest total weighted tardiness
                value with respect to other $t$ values for that experiment then
 8:                 Accept that participant coming at time $t$ is assigned to experiment $i$
 9:             end if
10:         end for
11:     end for
12: end for
```

For N experiments and K participants, the run time complexity of the algorithm is $O(N \times 2^K)$. The total space requirement of the algorithm is also $O(N \times 2^K)$.

6.2.6 Integer-Linear Programming Solution

Similar to scenarios 1, 2 and 3, we solved the assigment problem with integer-linear programming for scenario 4. Unlike the other scenarios, the solution now considers the varying release times of the experiments and varying participant arrival times at the same time. For scenario 4, we use a modified definition of $Tardiness_{i,t}$ in the formulation of the objective function. Note that instead of summing processing time (PT_i) with the time t that makes an experiment start as in the first scenario or summing the processing time (PT_i) with the maximum of arrival time t that makes an experiment start or experiment's release time as in the third scenario, now we sum the processing time with the maximum of the time $arrivals(i) + t$ that makes an experiment start and the experiment's release time since the experiment may start right after its release time if there are existing participants, or it may start at a later time $arrivals(i) + t$ after the release date. This is shown as $max(REL_i, arrivals(i) + t)$ in the below formulation.

$$Tardiness_i(t) = \begin{cases} 0 & if\ max(REL_i, arrivals(i) + t) + PT_i \leq DD_i \\ max(REL_i, arrivals(i) + t) + PT_i - DD_i & Otherwise \end{cases} \tag{6.2}$$

Other than the modified version of the objective function, our integer-linear programming formulation for varying arrivals of participant and varying release times of the experiments is the same as formulations for the other scenarios.

6.3 Experimental Results

In this section we report our experimental results for the algorithms we described in Section 6.2. Similar to 5.3, we report and compare the performance of the exhaustive enumeration, dynamic programming, greedy, branch and bound and integer-linear programming solutions. We then give the size of the largest solved instance for each algorithm, and investigate how close the solutions they provide are to optimal.

Similar to 3.5, 4.3 and 5.3 the first set of experiments investigate the percentage of the benchmark that the algorithms can process within a fixed amount of time. The execution time of the algorithms is limited to either 1 minute or 10 minutes. Results are reported in Table 6.1 for the 1-minute time restriction and in Table 6.2 for the 10-minute time restriction. Since online and offline greedy algorithms show the same execution performance, they are considered together in the following tables.

Table 6.1: The number and percentage of the instances that are processed within one minute.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	2	4%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
MS, MSP, NF, SCR SCRP, SIRT, MPRA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
B&B	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	3	6%	0	0%
	15	26	52%	4	8%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

Table 6.2: The number and percentage of the instances that are processed within ten minutes.

	N	$1 \leq ES \leq 3$		$4 \leq ES \leq 6$		$7 \leq ES \leq 9$	
		Processed	**Percentage**	**Processed**	**Percentage**	**Processed**	**Percentage**
Exhaustive	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
DP	5	50	100%	0	0%	0	0%
	10	5	10%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
MS, MSP, NF, SCR SCRP, SIRT, MPRA	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	50	100%
	15	50	100%	50	100%	50	100%
	50	50	100%	50	100%	50	100%
	100	50	100%	50	100%	50	100%
	200	50	100%	50	100%	50	100%
	500	50	100%	50	100%	50	100%
	1000	50	100%	50	100%	50	100%
	2000	50	100%	50	100%	50	100%
	5000	50	100%	50	100%	50	100%
B&B	5	5	10%	0	0%	0	0%
	10	0	0%	0	0%	0	0%
	15	0	0%	0	0%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%
ILP	5	50	100%	50	100%	50	100%
	10	50	100%	50	100%	2	4%
	15	50	100%	21	42%	0	0%
	50	0	0%	0	0%	0	0%
	100	0	0%	0	0%	0	0%
	200	0	0%	0	0%	0	0%
	500	0	0%	0	0%	0	0%
	1000	0	0%	0	0%	0	0%
	2000	0	0%	0	0%	0	0%
	5000	0	0%	0	0%	0	0%

As shown in Tables 6.1 and 6.2, when the experiment size (ES) or the number of experiments (N) increase, the total number of participants needed (K) for each benchmark instance increases and the number of solved instances within each segment decreases. The worst performance is observed using the exhaustive algorithm. The second-worst results are observed with the dynamic programming and branch and bound algorithms. Since the runtimes of these algorithms are exponential with respect to the number of experiments, these are the expected results. ILP performs better than exhaustive algorithm, dynamic programming and branch and bound for both 1-minute and 10-minute tests, yet it is slower than the greedy algorithms. Based on the results from both tables, the greedy algorithms are the fastest algorithms.

The second set of experiments is conducted to observe how close the solutions found by the algorithms are to optimal. The solutions provided by the dynamic programming algorithm are used as a baseline. The instances for this test consist of 29 instances from the benchmark segment with $N = 15$ and $4 \leq ES \leq 6$. We then applied the other algorithms to this benchmark segment for 1 minute and 10 minutes for every instance. The results from the comparison of the exhaustive, branch and bound, integer-linear programming and online and offline greedy algorithms are provided in Figure 6.2, Figure 6.3, Figure 6.4.

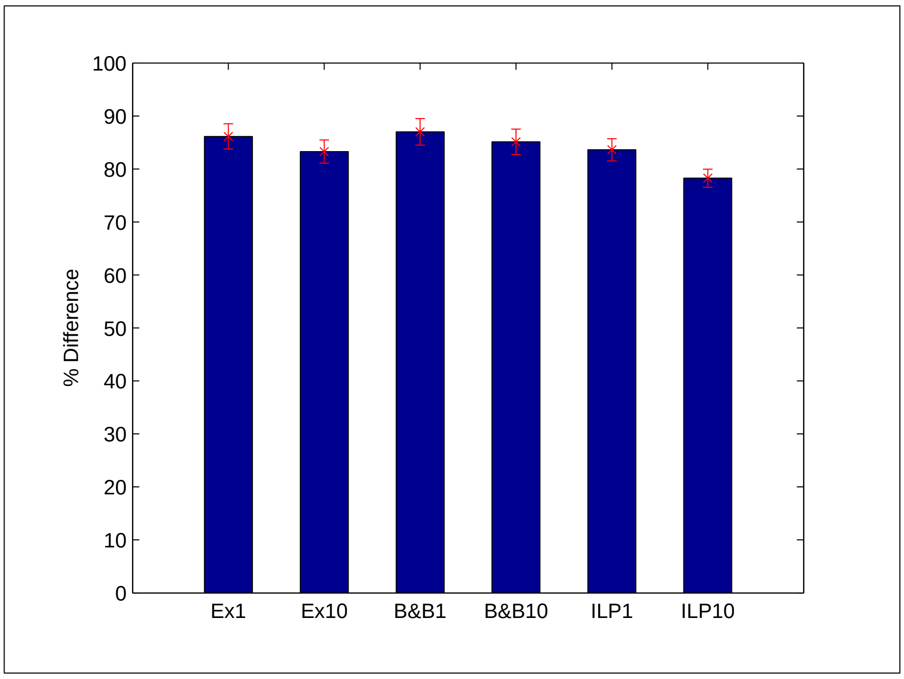


Figure 6.2: Percentage difference between exact algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. Ex1 and Ex10 stands for 1-minute and 10-minutes execution results of the exhaustive algorithm. B&B1, B&B10 and ILP1, ILP10 stand for 1-minute and 10-minute execution results of the branch and bound and ILP algorithms respectively.

As can be seen from Figure 6.2, when the execution time increases from 1 minute to 10 minutes, all of the compared algorithms find more accurate results. The best improvement is seen in the ILP results.

The intuition behind the optimality performance of exact algorithms is that although these algorithms are slow since they consider the all or partial solutions to find the exact results. This makes the solution time of the algorithms to be very slow and the optimal results couldn't be found within 1 minutes or 10 minutes of time.

We conducted the two-sample t-test to determine whether these results are statistically significant. In Table 6.3 we provide the significance results (p-values) for this test. The results show that there is a significant difference between B&B and ILP 10 minutes results.

