5-2018

Quantitative Methods For Select Problems In Facility Location And Facility Logistics

Bin Li
University of Arkansas, Fayetteville

Quantitative Methods For Select Problems In Facility Location And Facility Logistics

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

by

Bin Li
China University of Petroleum (North East)
Bachelor of Science in Management Science and Engineering, 2009
Beijing Institute of Technology
Master of Science in Management Science, 2012

May 2018
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Ashlea Bennett Milburn, Ph.D.
Dissertation Director

Shengfan Zhang, Ph.D.
Committee member

Jose Emmanuel Ramirez-Marquez, Ph.D.
Committee member

Chase Rainwater, Ph.D.
Committee member

Scott Mason, Ph.D.
Committee member

ABSTRACT

This dissertation presented three logistics problems. The first problem is a parallel machine scheduling problems that considers multiple unique characteristics including release dates, due dates, limited machine availability and job splitting. The objective of is to minimize the total amount of time required to complete work. A mixed integer programming model is presented and a heuristic is developed for solving the problem. The second problem extends the first parallel scheduling problem to include two additional practical considerations. The first is a setup time that occurs when warehouse staff change from one type of task to another. The second is a fixed time window for employee breaks. A simulated annealing (SA) heuristic is developed for its solution. The last problem studied in this dissertation is a new facility location problem variant with application in disaster relief with both verified data and unverified user-generated data are available for consideration during decision making. A total of three decision strategies that can be used by an emergency manager faced with a POD location decision for which both verified and unverified data are available are proposed: *Consider Only Verified*, *Consider All* and *Consider Minimax Regret*. The strategies differ according to how the uncertain user-generated data is incorporated in the planning process. A computational study to compare the performance of the three decision strategies across a range of plausible disaster scenarios is presented.

ACKNOWLEDGEMENTS

I would like to thank all the people who helped me during my doctoral studies, many of them have contributed to my studies and researches in their own particular way and for that I want to give them special thanks.

I am most grateful to my advisor Dr. Ashlea Bennett Milburn, who have provided me through guidance and support during my 5 years studies. Her criticism combined with heart-warming support have made me become more confident as a researcher, and at the same time made me realize that there is still a long road ahead to become a profession.

With a special mention to Dr. Scott Mason, who invested a huge amount of time in helping and guiding me on the first two chapters of my dissertation. Also Dr.Jose Emmanuel Ramirez-Marquez, who worked together with and mentoring me on the third chapter of my dissertation. Without their help I cannot finished such exciting work.

I am also grateful to the following university faculties: Dr. Shengfan Zhang, Dr. Chase rainwater, Dr. Manuel D. Rossetti, and Dr. Tish Pohl for their unfailing support and assistance.

And finally, last but not least, I wish to thank my family and the friends that support me during my entire studies and gives me the motivation to finsih this achievement.

TABLE OF CONTENTS

1 Introduction

According to the Council of Supply Chain Management Professionals (CSMCP), logistics is "the process of planning, implementing and controlling procedures for the efficient and effective transportation and storage of goods including services and related information from the point of origin to the point of consumption for the purpose of conforming to customer requirements. It includes inbound, outbound, internal and external movements" [1]. The resources managed in logistics include food, materials, equipment, time and information. A 2012 CSCMP report indicates the logistics cost as a percent of Gross Domestic Product (GDP) reached 8.5 percent [2].

The facility location problem is a critical logistics problem that has been studied for decades. It is a branch of operations research and computational geometry concerned with the optimal placement of facilities to minimize possible costs while considering factors like avoiding placing hazardous materials near housing and competitors' facilities. Facility location decisions are typically strategic in nature, given the operating horizon of most facilities. The location decisions are costly and difficult to reverse. However, problem parameters that affect location decisions, such as costs and demands, may fluctuate widely during a facility's operating horizon. Because of this, researchers have been developing models for facility location under uncertainty in recent years.

Facility logistics includes such problems as facility design and material handling for manufacturing, distribution and service facilities. Questions regarding both design and the operational aspects of material and information flow within facilities must be addressed with

the goal of improving productivity and performance. Human labor within warehouses falls within the scope of facility logistics and human labor costs are reported to comprise over half (55%) of total logistics costs [2].

This dissertation discusses one facility location problem and two facility logistics problems. The first problem is a facility location problem with application in humanitarian logistics, and is novel in that a new class of unverified user-generated data is considered along with verified data during the decision making process. The second and third problems are warehouse labor scheduling problems where employees have varying productivities for the different tasks that need to be performed.

In Chapter 2, an unrelated parallel machine scheduling problem with release dates, due dates, limited machine availabilities and job splitting is studied. It is motivated by a problem faced by a large medical supplier interested in decreasing human labor costs in their warehouses through optimized employee scheduling. Specifically, an opportunity to reduce the total amount of time required to complete work inside the DCs each day exists by considering individual employee productivities for each task. The objective is to find a schedule that minimizes the total amount of time to process all jobs. A mixed integer program (MIP) is presented and a constructive heuristic with local search is developed. The performance of the heuristic is compared to a commercial optimization solver on a wide variety of test instances. Computational results indicate the heuristic is fast and effective in practice.