Table 6.3: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	$EX1$	$EX10$	$B\&B1$	$B\&B10$	$ILP1$	$ILP10$
$EX1$	1.00	0.48	0.38	0.82	0.64	0.29
$EX10$	-	1.00	0.01	0.03	0.57	0.81
$B\&B1$	-	-	1.00	0.07	0.01	0.01
$B\&B10$	-	-	-	1.00	0.37	0.03
$ILP1$	-	-	-	-	1.00	0.11
$ILP10$	-	-	-	-	-	1.00

We also conduct the optimality tests for online and offline greedy algorithms. Our optimality comparison results of these algorithms for 1 minute and 10 minutes tests are provided in Figures 6.3 and 6.4.

According to the offline greedy algorithm comparison results shown in Figure 6.3, MPRA gives the best performance and NF gives the worst performance. Similar to scenario 3, these results were expected since NF was the only algorithm which does not make use of experiment release times and MPRA considers the experiment release times and all other experimental attributes appropriately. MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms find solution that are on the average 0.45%, 0.45%, 0.92%, 0.54%, 0.53%, 0.60% and 0.41%, respectively, from optimal for both 1-minute and 10 minutes tests.

In Table 6.4 we report the two sample t-test significance results for the optimality results of the offline greedy algorithms. The results show that in general algorithms' results are significanly different from each other.

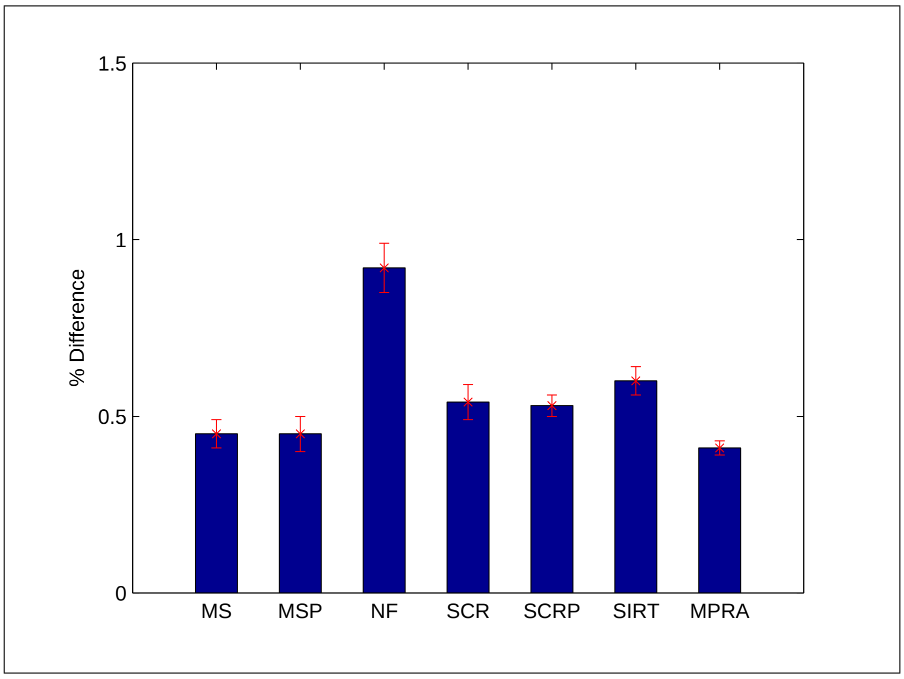


Figure 6.3: Percentage difference between offline greedy algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. MS stands for 1-minute and 10-minutes execution results of the minimum slack since they are the same value. MSP, NF, SCR, SCRP, SIRT, MPRA stand for 1-minute and 10-minute execution results of the offline greedy algorithms respectively.

Table 6.4: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	MS	MSP	$MPRA$	NF	$SIRT$	SCR	$SCRP$
MS	1.00	0.65	0.01	0.01	0.01	0.01	0.37
MSP	-	1.00	0.01	0.01	0.01	0.01	0.36
$MPRA$	-	-	1.00	0.01	0.01	0.01	0.01
NF	-	-	-	1.00	0.01	0.01	0.01
$SIRT$	-	-	-	-	1.00	0.01	0.01
SCR	-	-	-	-	-	1.00	0.06
$SCRP$	-	-	-	-	-	-	1.00

According to the online greedy algorithm comparison results shown in Figure 6.4, MPRA again performs best and the NF algorithm performs worst. MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms find the same solution for 1-minute and 10 minutes tests and they are on the average 0.56%, 0.55%, 0.99%, 0.59%, 0.58%, 0.65% and 0.49% from optimal, respectively.

The intuition behind these results is that in the case of online algorithms, since we do not

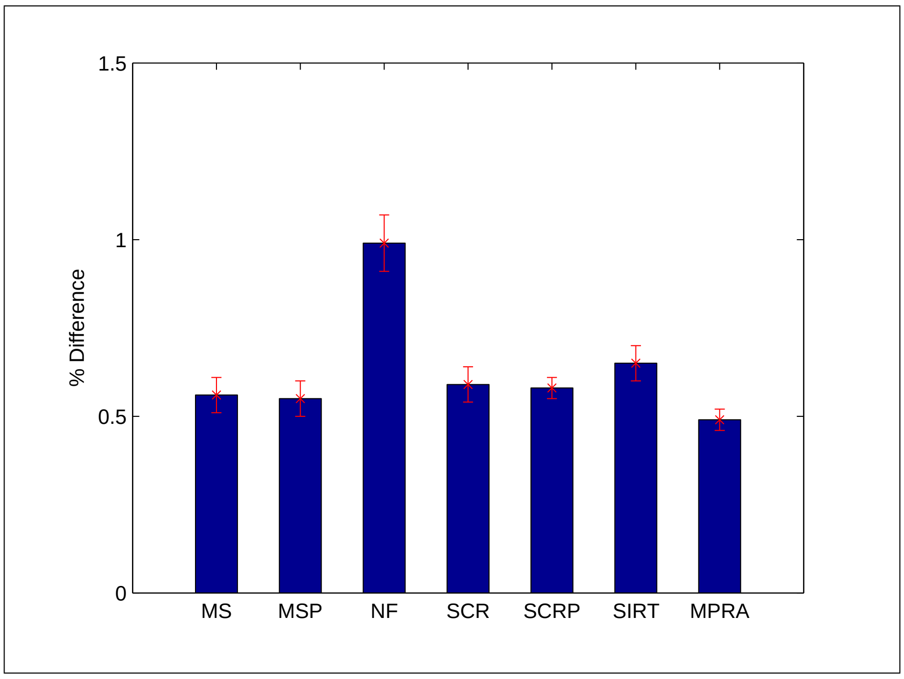


Figure 6.4: Percentage difference between online greedy algorithms and dynamic programming after running its instances with 1-minute and 10-minute time restrictions. MS stands for 1-minute and 10-minutes execution results of the minimum slack since they are the same value. MSP, NF, SCR, SCRP, SIRT, MPRA stand for 1-minute and 10-minute execution results of the offline greedy algorithms respectively.

know the participant arrival sequence and experiment release times, assigning participants to experiments is a more difficult task and this affects the optimality performance results of the algorithms. NF performs worse than the other algorithms because it is not using experiment properties such as instance sizes, due dates, processing times in its formulation. MPRA is the best performing one because it makes use of all the key experiment properties appropriately while finding the assignment order.

Note that similar to scenario 3, the solutions found by the online greedy algorithms are slightly worse than the solutions found by the offline algorithms, which was expected since in the case of online algorithms the number of incoming participants and experiment release times are unknown prior to participant assignment.

We report the two sample t-test significance results for the online greedy heuristics in In Table 6.5. The results show that in general algorithms are significantly different from each other.

Table 6.5: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	MS	MSP	$MPRA$	NF	$SIRT$	SCR	$SCRP$
MS	1.00	0.58	0.01	0.01	0.01	0.01	0.34
MSP	-	1.00	0.01	0.01	0.01	0.01	0.32
$MPRA$	-	-	1.00	0.01	0.01	0.01	0.01
NF	-	-	-	1.00	0.01	0.01	0.01
$SIRT$	-	-	-	-	1.00	0.01	0.01
SCR	-	-	-	-	-	1.00	0.05
$SCRP$	-	-	-	-	-	-	1.00

For harder instances where we could not find exact solutions within 10 minutes, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. Table 6.6 shows the average difference between the LP lower bound and optimal ILP solutions for small instances where optimal results were obtained using ILP within 10 minutes.

The average difference between the LP lower bound and the optimal results is 6.5%. In the same way, the average difference between the LP lower bound and the solutions computed by the MS, MSP, NF, SCR, SCRP, SIRT, MPRA algorithms is found to be 0.5%, 0.8%, 0.6%, 0.6%, 0.7% and 0.2% respectively.

Table 6.6: Comparison of LP and ILP lower bounds for small instances as well as LP and offline greedy soluntions on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. ILP	5.6%	6.7%	7.1%
LP vs. MS	0.4%	0.5%	0.6%
LP vs. MSP	0.4%	0.5%	0.6%
LP vs. NF	0.7%	0.8%	0.9%
LP vs. SCR	0.5%	0.5%	0.7%
LP vs. SCRP	0.4%	0.6%	0.7%
LP vs. SIRT	0.7%	0.8%	0.7%
LP vs. MPRA	0.2%	0.2%	0.3%

The above results indicate that offline MPRA results are closest to the LP results. Table 6.7 shows the average difference between the LP lower bound and the solutions computed by the greedy algorithm based on MS for larger instances. The average difference between the LP lower bound and the solutions computed by the online MS algorithm is 0.5%. In the same way, the average difference between the LP lower bound and the solutions computed by the online MSP, NF, SCR, SCRP, SIRT and MPRA algorithms is found to be 0.6%, 0.9%, 0.6%, 0.5%, 0.7% and 0.3% respectively.

Table 6.7: Comparison of LP lower bounds and online greedy solutions on the average total weighted tardiness for large instances.

	$1 \leq ES \leq 3$	$4 \leq ES \leq 6$	$7 \leq ES \leq 9$
N	$Difference$	$Difference$	$Difference$
LP vs. MS	0.5%	0.6%	0.5%
LP vs. MSP	0.5%	0.6%	0.6%
LP vs NF	0.7%	0.9%	1.0%
LP vs. SCR	0.5%	0.6%	0.7%
LP vs. SCRP	0.5%	0.5%	0.6%
LP vs. SIRT	0.6%	0.8%	0.7%
LP vs. MPRA	0.3%	0.4%	0.3%

Above results show us that on the average the MPRA heuristic perform better than the other greedy heuristics. Next, we compare our online and offline version of greedy algorithms.

Table 6.8: Comparison of online and offline greedy algorithms.

$Algorithm$	$Difference$
MS	5.2%
MSP	0.2%
NF	12.4%
SCR	7.1%
SCRP	6.5%
SIRT	8.8%
MPRA	3.6%

Similar to scenario 3, the above results show that the percentage of difference between online and offline algorithms which is calculated by dividing the difference between the results obtained by the offline and online version of an algorithm to the results obtained by the online version are not very significant. The largest difference is seen with nextfit. The results show us that for MSP the difference is relatively small which means this algorithm has more stable performance with respect to other algorithms for both online and offline cases. We conclude that, in general, the MPRA greedy heuristic is preferable to other greedy heuristics and provides significantly better results over many benchmark instances for both online and offline cases in terms of optimality and execution performance. Overall, similar to scenario 3, these results imply that the greedy algorithms should be chosen to solve assignment problems based on varying participant arrivals and varying experiment release times and MPRA can be a good choice for this purpose.

6.3.1 The Robustness of Online Algorithms

To analyze the robustness of the proposed algorithms, we focus on online algorithms for scenario 4 because it is the only case where we developed online algorithms and participants are arriving at varying times.

We evaluated online algorithms over nonbursty (our original model), weak bursty, and strong bursty environments. In each case, the average inter-arrival time was 1 second. The details of the weak bursty and strong bursty environments are provided in Figures 6.5 and 6.6. Since the scenario we considered throughout the thesis is an example of a nonbursty environment, the details of nonbursty environment are not provided. Each arrival process is drawn from a 2-state Markovian-Modulated Poisson Process [67] and is used to generate the different bursty flows.

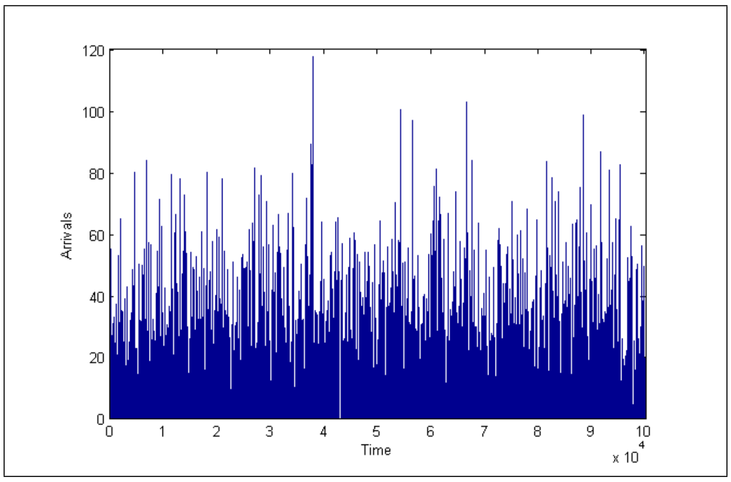


Figure 6.5: Weak bursty environment.

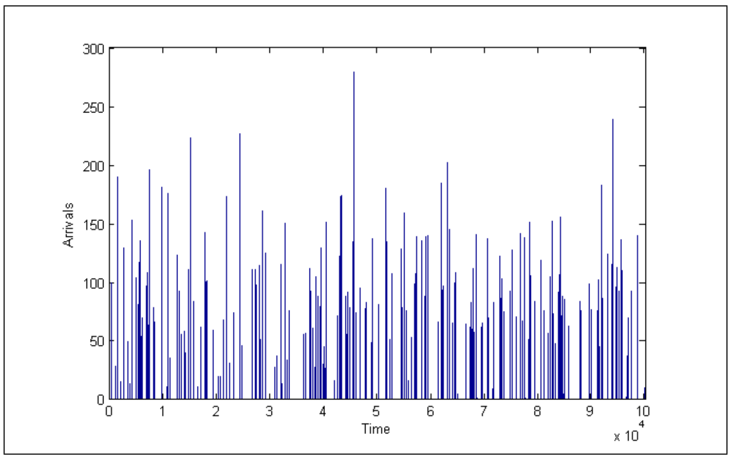


Figure 6.6: Strong bursty environment.

We tested the effectiveness of the online algorithms over the burstiness dataset. First, optimal total weighted tardiness values for the nonbursty, weak bursty and strong bursty data are found by ILP and then online greedy algorithms are run over the same data. Finally optimality is calculated as (online greedy solution - ILP solution) $\div$ online greedy solution $\times$ 100. The burstiness dataset contains more than 100000 participants and this number is enough to make all experiments start. The results show that MPRA gives better results with respect to other algorithms in non bursty, weak bursty and strong bursty environments. The details of the observed results are given in Figure 6.7.