In Chapter 3, a more complex variant of the problem presented in Chapter 2 is introduced. The unrelated parallel machine scheduling problem retains the release dates, due dates, limited machine availabilities, job splitting, and unique objective problem character-

istics. Additionally, machine downtime must be scheduled (for employee breaks) and there are penalties associated with switching from some types of tasks to others. A meta-heuristic is developed for solving the problem and its performance is compared to a commercial optimization solver.

The new facility location problem variant is the focus of Chapter 4 and has application within humanitarian logistics. Humanitarian logistics is a branch of logistics specializing in the delivery and warehousing of supplies during natural disasters or other types of emergencies [3]. Specifically, in this chapter, the location of Points of Distribution (PODs) for emergency relief supplies is considered. The problem is unique in that both verified data and unverified user-generated data are available for consideration during decision making. This models the recent need to integrate unverified social data (e.g., Twitter posts) with data from more traditional sources, such as on-the-ground assessments, to make optimal decisions during disaster relief. Integrating social data can enable identifying larger numbers of needs in shorter amounts of time, but because the information is unverified, some of it may be inaccurate. This paper seeks to provide a "proof of concept" illustrating how the unverified social data may be exploited. To do so, a framework for incorporating uncertain user-generated data when locating Points of Distribution (PODs) for disaster relief is presented. Then, three decision strategies that differ in how the uncertain data is considered are defined. Finally, the framework and decision strategies are demonstrated via a small computational study to illustrate the benefits user-generated data may afford across a variety of disaster scenarios.

2 A Heuristic for Scheduling Unrelated Parallel Machines Subject to Job Splitting

2.1 Introduction

Distribution centers (DCs) form the backbone of logistics systems, serving as hubs for storing and rerouting goods. A 2012 Council of Supply Chain Management Professionals (CSCMP) report indicates that warehousing costs in U.S. logistics systems total approximately $143 billion annually, comprising 29.2% of total carrying costs [2]. Primary DC activities include transferring, shipping, receiving and storing goods. Many of these activities require human labor, which plays an important role in logistics systems. According to the CSCMP report, labor costs comprise over half (55%) of total logistics costs [2]. Napolitano (2011) states that to control and manage this cost, managers often rely on labor management programs that promote and measure efficient methods for performing tasks [4]. In some cases, an employee may have higher than baseline performance for a certain task, meaning they can complete the task faster than another employee, according to Harrington (2008) [5]. Employee performance can be estimated, for example, through historical data. Therefore, in a DC context, an employee who is very productive in picking, packing, and/or receiving will finish the job efficiently and potentially save labor costs for the company.

Our research is motivated by a problem faced by a large medical product supplier. They are interested in decreasing labor costs in their DCs through optimized employee scheduling. Specifically, an opportunity exists to reduce the total amount of time required to complete work inside the DCs each day by considering individual employee productivities

for each task.

The work that must be completed within their DC includes tasks such as carting, picking, reaching, receiving, and shipping. For every task, a set of information is known: (i) a ready time describing when it becomes available, (ii) a cutoff time describing when it must be completed, (iii) a standard duration indicating how long it will take to complete the task under baseline productivity, and (iv) a vector of employee productivities describing how much faster or slower than baseline productivity each employee can complete the task. Tardiness is not permitted. Some tasks can be split among multiple employees, and some cannot. For those that can be split among employees, those employees can work on the task simultaneously. A set of employees is available for completing tasks. For each employee, the known information includes: (i) the time their shift begins, (ii) the time their shift ends, (iii) a vector of task eligibilities describing the set of tasks the employee is trained to complete, and (iv) a vector of task productivities analogous to the vector of employee productivities described above. To account for breaks, employees can be busy at most a certain percentage of their shifts (e.g., 85%). Furthermore, employees can only work on one task at a time; there is no preemption in this problem. All task and employee information is available to decision makers at the beginning of each day. It is assumed that employee productivity does not change throughout the day. The problem is to assign all tasks to employees subject to task ready time, task cutoff time, and employee shift constraints.

The objective of our problem is to minimize the total work hours required to perform all tasks, which is not a typical objective in scheduling problems. We are choosing this objective due to a business consideration from the company. In the company's DCs, work hours are directly related to labor costs, as workers are paid hourly wages. Therefore,

minimizing total work hours is equivalent to minimizing total labor costs. The common scheduling problem objective of makespan is not equivalent to total work hours. Makespan is the difference between the time the last job finishes and the time the first job started. Total work hours, on the other hand, accounts for the fact that multiple machines (i.e., people) are working simultaneously and each incurs a cost for time spent working. Another common scheduling problem objective is to minimize tardiness. This is also not appropriate for our problem, as tardiness is generally not allowed in the company's DC.