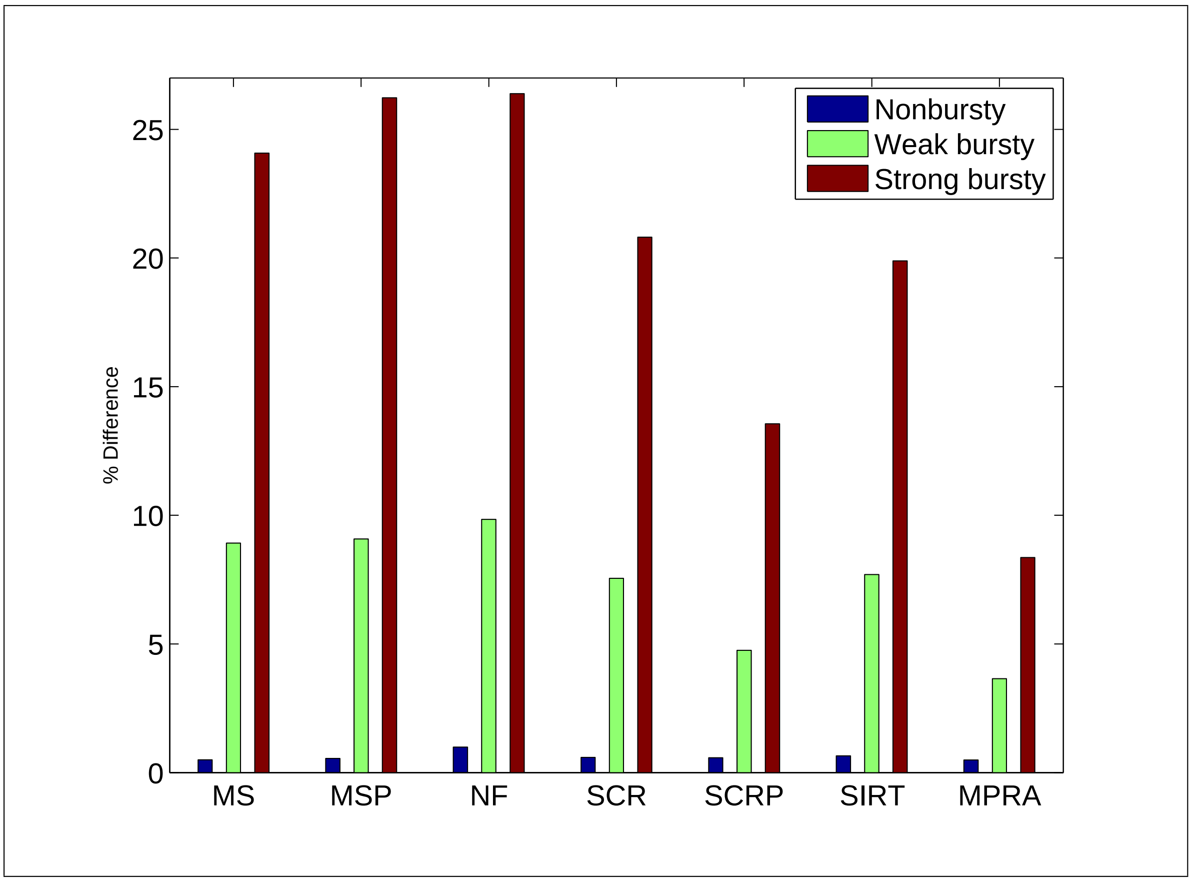


Figure 6.7: Percentage difference between online greedy algorithms and ILP results with different burstiness after running its instances 10 minutes. Different colors indicate results for datasets with different burstiness. Blue is for nonbursty dataset, green is for weak bursty dataset and red is for strong bursty dataset.

Based on the results described above, MPRA is the best performing algorithm. This is a result of the fact that the algorithm makes appropriate use of all the experimental parameters. However, the algorithms' results are affected by varying levels of burstiness because it is not designed to handle burstiness (ie. it is not burstiness aware). Note that none of the compared algorithms display robustness in the presence of burstiness. MPRA is affected by the burstiness to approximately the same degree as the other algorithms and the better results in terms of optimality is based on the participant arrival rate. In a weak bursty environment, participants are generally assigned over time, whereas in a strong bursty environment nearly all participants are assigned at once. Therefore in weak bursty environment continuous arrival of participants make the experiments start within a short time after the experiments are released. However, in a strong bursty environment, there may be time periods with no arrivals of participants. This can cause the experiments' due dates to be passed without an assignment, resulting in higher weighted tardiness.

6.3.2 Parameters That Have the Biggest Effect on Total Weighted Tardiness

In this section, we describe the results of evaluating the effects of varying input parameters on overall total weighted tardiness. Those parameters include arrival rate, release times and the number of experiments N over the online algorithms, for scenario 4.

To test the effect of arrival rate on total weighted tardiness, we conducted two different experiments. In both experiments we kept the number of experiments $N = 100$ to be a constant value and assumed all experiments are released at the beginning. In one of the experiments we assumed participants are arriving sequentially as in the first and third scenario and in the second experiment we assumed participants are arriving at varying times as in the second and fourth scenario. Later we used ILP to calculate optimal total weighted tardiness values for these two settings and report the difference percentage as $100 \times$ ((considered online algorithm's result - ILP result) $\div$ considered online algorithm's result). Finally we compare the optimality results of the experiment with sequential arrivals to the experiment with varying arrivals. The results are displayed in Figure 6.8.

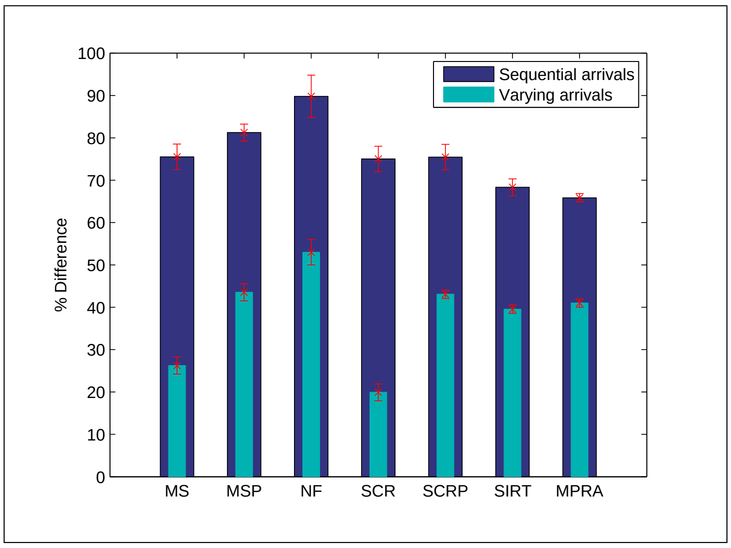


Figure 6.8: Percentage difference between online greedy algorithms and ILP results for varying vs. sequential arrivals after running its instances 10 minutes

According to the above figure, it is seen that there is a significant difference between the optimal results found by ILP and the results found by sequential arrivals and varying arrivals. With these settings for both sequential arrivals and varying arrivals, each experiment starts at different times when the last required participant is fulfilled and this results in a difference between the optimal results and the results found by the online algorithms which are not optimal.

In the same way, to test the effect of varying release times over the total weighted tardiness, we conducted two different experiments. In both experiments we kept the number of experiments $N = 100$ to be a constant value and assumed all participants are arriving sequentially. In one of the experiments we assumed experiments are released at the beginning as in the first and second scenario and in the second experiment we assumed released times are varying as in the third and fourth scenario. We present the difference between these two experiments in Figure 6.9.

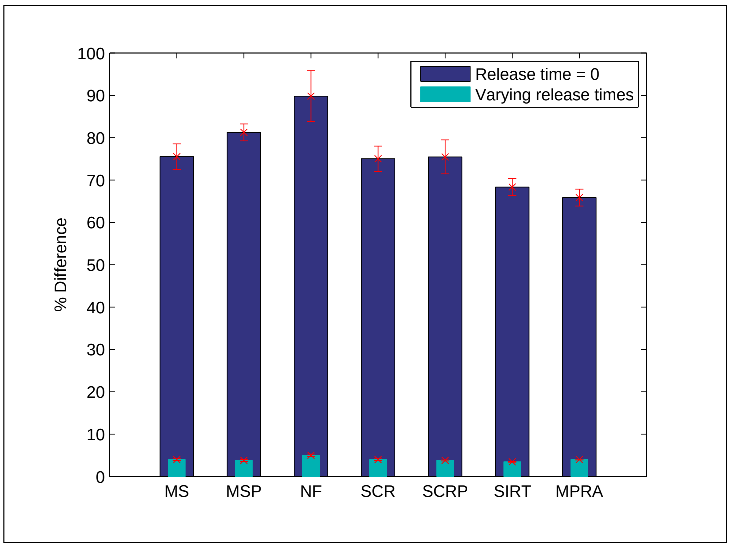


Figure 6.9: Percentage difference between online greedy algorithms and ILP results for release times = 0 vs. varying release times after running its instances 10 minutes.

According to the above figure, there is a big difference between the results found by the first experiment and the second experiment, and varying release times gives much better results. This is because the participants who are waiting until the experiments are released are assigned at once and thus the experiments may start immediately after their release. This results in the total weighted tardiness values found by the online algorithms to be very close to the results found by ILP. In contrast, when we consider experiments that are released at the beginning, experiments start at different times when they are fulfilled with participants and this results in a difference between the optimal results and the results found by the online algorithms which are not optimal.

Finally we considered the impact of experiment count N on total weighted tardiness. Similar to the others we conducted two different experiments. In both experiments we assumed all participants arrive sequentially and experiments are released at the beginning. Then we changed the number of experiments to be $N = 10$ in one experiment and $N = 100$ in the other experiment. We present the difference between these two experiments in Figure 6.10

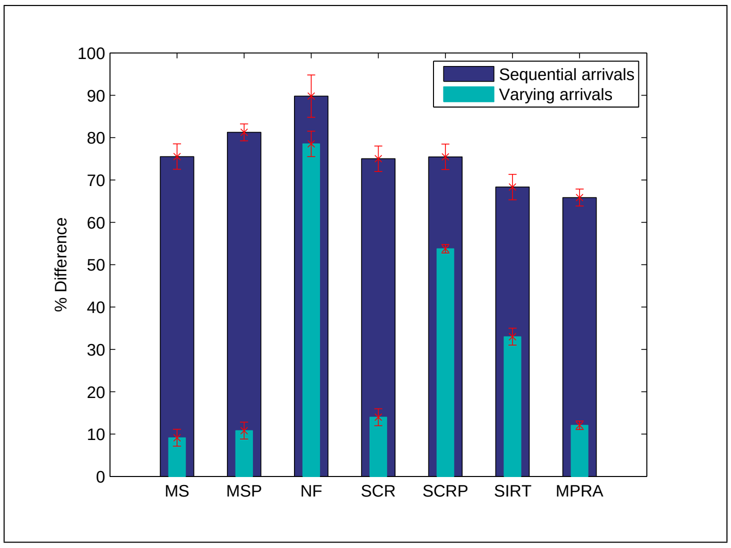


Figure 6.10: Percentage difference between online greedy algorithms and ILP results for varying N values after running its instances 10 minutes.

According to the above figure, when we increase N, the difference between the ILP and online algorithms increases. This result was expected since finding the optimal value for a larger problem is a more difficult task then finding the optimal value for a smaller problem for online algorithms.

In conclusion, we observe that release times and experiment count N have bigger effect on total weighted tardiness than arrivals input parameter.

6.3.3 Volunteer Science Data Analysis

Volunteer Science [32] enables researchers to quickly design and deploy behavioral experiments while recruiting participants from a substantially larger subject pool of any active Facebook user. There are many experiments in the framework that run every day and require different number of participants. This platform enables us to collect participant arrival data easily from the conducted experiments. We use Volunteer Science participant arrival data from the last year. Arrival counts and inter-arrival times are shown in Figure 6.11. The dataset contains $92,081$ participant arrivals within the one-year period. We conduct optimality comparison test for our online algorithms by using the Volunteer Science's real world data.

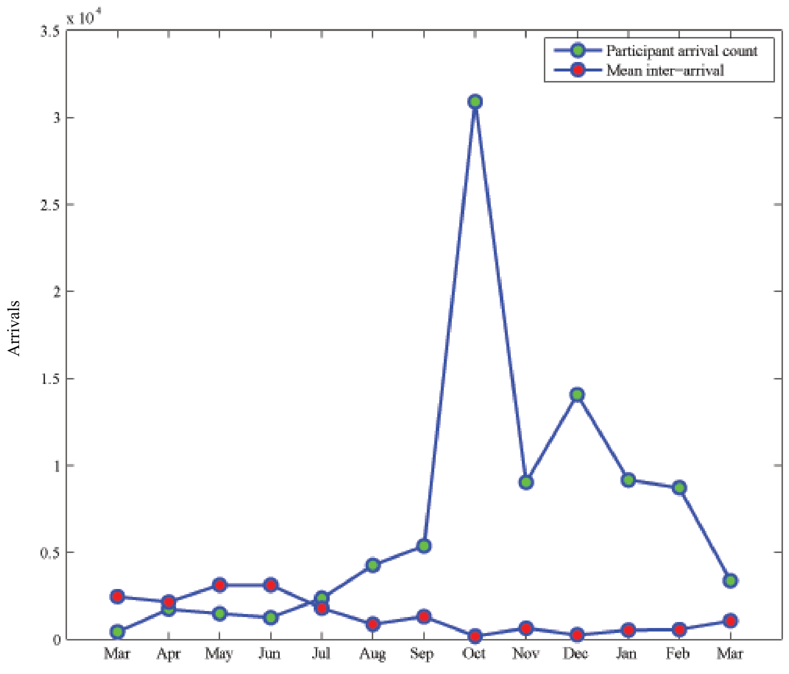


Figure 6.11: Participant arrivals count for 1 year period.

Optimality of the algorithms is calculated as ((result of online greedy algorithm - ILP result) $\div$ result of online greedy algorithm) $\times$ 100. The optimality test results are given in Figure 6.12. These results show that, in general, online greedy algorithms are far from being optimal and their results are close to each other. The best results are observed in SCR and MPRA algorithms.

We report the two sample t-test significance results for the optimality results of Volunteer Science data in In Table 6.9. The results show that there is no statistically significant difference between any of the average result values.

Table 6.9: Significance results for optimality comparisons of online greedy algorithms over Volunteer Science data.

	MS	MSP	$MPRA$	NF	$SIRT$	SCR	$SCRP$
MS	1.00	0.68	0.81	0.43	0.77	0.88	0.67
MSP	-	1.00	0.83	0.62	0.90	0.54	0.99
$MPRA$	-	-	1.00	0.51	0.94	0.67	0.83
NF	-	-	-	1.00	0.57	0.33	0.62
$SIRT$	-	-	-	-	1.00	0.64	0.89
SCR	-	-	-	-	-	1.00	0.53
$SCRP$	-	-	-	-	-	-	1.00

According to the above results, although SCR and MPRA performed slightly better than the other considered algorithms over the Volunteer Science data, one could use the other tested online algorithms for an online experimental environment since the results between the compared algorithms are not statistically significant. If one needs good performance with optimal solutions on the Volunteer Science SCR or MPRA can be used to get the best results. If one needs more stable results, MS can be used since its optimality performance is more stable than the other compared algorithms with a small range of change in error.

In order to determine how similar the created burstiness data to Volunteer Science's real world dataset, we conducted two-sided Wilcoxon rank sum test [68]. This test's null hypothesis assumes that the compared x and y data samples are from the same population, i.e., show the same behavior, and we reject the null hypothesis if the resulting p is less than 0.05. We compared each burstiness data with Volunteer Science data and the results are given in Table 6.10.

Table 6.10: Comparison of syntethic and real-world datasets.

$Dataset$	$p - value$
Nonbursty	0.010
Weak bursty	0.001
Strong bursty	0.001

These results imply that the burstiness dataset is not similar to the Volunter Science's real world data and Volunteer science data does not show a specific bursty behavior.

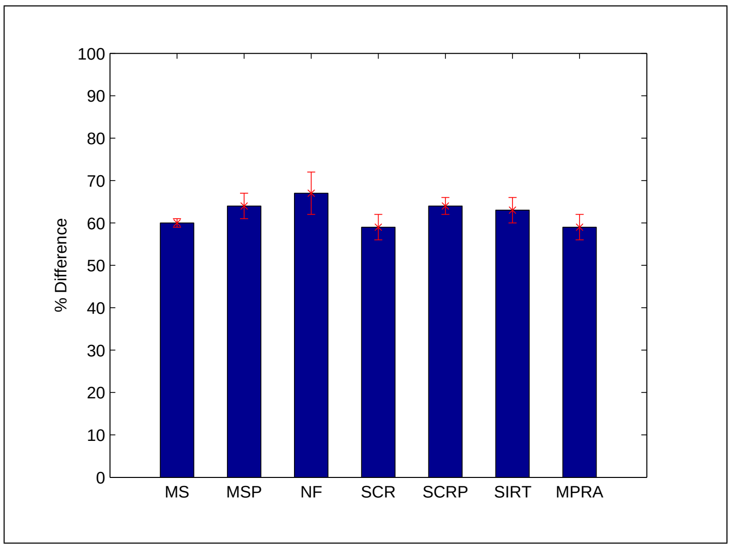


Figure 6.12: Percentage difference between online greedy algorithms and ILP results over Volunteer Science data after running its instances 10 minutes.