The contributions of this chapter are as follows. First, a new problem is introduced. Specifically, the problem described above is modeled as an unrelated parallel machine scheduling problem with release dates (ready times), due dates (task cutoff times), limited machine availabilities (employee shift schedules) and job splitting for some of the tasks. To the best of our knowledge, these problem characteristics have not been simultaneously considered in the personnel or machine scheduling literature. Furthermore, a unique objective function is used. The objective is to minimize the total job processing time on all machines, which is different than traditional scheduling problem objectives such as minimize makespan and tardiness. Second, a constructive heuristic with local search for solving the problem variant where all jobs can be split is developed. A computational study is conducted to compare its performance with a commercial optimization solver. The study includes 480 test instances that vary according to a number of factors such as number of machines, number of tasks, timing of release dates and due dates, and machine productivity and availability. Results demonstrate the heuristic is fast and effective.

The remainder of this chapter is organized as follows. In the next section, we review related literature. In Section 2.4, a formal problem statement is provided and a mixed integer

programming formulation (MIP) is introduced. The details of a constructive heuristic and local search improvement scheme used to solve the problem variant where job splitting is allowed for all jobs are presented in Section 2.5. Section 2.6 describes a computational study used to compare the developed heuristic with a commercial optimization solver. Finally, results and directions for future work are provided in Section 2.7.

2.2 Literature Review

We first discuss the relationship between this research and the personnel scheduling literature. Ernst et al. (2004) classified rostering and personnel scheduling problems into 17 categories, including, for example, crew scheduling, days off scheduling, shift scheduling, and task assignment [6]. These categories of problems generally involve allocating suitably qualified staff to meet time-dependent demand for various services while observing industrial workplace agreements and attempting to satisfy individual work preferences. For example, crew scheduling, typically applied in transportation systems, involves the assignment of crew members to duty schedules. This problem is more commonly referred to as shift scheduling when encountered in other systems (e.g., healthcare systems). Days off scheduling requires determining the off-work days for each worker over a planning horizon. In these problems, employees must be assigned to a set of working shifts having predetermined begin and end times. Furthermore, individual skillsets and work preferences are often considered when making decisions. In contrast, the problem described in this paper is to allocate a set of tasks between a group of workers whose work shifts are known. The category of personnel scheduling problems defined in Ernst et al. (2004) most similar to this is task assignment, in which a set of tasks with known start and end times must be allocated to workers [6].

The decision variables in our problem include not only task allocation decisions, but also the determination of task start and end times. Therefore, the similarity of this research to the personnel scheduling literature is limited. We turn our attention to a review of the machine scheduling literature.

By modeling the employees in this research as machines, we find related research in the machine scheduling literature. Though the parallel machine scheduling problem with identical machines has been studied extensively in the last few decades, the unrelated machines variant has received less attention in the literature. A survey of the unrelated parallel machine scheduling literature is presented in Pfund, Fowler, and Gupta (2004) [7]. A total of 44 papers are reviewed in the survey. The survey indicates that while minimization of makespan has been fairly widely studied, problems that include processing characteristics such as release times, sequence dependent setup times, and preemption remain largely unstudied. Problems of minimizing total weighted tardiness have likewise not been well studied. The survey also indicates that much more work needs to be done to solve unrelated parallel machine problems involving due date-related performance measures. Papers that address unrelated parallel machine scheduling with some of the additional characteristics present in the problem studied in this paper (i.e., limited machine availability and job splitting) are reviewed below.

The unrelated parallel machine scheduling problem with due date constraints represents an important but relatively less-studied scheduling problem in the literature. We begin our review of the unrelated parallel machine scheduling literature with those papers that include due dates and setup times. Ravetti *et al.* (2007) addresses an unrelated parallel machine scheduling problem with sequence dependent setups and due dates, where the ob-

jective is to minimize the sum of the makespan and weighted delays. A metaheuristic based on greedy randomized adaptive search procedure (GRASP) is used as a solution method [8]. Chen (2009) discusses a similar unrelated parallel machine problem with due dates and sequence and machine-dependent setup times where the objective is to minimize total tardiness. Computational results indicate that the proposed heuristic is capable of obtaining significantly better solutions than state-of-the-art algorithms on a benchmark problem set [9]. Ying and Lin (2012) also addresses an unrelated parallel machine scheduling problem with sequence and machine-dependent setup times under due date constraints. An artificial bee colony (ABC) algorithm is used to minimize total tardiness. Computational results show the ABC algorithm outperforms existing algorithms for most test instances studied [10].

Unrelated parallel machine scheduling problems having both due dates and release dates are less studied in the literature also. Bank and Werner (2001) considers an unrelated parallel machine problem with release dates where every job has a common due date and the objective is to minimize the weighted sum of linear earliness and tardiness penalties. Various constructive and iterative heuristic algorithms are compared on problems with up to 500 jobs and 20 machines [11]. Valente and Alves (2004) also studies a problem with release dates and a common due date where the objective is to minimize the sum of deviations of job completion times. However, the proposed algorithm can only solve problems having a single machine [12].

Limited machine availability is used to model situations in which machines cannot be assumed to be available 100% of the time, for example, during scheduled maintenance. In our paper, we use it to model employee shift constraints. Other unrelated parallel machine scheduling problems in the literature with limited machine availability are as follows.