Chapter 7

Conclusions

In traditional laboratories participants are in an artificial environment, the sample size (ie. the number of arriving participants to the environment) is limited and conducting experiments can be expensive and time consuming. For the case of participant assignment problem in online laboratories, participants that are attending to experiments are in their natural settings, sample size is unlimited and conducting experiments can be cheap and quick. When the conducted experiments are not identical, it is difficult to satisfy research goals. For the case of participant assignment problem, there can be large number of experiments that each require varying number of participants and we want to assign participants to experiments so more important experiments finish earlier. For this reason our objective is to optimize participant assignment process for a customizable experimental framework to complete studies on time.

Our work represents the first formulation of participant assignment as a job scheduling problem that we are aware of. We developed and evaluated participant assignment algorithms for the following scenarios: 1) participants arrive sequentially and experiments are released at the beginning, 2) participants arrive at varying times and experiments are released at the beginning, 3) participants arrive sequentially and experiments are released at varying times, and 4) participants arrive at varying times and experiments are released at varying times. We also considered online and offline greedy heuristics, synthetic and real datasets and bursty and nonbursty data. This study represents the first structured analysis of participant assignment algorithms that considers these cases.

In the developed online experimental framework we assumed that participants arrive one at a time, N experimental instances are given with total instance size of K, and they have properties like experimental instance processing time, due date, and priority between 1 and 9. In the above definition experimental instance size is the number of participants needed for an experiment to start, experimental instance processing time refers to the execution time of an experimental instance, experimental instance due date refers to the desired latest completion time of an experimental instance, and priority between 1 and 9 refers to the importance of each experiment instance. An assignment is complete when the correct number of participants is assigned to experimental instances to make them start. The tardiness of an experimental instance i that starts at time t is the positive difference between its completion time and due date where experiment instance's completion time is the time at which the instance has finished processing. The weighted tardiness is then calculated by multiplying Tardiness (t) by the priority value (PR) of experiment i. In this framework our goal is to find an optimal assignment such that it is complete and the total weighted tardiness of the experiments is minimized.

For the participant assignment scenario where participants are arriving sequentially and experiments are released at the beginning we considered several algorithms such as exhaustive enumeration, greedy algorithms, dynamic programming, and integer-linear programming. Exhaustive Enumeration goes through every possible experimental assignment for each incoming participant to find the minimum total weighted tardiness. Greedy algorithm based on earliest due date assigns incoming participants to the experiment with the earliest due date. Greedy algorithm based on least-cost-last selects the experiment with the lowest weighted tardiness to be scheduled at last. Apparent tardiness cost greedy algorithm calculates an assignment index for every available experiment and assigns the next participant to the experiment with the highest assignment index. Apparent tardiness cost for participant assignment greedy algorithm calculates an assignment index in the same way but also considers experiment instance sizes while calculating assignment index. Dynamic programming solution solves the problem bottom up and integer-linear programming solution solves the problem by defining variables considering whether a participant assignment makes an experiment start or not. Algorithms are evaluated over synthetic dataset. Experimental instance size (ES) is assigned randomly from a uniform distribution with a range: 1 to 3, 4 to 6, or 7 to 9 A segment of the benchmark includes 50 instances. Each benchmark instance contains a set of N experiments and K participants are needed to make all experiments start.

The first test is the percentage of the benchmark that the algorithms can process within 1 minute and 10 minutes limitations and the results show that greedy algorithms are the fastest algorithms and integer-linear programming completes more quickly that exhaustive and dynamic programming algorithms. The second set of experiments is conducted to determine how close the solutions provided by these algorithms are to optimal. The solutions provided by the dynamic programming algorithm are used as a baseline. We applied the dynamic programming algorithm for

30 minutes to every instance in a subset of a benchmark segment that was not solved by any of our exact algorithms in less than 10 minutes. We then applied the other algorithms to this benchmark segment for 1 minute and 10 minutes for every instance. We report the difference between the cost of the solutions found by a target algorithm and the optimal cost, divided by the optimal cost. Exhaustive and ILP algorithms find solution that are on average 75.6% and 3.2% respectively from optimal for 1-minute tests, and 72.9% and 2.3% respectively from optimal for 10-minute tests EDD, LCL, ATC and ATCPA algorithms are 43.4%, 13.3%, 25.8% and 12.1% respectively from optimal for both 1-minute and 10 minutes tests. For harder dataset instances, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. According to results The average difference between LP and ILP is 26.5%, LP and EDD is 72.0%, LP and LCL is 60.9%, LP and ATC 66.2%, LP and ATCPA is 60.3%. According to optimality and LP comparison tests, ATCPA heuristic performs better than the other greedy heuristics. Finally by taking the ATCPA algorithm as baseline algorithm, we conducted nonparametric Wilcoxon test to determine if the difference between the compared greedy heuristics and the baseline greedy heuristic is statistically significant and the results indicate that the difference between ATCPA and the other compared greedy heuristics are statistically significant. For the scenario of sequential participant arrival where experiments are released at the beginning, overall results indicate that greedy algorithms outperformed the others in 1-minute and 10-minute tests. ILP works faster than the exhaustive and dynamic programming methods at finding the exact solutions. If one attaches more importance to faster execution of the assignment process, greedy algorithms should be chosen ATCPA greedy heuristic performs reasonably well at finding the exact solutions.

For the scenario of varying arrivals of participants to the experiments which are released at the beginning, the algorithms and the considered experiments are kept same. However, algorithms are modified in a way that it considers the varying arrivals of participants to the experiments. The algorithms consider an arrival array that contains varying arrival times of the participants. The execution test results show that greedy algorithms are the fastest algorithms and integer-linear programming completes more quickly that exhaustive and dynamic programming algorithms. According to optimality tests apparent tardiness cost algorithm outperform the other algorithms. Exhaustive and ILP algorithms find solution that are on average 11.07% and 9.05% respectively from optimal for 1-minute tests, and 10.96% and 0.01% respectively from optimal for 10-minute tests EDD, LCL, ATC and ATCPA algorithms are 6.29%, 3.36%, 0.09% and 3.59% respectively from optimal for both 1-minute and 10 minutes tests. For harder dataset instances, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. According to results the average difference between LP and ILP is 72.4%, LP and EDD is 77.2%, LP and LCL is 74.7%, LP and ATC 72.0%, LP and ATCPA is 74.6%. According to optimality and LP comparison tests, ATC heuristic outperforms the other greedy heuristics. Significance tests also verify that the difference between ATC and other algorithms is significant.

For the scenario of sequential arrivals of participants to the experiments which are released at varying times, in addition to same algorithms, online and offline greedy algorithms and branch and bound algorithm is also considered. Existing algorithms are modified so that they consider varying experiment release times by making use of an array that contains varying release times of the experiments. The execution test results show that online and offline greedy algorithms are the fastest algorithms and completes more quickly than the other algorithms. According to optimality tests within exact algorithms, ILP outperforms the other algorithms. For the case offline and online greedy algorithms, maximum priority with remaining attributes (MPRA) outperform the other greedy algorithms. For harder dataset instances, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. According to results the average difference between the LP lower bound and the optimal results is 6.6%. The average difference between the LP lower bound and the offline greedy algorithms MS, MSP, NF, SCR, SCRP, SIRT, MPRA are 0.4%, 0.3%, 0.7%, 0.4%, 0.3%, 0.5%, 0.2% respectively. In the same way the average difference between the LP lower bound and the online greedy algorithms MS, MSP, NF, SCR, SCRP, SIRT, MPRA are 0.4%, 0.3%, 0.7%, 0.4%, 0.4%, 0.5%, 0.3% respectively. According to online and offline greedy comparison results with LP results, it is seen again that MPRA is the best performing algorithm. Overall, these results imply that the greedy algorithms should be chosen to solve assignment problems based on uniform participant arrivals and varying experiment release times and MPRA can be a good choice for this purpose.