Suresh and Ghaudhuri (1996) considers a problem were each machine has several unavailable periods and the objective is to minimize makespan. The problem is transformed by treating unavailable periods as jobs that can only be performed on the associated machine [13]. Cheng, Hsu, and Yang (2011) study an unrelated parallel machine scheduling problem where machine availabilities model deteriorating maintenance activities and the objective is to minimize total completion time. The length of maintenance activities (i.e., machine unavailabilities) increases linearly with starting time [14]. In another paper, Yang (2013) also studies similar unrelated parallel machine problems with deterioration effects and aging effects with multi-maintenance activities [15]. Currently there are no papers in the literature that simultaneously consider limited machine availabilities in conjunction with due dates and release dates.

Job splitting is not a common problem characteristic treated in the unrelated parallel machine scheduling literature, as currently there are only a few papers meeting this description. Logendran and Subur (2004) studies an unrelated parallel machine scheduling problem where jobs can be split, but only into two equal or nearly equal portions. The objective is to minimize the total weighted tardiness of all jobs. A tabu search algorithm is applied and a factorial experiment based on a split-plot design is performed to test the heuristic on a range of problems having between 9 and 60 jobs and 3 and 15 machines [16]. Eroglu, Ozmutlu, and Koksal (2013) discuss an unrelated parallel machine scheduling problem with sequence-dependent setup times and job splitting where the objective is to minimize makespan. A genetic algorithm is proposed and computational results indicate the algorithm is capable of solving problems with 75 machines and 111 jobs in a reasonable amount of CPU time [17]. Our treatment of job splitting is different from the above, in that jobs can be split arbitrarily

among eligible machines, as opposed to into two equal portions.

Authors	**Objective** (Minimize)	**UPM**	**Due Dates**	**Release Dates**	**Lim. Avail.**	**Job Splitting**
Ravetti *et al.* (2007)	makespan, weighted delay	✓	✓			
Chen (2009)	total tardiness	✓	✓			
Ying and Lin (2012)	total tardiness	✓	✓			
Bank and Werner (2001)	total tardiness	✓	✓	✓		
Valente and Alves (2004)	sum of deviations of job completion times		✓	✓		
Suresh and Ghaudhuri (1996)	makespan	✓			✓	
Cheng, Hsu, and Yang (2011)	total completion time	✓			✓	✓
Yang (2013)	total completion time	✓			✓	✓
Logendran and Subur (2004)	total weighted tardiness	✓		✓		✓
Eroglu, Ozmutlu, and Koksal (2013)	makespan	✓				✓

Table 2.1: Summary of reviewed unrelated parallel machine scheduling papers

Table 2.1 summarizes the papers reviewed above and indicates the problem characteristics included in each paper. The abbreviation UPM is for unrelated parallel machines. Note that none of the papers consider all five problem characteristics simultaneously. By doing so, our paper fills a gap in the literature. Additionally, a unique problem objective is considered in our paper. Specifically, the objective is to minimize the total amount of time required to complete the work, which is different from makespan (the time the last job completes) and the sum of completion times.

2.3 Problem Description

2.4 Problem Description

The problem presented in this paper is described formally as follows. There are two disjoint sets $\mathcal{S}$ and $\mathcal{T}$ of jobs that must be scheduled. For jobs in $\mathcal{S}$ splitting is allowed, but jobs in $\mathcal{T}$ cannot be split. The set of all jobs is $\mathcal{N} = \mathcal{S} \bigcup \mathcal{T}$. Each job $j \in \mathcal{N}$ can begin no earlier than its release date r_j and must be finished before its due date d_j. The standard duration of job j on a machine with baseline productivity is known and is denoted u_j. There is a set $\mathcal{M}$ of unrelated parallel machines available for processing jobs. The availability of each machine $i \in \mathcal{M}$ is described by a shift begin time b_i and shift end time e_i; a machine cannot process jobs before b_i or after e_i. Furthermore, each machine can only process jobs for $(1 - \alpha)\%$ of the time they are available, as $\alpha\%$ of a machine's shift must be reserved for break time. The binary parameter t_{ij} describes whether machine i is eligible to process job j (1 indicates the machine is eligible; 0 indicates it is not). For every job j for which machine i is eligible, the productivity ρ_{ij} of machine i for job j is known. If $\rho_{ij} < 1$, machine i has faster than baseline productivity for job j; if $\rho_{ij} > 1$, machine i has slower than baseline productivity for job j. Therefore, the time required for machine i to complete all of job j is $\rho_{ij} u_j$. However, it should be noted that for jobs in $\mathcal{S}$ job splitting is allowed and multiple machines can work simultaneously on portions of the same job. Let X_{ij} represent the fraction of job j assigned to machine i. For $j \in \mathcal{S}$, $X_{ij} \in \{0, 1\}$, but for $j \in \mathcal{T}$, X_{ij} can take fractional values. Then, the time machine i will spend on job j is $\rho_{ij} u_j X_{ij}$. The problem is to assign each job to an eligible machine or a set of eligible machines, such that job release date and due date constraints and machine availability constraints are not violated. The objective is

to minimize the total amount of time required to process all jobs.

Below we offer a mixed integer programming (MIP) formulation for the unrelated parallel machine scheduling problem defined above, with job splitting allowed for some jobs. Then we present a heuristic for solving the problem in the case when all jobs are splittable and the variables X_{ijk} are continuous. For the application we considered, the assumption that every job can be split is reasonable. The jobs include tasks such as order picking and pallet picking. If there are, for example, ten standard hours of order picking that must be completed in the warehouse on a given day, the work can be divided among any number of pick lists and then be assigned to employees (i.e. machines). All jobs in the application we consider are of this nature.

The mixed integer programming (MIP) formulation of the problem is described in Equations (2.1)-(2.15). Decision variable X_{ijk} is the fraction of job j assigned to the k^{th} position of machine i. The index k is used to keep track of the sequence of jobs for each machine, and $\mathcal{K}$ is the set of positions available on each machine. Binary decision variable Z_{ijk} indicates whether any portion of job j is assigned to machine i in position k. Additionally, decision variables s_{ik} and c_{ik} are used to keep track of the start and completion times of the k^{th} job on machine i, respectively.

$$\min \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{N}} \sum_{k\in\mathcal{K}} \rho_{ij} u_j X_{ijk} \tag{2.1}$$

subject to

$$\sum_{i\in\mathcal{M}} \sum_{k\in\mathcal{K}} X_{ijk} = 1, \forall\ j \in \mathcal{N} \tag{2.2}$$

$$Z_{ijk} \leq t_{ij},\ \forall\ i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{K} \tag{2.3}$$

$$X_{ijk} \leq Z_{ijk},\ \forall\ i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{K} \tag{2.4}$$

$$\sum_{j \in \mathcal{N}} Z_{ijk} \leq 1,\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.5}$$

$$c_{ik} = s_{ik} + \sum_{j \in \mathcal{N}} \rho_{ij} u_j X_{ijk},\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.6}$$

$$s_{ik} \geq c_{i(k-1)},\ \forall\ i \in \mathcal{M}, k \in \mathcal{K}(k \neq 1) \tag{2.7}$$

$$s_{ik} \geq \sum_{j \in \mathcal{N}} (r_j Z_{ijk}),\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.8}$$

$$c_{ik} \leq \sum_{j \in \mathcal{N}} (d_j Z_{ijk}) + G(1 - \sum_{j \in \mathcal{N}} Z_{ijk}),\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.9}$$

$$\sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{N}} \rho_{ij} u_j X_{ijk} \leq (1 - \alpha)(e_i - b_i)\ \forall\ i \in \mathcal{M} \tag{2.10}$$

$$s_{ik} \geq b_i,\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.11}$$

$$c_{ik} \leq e_i,\ \forall\ i \in \mathcal{M}, k \in \mathcal{K} \tag{2.12}$$

$$0 \leq X_{ijk} \leq 1 \; \forall \; i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{K} \tag{2.13}$$

$$Z_{ijk} \in \{0, 1\} \; \forall \; i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{K} \tag{2.14}$$

$$s_{ik}, c_{ik} \geq 0 \; \forall \; i \in \mathcal{M}, k \in \mathcal{K} \tag{2.15}$$

The objective described in Equation (2.1) is to minimize the total time required to process all jobs. Constraint set (2.2) requires that all of each job be assigned to some machine or set of machines. Constraint set (2.3) ensures that machines only operate on jobs for which they are eligible, and Constraint set (2.4) links the X and Z variables. Constraint set (2.5) states that at most one job can occupy each position on each machine. Constraint set (2.6) assures that the time the job in the k^{th} position on machine i completes is the time it started plus the time required for machine i to complete the assigned portion of the job. Constraint set (2.7) ensures that the time the job in the k^{th} position on machine i begins is no earlier than the time the previous job ended. Constraint set (2.8) assures that machines cannot begin working on a job before the job is ready, while constraint set (2.9) enforces the due date for a job. The parameter G represents a large value so that when there is no job in position k, the completion time for position k (c_{ik}) will not be constrained. Constraint set (2.10) states that a machine can work no longer than its shift duration, excluding break time. Constraint sets (2.11) and (2.12) ensure that a machine cannot process jobs before its shift begins or after it ends. Constraint sets (2.13) through (2.15) are variable definition constraints.