For the scenario of varying arrivals of participants to the experiments which are released at varying times, the algorithms are kept as the same with the previous scenario. However, the algorithms are modified so that an arrival array which contains the varying arrivals of the participants is also considered. The execution test results show that online and offline greedy algorithms are the fastest algorithms and completes more quickly than the other algorithms. According to optimality tests within exact algorithms, ILP outperforms the other algorithms. For the case offline and online greedy algorithms, maximum priority with remaining attributes (MPRA) outperform the other greedy algorithms. For harder dataset instances, we compare solutions found by heuristics to those found by relaxing the ILP formulation to give a lower bound on the total minimum weighted tardiness. According to results the average difference between the LP lower bound and the optimal results is 6.5%. The average difference between the LP lower bound and the offline greedy algorithms MS, MSP, NF, SCR, SCRP, SIRT, MPRA are 0.5%, 0.5%, 0.8%, 0.6%, 0.6%, 0.7%, 0.2% respectively. In the same way the average difference between the LP lower bound and the online greedy algorithms MS, MSP, NF, SCR, SCRP, SIRT, MPRA are 0.5%, 0.6%, 0.9%, 0.6%, 0.5%, 0.7%, 0.3% respectively. Similar to the previous scenario, these results imply that the greedy algorithms should be chosen to solve assignment problems based on varying participant arrivals and varying experiment release times and MPRA can be a good choice for this purpose.

Bibliography

[1] L. Tesfatsion, "Experimental design: Basic concepts and terminology," http://www2.econ.iastate.edu/tesfatsi/expdesign.pdf, 2013.

[2] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Trans. Software Eng.*, vol. 12, no. 7, pp. 733–743, 1986.

[3] S. Cooper, A. Treuille, J. Barbero, A. Leaver-Fay, K. Tuite, F. Khatib, A. C. Snyder, M. Beenen, D. Salesin, D. Baker, and Z. Popović, "The challenge of designing scientific discovery games," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, ser. FDG '10. New York, NY, USA: ACM, 2010, pp. 40–47. [Online]. Available: http://doi.acm.org/10.1145/1822348.1822354

[4] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002. [Online]. Available: http://dx.doi.org/10.1145/581571.581573

[5] L. DeBruine and B. Jones, "Face research," http://faceresearch.org/, 2013.

[6] L. von Ahn, "Games with a purpose," *IEEE Computer*, vol. 39, no. 6, pp. 92–94, 2006.

[7] C. Lintott, "Zooniverse," https://www.zooniverse.org/, 2013.

[8] K. Siorpaes and M. Hepp, "Games with a purpose for the semantic web," *IEEE Intelligent Systems*, vol. 23, no. 3, pp. 50–60, May 2008. [Online]. Available: http://dx.doi.org/10.1109/MIS.2008.45

[9] C. Garbin, "Psyc 350: Research methods and data analysis," http://psych.unl.edu/psycrs/350/index.html, 2013.

[10] J. Freese and J. Druckman, "Time-sharing experiments for the social sciences," http://www.tessexperiments.org/, 2013.

[11] R. Goldstone, "The group experiment environment project," http://groups.psych.indiana.edu/, 2013.

[12] W. Rafelsberger and A. Scharl, "Games with a purpose for social networking platforms," in *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, ser. HT '09. New York, NY, USA: ACM, 2009, pp. 193–198. [Online]. Available: http://doi.acm.org/10.1145/1557914.1557948

[13] S. Matyas, C. Matyas, C. Schlieder, P. Kiefer, H. Mitarai, and M. Kamata, "Designing location-based mobile games with a purpose: collecting geospatial data with cityexplorer," in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ser. ACE '08. New York, NY, USA: ACM, 2008, pp. 244–247. [Online]. Available: http://doi.acm.org/10.1145/1501750.1501806

[14] D. Jensen and J. Neville, "Data mining in social networks," in *In National Academy of Sciences Symposium on Dynamic Social Network Modeling and Analysis*, 2002.

[15] C.-C. Chang and Y.-C. Chin, "Predicting the usage intention of social network games: An intrinsic-extrinsic motivation theory perspective," *IJOM*, vol. 1, no. 3, pp. 29–37, 2011.

[16] M. Brennan, N. Rae, and M. Parackal, "Survey-based experimental research via the web: Some observations," *Marketing Bulletin*, vol. 10, pp. 83–92, 1999.

[17] W. Mason and D. J. Watts, "Collective problem solving in networks," http://dx.doi.org/10.2139/ssrn.1795224, 2011.

[18] ——, "Financial incentives and the "performance of crowds"," in *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ser. HCOMP '09. New York, NY, USA: ACM, 2009, pp. 77–85. [Online]. Available: http://doi.acm.org/10.1145/1600150.1600175

[19] R. Pastor, J. Sanchez, and S. Dormido, "A xml-based framework for the development of web-based laboratories focused on control systems education," *The International Journal of Engineering Education*, vol. 19, pp. 445–454, 2003.

[20] U.-D. Reips and C. Neuhaus, "Wextor: A web-based tool for generating and visualizing experimental designs and procedures," *Behavior Research Methods, Instruments, and Computers*, vol. 34, pp. 234–240, 2002. [Online]. Available: http://dx.doi.org/10.3758/BF03195449

[21] F. Keller, M. Corley, S. Corley, L. Konieczny, and A. Todirascu, "Webexp: A java toolbox for web-based psychological experiments - users' guide for webexp 2.1," Tech. Rep., 1998.

[22] C. Drew, M. Hardman, and J. Hosp, *Designing and Conducting Research in Education*. SAGE Publications, 2007. [Online]. Available: http://books.google.com/books?id=fYX7sfflKZYC

[23] A. Evans and B. Rooney, *Methods in Psychological Research*. SAGE Publications, 2010. [Online]. Available: http://books.google.com/books?id=ONlAlG4W38cC

[24] R. A. Fisher, *The design of experiments / by Sir Ronald A. Fisher*, 7th ed. Oliver and Boyd, Edinburgh :, 1960.

[25] E. L. Lawler, "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness," in *Studies in Integer Programming*, ser. Annals of Discrete Mathematics, B. K. P.L. Hammer, E.L. Johnson and G. Nemhauser, Eds. Elsevier, 1977, vol. 1, pp. 331 – 342. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167506008707428

[26] J. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," in *Studies in Integer Programming*, ser. Annals of Discrete Mathematics, B. K. P.L. Hammer, E.L. Johnson and G. Nemhauser, Eds. Elsevier, 1977, vol. 1, pp. 343 – 362. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016750600870743X

[27] C. R. Scrich, V. A. Armentano, and M. Laguna, "Tardiness minimization in a flexible job shop: A tabu search approach," *Journal of Intelligent Manufacturing*, vol. 15, pp. 103–115, 2004, 10.1023/B:JIMS.0000010078.30713.e9. [Online]. Available: http://dx.doi.org/10.1023/B:JIMS.0000010078.30713.e9

[28] T. Loukil, J. Teghem, and P. Fortemps, "A multi-objective production scheduling case study solved by simulated annealing," *European Journal of Operational Research*, vol. 179, no. 3, pp. 709–722, June 2007. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v179y2007i3p709-722.html

[29] C.-F. Liaw, Y.-K. Lin, C.-Y. Cheng, and M. Chen, "Scheduling unrelated parallel machines to minimize total weighted tardiness," *Comput. Oper. Res.*, vol. 30, no. 12, pp. 1777–1789, Oct. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0305-0548(02)00105-3

[30] H. Zhou, Z. Li, and X. Wu, "Scheduling unrelated parallel machine to minimize total weighted tardiness using ant colony optimization," in *Automation and Logistics, 2007 IEEE International Conference on*, Aug., pp. 132–136.

[31] L. A. Frederic Dugardin, Hicham Chehade, F. Yalaoui, and C. Prins, *Hybrid Job Shop and Parallel Machine Scheduling Problems: Minimization of Total Tardiness Criterion.* Eugene Levner, 2007.

[32] N. University, "Volunteer science," http://volunteerscience.com/, 2013.