A simple example with three jobs and two machines is included to illustrate this problem. In this example, we are assuming there is no break time ($\alpha = 0$). Table 2.2 provides the job details. Each job has a release date, due date, and a standard duration. For example, job 1 can be completed between 8:00 and 14:00 and requires 5.00 standard hours. Table 2.3 gives the machine details. The productivity information indicates how fast (or slow) each machine is for each job. For example, machine A requires 1.1 times the standard duration of job 1 to process job 1. Missing productivity values indicate the machine is not eligible for the job. For example, machine B is not eligible for job 2. The machine availability information indicates the shift begin and end time for each machine. For example, machine A is available between 8:00 and 16:00. A feasible solution is contrasted with the optimal solution as follows.

j	**Ready Time** (r_j)	**Due Date** (c_j)	**Duration** (u_j)
1	8.00	14.00	4.89
2	10.00	20.00	4.00
3	12.00	19.00	5.00

Table 2.2: Example: Jobs List

i	**Productivity** (ρ_{ij})			**Machine Availability**	
	$j = 1$	$j = 2$	$j = 3$	b_i	e_i
A	1.1	1.0	0.9	8:00	16:00
B	0.8	–	1.2	12:00	20:00

Table 2.3: Example: Machines List

A feasible (but not optimal) solution is described in Table 2.4 and depicted in Figure

2.1. The time labels in Figure 2.1 have been rounded to the nearest minute. First, note that job 2 can only be performed by machine A due to eligibility restrictions, and machine A will require 4 hours to process all of job 2. Job 2 is not yet available at the start of machine A's shift, so it is assigned to the last part of machine A's shift, from 12:00 to 16:00. This renders machine A unable to perform any of job 3, even though A is faster than B for job 3, because job 3 is released at time 12:00 but A is busy with job 2 at that time. Therefore, all of job 3 will be assigned to B, and 6 hours will be required (5×1.2). Working backwards from the release date and due date of job 3 and shift of machine B, job 3 can be placed in machine B's schedule from 13:00 to 19:00. Machine B is faster than machine A for job 1, but is now busy throughout the entire window of job 1 except for 12:00 to 13:00. Therefore, machine B is assigned to work on job 1 from 12:00 to 13:00, and the remainder of job 1 is assigned to machine A, which processes the job from 8:00 to 12:00. The total time required to complete all work in this solution is 15 hours.

Table 2.4: Fraction of Each Job Assigned to Each Machine in Feasible Solution

	Job 1	**Job 2**	**Job 3**
Machine A	0.743	1.000	0.000
Machine B	0.257	0.000	1.000

The optimal solution described in Table 2.5 and depicted in Figure 2.2 is obtained by solving the MIP presented above. In the figure, times are rounded to the nearest minute. Job 1 is split into 2 pieces, but more of the job is given to machine B than A, since machine B is faster for job 1 than machine A. As the previous solution presented, job 2 is still performed

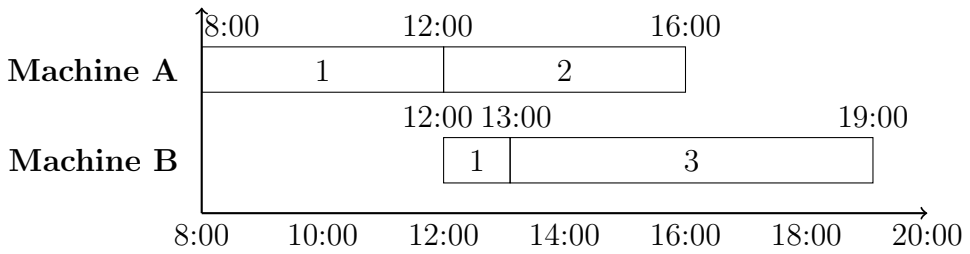


Figure 2.1: Example: A Feasible Schedule

by machine A in the optimal solution, since machine A is the only eligible machine. Like job 1, job 3 is also split. Machine A is faster for job 3 so machine A is assigned as much of job 3 as possible, given other constraints on machine availability and job release dates and due dates. The total processing time associated with this solution is 14.17 hours.

Table 2.5: Fraction of Each Job Assigned to Each Machine in Optimal Solution

	Job 1	**Job 2**	**Job 3**
Machine A	0.489	1.000	0.304
Machine B	0.511	0.000	0.696

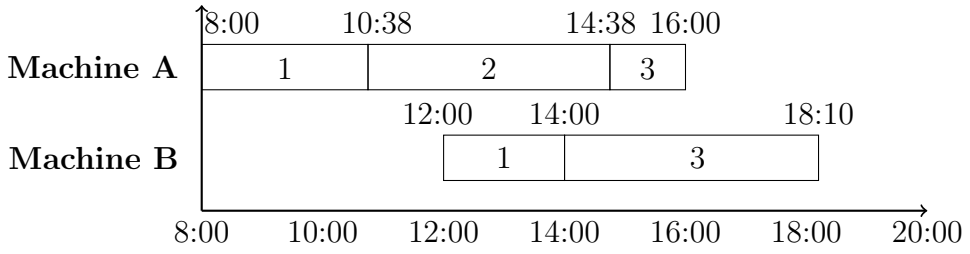


Figure 2.2: Example: An Optimal Schedule

2.5 Methods

Two methods are used to solve the problem presented in this paper. First, a commercial solver is used to obtain solutions to the MIP presented in Section 2.5. Next, a constructive heuristic is used in conjunction with a local search improvement scheme to produce approximate solutions. The details of the constructive heuristic and local search method are provided in this section.

The primary steps of the constructive heuristic used to generate an initial solution are described in Algorithm 1. Let v_j be the unassigned duration of job j during some step of Algorithm 1, and $V = \sum_{j=1}^{N} v_j$ be the total unassigned durations of all jobs. Note that in the initialization step, all jobs are unassigned, so V is the sum of the standard durations of all jobs, $V = \sum_{j=1}^{n} u_j$. While $V > 0$ and thus some unassigned job durations remain, the function **FindJob()** is used to select a job for assignment to a machine. Specifically, **FindJob()** finds the job $\hat{j} \in \mathcal{N}$ currently having maximum unassigned duration v_j. Next, the function **FindMachine()** identifies the eligible machine having the best productivity for job $\hat{j}$ (i.e., lowest $\rho_{i\hat{j}}$). If the machine identified is not available to complete some portion of $\hat{j}$ (for example, the release date and due date may not match up with the machine availability), the availability of the next-most productive machine for job $\hat{j}$ will be inspected, and so on. Once some machine $\hat{i}$ that can complete some portion of $\hat{j}$ has been identified, some portion $X_{\hat{i}\hat{j}\hat{k}}$ of it will be assigned to position $\hat{k}$ of machine $\hat{i}$ according to function **AddSchedule()**, which will be described later. If no such machine $\hat{i}$ can be identified, the algorithm terminates with an infeasible solution. Note that some portion of job $\hat{j}$ may remain unassigned. Upon completion of **AddSchedule()**, the unassigned duration $v_{\hat{j}}$ is updated,

as is the total unassigned duration of all jobs V. Then the algorithm returns to step 2 to find another job for assignment. Eventually, the algorithm terminates and returns initial schedule S. The schedule S will then be passed to the improvement algorithm.

Algorithm 1 Constructive Heuristic for Finding Initial Solution

Require: Machines $i = 1, \ldots, m$ and jobs $j = 1, \ldots, n$

1: **Initialize:** $v_j = u_j \ \forall \ j \in \mathcal{N}$, $V = \sum_{j \in \mathcal{N}} u_j$

2: **while** $V > 0$ **do**

3: $\quad \hat{j} =$ **FindJob**$(\mathcal{N})$

4: $\quad \hat{i} =$ **FindMachine**$(\hat{j}, \mathcal{M})$

5: $\quad S =$ **AddSchedule**$(\hat{j}, \hat{i})$

6: $\quad v_{\hat{j}} \leftarrow v_{\hat{j}} - u_{\hat{j}} X_{\hat{i}\hat{j}\hat{k}}$; $V \leftarrow v_{\hat{j}} - u_{\hat{j}} X_{\hat{i}\hat{j}\hat{k}}$

7: **end while**

8: **return** S

To find the appropriate position for job $\hat{j}$ in the schedule of machine $\hat{i}$, and the appropriate amount of the job to assign, **AddSchedule()** is as follows:

Step 1 Assume positions $1, 2, \ldots \beta$ in the schedule of machine $\hat{i}$ have been filled. Then, there are $\beta + 1$ possible insertion locations for job $\hat{j}$: one preceding each of the β positions, and one following the β^{th} position. Choosing any one of these locations will result in a modified alternative schedule for machine $\hat{i}$; not only will a new job fill a new position in the schedule, but the start and completion times of other jobs already in the schedule may change. Alternative schedules are generated where jobs are moved around so that maximum duration "gaps" are created for inserting a new job.

Figure 3 illustrates this idea. Suppose there is a machine for which two tasks are currently assigned in the initial schedule: a and b, in that order. Alternative 1 is to place the new job before a. Moving jobs a and b to begin as late as possible without violating the due date of either generates a large "gap" of available time prior to job a. Alternative 2 is to place the new job between a and b. Job a cannot be moved any earlier due to its release date and the schedule begin time, but job b can be moved as late as possible without violating its due date. This generates a large window of time between job a and b. The final alternative is to place the new job after b. Ideally, both jobs a and b would be moved to begin as early as possible without violating the release date of either. However, in this example, there is actually no room to adjust the start times of a or b. But, there is still a window of time between the completion of b and the end of the machine's schedule.