[33] R. L. Goldstone and T. M. Gureckis, "Collective behavior," *Topics in Cognitive Science*, vol. 1, no. 3, pp. 412–438, 2009. [Online]. Available: http://dx.doi.org/10.1111/j.1756-8765.2009.01038.x

[34] J. Du and J. Leung, "Minimizing total tardiness on one machine is np-hard," *Mathematics of Operations Research*, vol. 15, pp. 483–495, 1990.

[35] C. Koulamas, "The single-machine total tardiness scheduling problem: Review and extensions," *European Journal of Operational Research*, vol. 202, no. 1, pp. 1 – 7, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221709002641

[36] F. Yalaoui and C. Chu, "Parallel machine scheduling to minimize total tardiness," *International Journal of Production Economics*, vol. 76, no. 3, pp. 265 – 279, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092552730100175X

[37] R. Alvarez-Valdes, A. Fuertes, J. M. Tamarit, G. Gimenez, and R. Ramos, "A heuristic to schedule flexible job-shop in a glass factory," *European Journal of Operational Research*, vol. 165, no. 2, pp. 525–534, September 2005. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v165y2005i2p525-534.html

[38] L. Mo?nch, "Heuristics to minimize total weighted tardiness of jobs on unrelated parallel machines," in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, Aug., pp. 572–577.

[39] N. Souayah, I. Kacem, M. Haouari, and C. Chu, "Scheduling on parallel identical machines to minimise the total weighted tardiness," *International Journal of Advanced Operations Management*, vol. 1, no. 1, pp. 30–69, 2009. [Online]. Available: http://www.ingentaconnect.com/content/ind/ijaom/2009/00000001/00000001/art00002

[40] M. Held and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 1, pp. 196–210, 1962. [Online]. Available: http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SMJMAP000010000001000196000001&idtype=cvips&gifs=yes

[41] S. Khuller and J. Mestre, "An optimal incremental algorithm for minimizing lateness with rejection," in *Algorithms - ESA 2008*, ser. Lecture Notes in Computer Science, D. Halperin and K. Mehlhorn, Eds. Springer Berlin Heidelberg, 2008, vol. 5193, pp. 601–610. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87744-8_50

[42] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[43] A. P. J. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, vol. 33, no. 8, pp. pp. 1035–1047, 1987. [Online]. Available: http://www.jstor.org/stable/2632177

[44] A. P. J. Vepsalainen, M. Greenberg, and T. E. Morton, "Minimizing total tardiness on one machine is np-hard," *Working Paper 85-09-03, Department of Decision Sciences, University of Pennsylvania*, 1985.

[45] S. Verma and M. Dessouky, "Single-machine scheduling of unit-time jobs with earliness and tardiness penalties," *Math. Oper. Res.*, vol. 23, no. 4, pp. 930–943, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1287/moor.23.4.930

[46] J. M. Akker, G. Diepen, and J. A. Hoogeveen, "Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs," *J. of Scheduling*, vol. 13, no. 6, pp. 561–576, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10951-010-0181-1

[47] M. M. Dessouky, "Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness," *Comput. Ind. Eng.*, vol. 34, no. 4, pp. 793–806, Sep. 1998. [Online]. Available: http://dx.doi.org/10.1016/S0360-8352(98)00105-3

[48] L. Mönch, H. Balasubramanian, J. W. Fowler, and M. E. Pfund, "Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times," *Comput. Oper. Res.*, vol. 32, no. 11, pp. 2731–2750, Nov. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2004.04.001

[49] A. Jafari, S. Hassani, and P. Chiniforooshan, "A hybrid differential evolution algorithm for scheduling parallel batch processing machine with unequal job ready times," *Proceedings of the World Congress on Engineering and Computer Science*, vol. 2, 2012.

[50] C. Reeves, "Heuristics for scheduling a single machine subject to unequal job release times," *European Journal of Operational Research*, vol. 80, no. 2, pp. 397 – 403, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0377221793E0290E

[51] M. Mahnam, G. Moslehi, and S. M. T. F. Ghomi, "Single machine scheduling with unequal release times and idle insert for minimizing the sum of maximum earliness and tardiness," *Mathematical and Computer Modelling*, vol. 57, no. 910, pp. 2549 – 2563, 2013, system Dynamics in Project Management; Applied Mathematics and Computational Science and EngineeringSelected Papers of the Seventh PanAmerican Workshop June 611 2010, Venezuela. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895717713000241

[52] C.-C. Wu, P.-H. Hsu, and K. Lai, "Simulated-annealing heuristics for the single-machine scheduling problem with learning and unequal job release times," *Journal of Manufacturing Systems*, vol. 30, no. 1, pp. 54 – 62, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0278612511000239

[53] M. Akturk and D. Ozdemir, "A new dominance rule to minimize total weighted tardiness with unequal release dates," *European Journal of Operational Research*, vol. 135, no. 2, pp. 394 – 412, 2001, financial Modelling. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221700003192

[54] T. Cheng, Z.-L. Chen, and N. Shakhlevich, "Common due date assignment and scheduling with ready times," *Computers and Operations Research*, vol. 29, no. 14, pp. 1957 – 1967, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054801000673

[55] L. Mönch, J. Zimmermann, and P. Otto, "Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines,"

Eng. Appl. Artif. Intell., vol. 19, no. 3, pp. 235–245, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.engappai.2005.10.001

[56] A. Kan, *Machine scheduling problems: classification, complexity and computations*. Nijhoff, 1976. [Online]. Available: http://books.google.com/books?id=-sMSAQAAMAAJ

[57] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Math. Program.*, vol. 46, no. 3, pp. 259–271, Feb. 1990. [Online]. Available: http://dx.doi.org/10.1007/BF01585745

[58] E. M. Arkin and E. B. Silverberg, "Scheduling jobs with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1 – 8, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X87900370

[59] V. Gabrel, "Scheduling jobs within time windows on identical parallel machines: New model and algorithms," *European Journal of Operational Research*, vol. 83, no. 2, pp. 320 – 329, 1995, ¡ce:title¿EURO Summer Institute Combinatorial Optimization¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037722179500010N

[60] Y. Xi and J. Jang, "Scheduling jobs on identical parallel machines with unequal future ready time and sequence dependent setup: An experimental study," *International Journal of Production Economics*, vol. 137, no. 1, pp. 1 – 10, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092552731200028X

[61] L. Yang-Kuei and L. Chi-Wei, "Dispatching rules for unrelated parallel machine scheduling with release dates," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1-4, pp. 269–279, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00170-013-4773-8

[62] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Sequencing and scheduling: algorithms and complexity," in *Logistics of Production and Inventory*, ser. Handbooks in Operations Research and Management Science, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, Eds. Elsevier, 1993, vol. 4, ch. 9, pp. 445–522. [Online]. Available: http://www.sciencedirect.com/science/article/B7P6G-4FKY22V-1B/2/e636a20dd1005f17be8e9704711ac821

[63] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960. [Online]. Available: http://jmvidal.cse.sc.edu/library/land60a.pdf

[64] G. Centeno and R. L. Armacost, "Parallel machine scheduling with release time and machine eligibility restrictions," *Computers and Industrial Engineering*, vol. 33, no. 12, pp. 273 – 276, 1997, ¡ce:title¿Proceedings of the 21st International Conference on Computers and Industrial Engineering¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835297000910

[65] ——, "Minimizing makespan on parallel machines with release time and machine eligibility restrictions," *International Journal of Production Research*, vol. 42, no. 6, pp. 1243–1256, 2004. [Online]. Available: http://www.ingentaconnect.com/content/tandf/tprs/2004/00000042/00000006/art00009

[66] H. Kellerer and V. Strusevich, "Scheduling parallel dedicated machines under a single non-shared resource," *European Journal of Operational Research*, vol. 147, no. 2, pp. 345 – 364, 2003, ¡ce:title¿Fuzzy Sets in Scheduling and Planning¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221702002461

[67] J. Tai, J. Zhang, J. Li, W. Meleis, and N. Mi, "Ara: Adaptive resource allocation for cloud computing environments under bursty workloads," in *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, Nov 2011, pp. 1–8.

[68] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 03 1947. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177730491