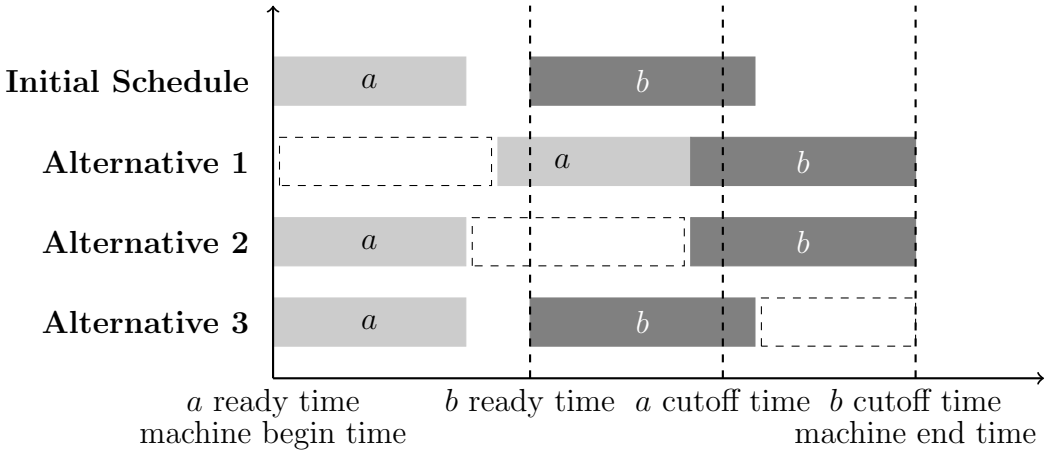


Figure 2.3: Illustration of Alternative Schedules and Insertion Locations

To create the maximum duration "gap" associated with potential insertion location L on machine i, first move all of the preceding jobs on the machine to begin as early as

possible. That is, for job j_1 in position 1, update the earliest possible start time to $\sigma_1 = max\{r_{j_1}, b_i\}$, the later of the release date or the machine shift begin time. Define the modified completion time of job j_l in position l as γ_l, computed as:

$$\gamma_l = \sigma_l + \rho_{ij_l} u_{j_l} X_{ij_l l}. \tag{2.16}$$

Then for jobs in positions $l = 2, 3, \ldots, L-1$, find the earliest possible start time σ_l using:

$$\sigma_l = \max\{\gamma_{l-1}, r_{j_l}\}. \tag{2.17}$$

Each job in each position will begin as soon as either the job is released or the previous job ends (when begun at its modified earliest possible start time σ_{l-1}), whichever comes latest. Next, move all of the subsequent jobs on the machine after position L to begin/complete as late as possible. That is, beginning with position $l = \beta$, find the latest possible completion time γ_β for the job in position β using:

$$\gamma_\beta = \min\{e_i, d_{j_\beta}\}. \tag{2.18}$$

Job j_β in position β will begin as late as possible without violating the end of the shift of machine i or the due date of j_β. Note that modified start time σ_β can be determined from γ_β by rearranging Equation (2.16). Then, working backwards from positions $l = \beta - 1$ to $l = L + 1$, find each latest possible completion time γ_l using:

$$\gamma_l = \min\{\sigma_{l+1}, d_{j_l}\}, \tag{2.19}$$

and update the modified starting times σ_l accordingly. Jobs in positions $L+1$ through $\beta - 1$ will begin as late as possible without violating the due date of the current job or the modified start time of the next job.

Step 2 Update the starting times $s_{\hat{i}l}$ and completion times $c_{\hat{i}l}$ for all $l = 1, 2, \ldots, \beta$ except $l = L$ to the times σ_l and γ_l identified above. Then, the maximum duration "gap" g_L associated with potential insertion location L is $s_{\hat{i},L+1} - c_{\hat{i},L-1}$; the latest possible start time of the job in the next position minus the earliest possible completion time of the job in the previous position.

Step 3 Pick the insertion location $\hat{k}$ in the schedule of machine $\hat{i}$ and the amount of job $\hat{j}$ to assign there as follows. First, the insertion location $\hat{k}$ is determined by finding the potential insertion location having the largest gap; that is, $\hat{k} = \arg\min_{l=1,\ldots,\beta} g_l$. Next, the amount of time the machine should spend working on this job, D^*, is computed using:

$$D^* = \min \left\{ \rho_{\hat{i}\hat{j}} v_{\hat{j}},\ g_{\hat{k}},\ (1-\alpha)(e_{\hat{i}} - b_{\hat{i}}) - \sum_{\omega=1}^{\beta} (c_{\hat{i}\omega} - s_{\hat{i}\omega}) \right\}. \tag{2.20}$$

Note that for job $\hat{j}$ with unassigned duration $v_{\hat{j}}$, the amount of time required for machine $\hat{i}$ to complete all of what remains of $\hat{j}$ is $\rho_{\hat{i}\hat{j}} v_{\hat{j}}$. Thus, Equation 3.9 compares the remaining duration of the task, the gap of time available, and the remaining shift availability of the machine. If the remaining duration of the task is the smallest value of the three, then all of what remains of $\hat{j}$ will be completed by machine $\hat{i}$. Otherwise, some portion of $\hat{j}$ will remain to be assigned to a machine in a future iteration.

After the initial solution is generated, a local search improvement algorithm, described in Algorithm 2, is applied to schedule S. The swap neighborhood is explored in this algorithm. Specifically, a swap neighbor of schedule S is any schedule that can be reached by swapping job p in the y^{th} position on machine Γ with job q in the z^{th} position on machine Λ. More precisely, the exact amounts of standard durations of p and q assigned to their current machines will be swapped to the other machine. We refer to this swap using the notation $swap(p, q)$ from this point forward.

To determine whether $swap(p, q)$ is feasible, the function **ViolateConstraint()** is used. The function checks conditions such as whether machines Γ and Λ are eligible for jobs q and p, respectively, and whether their machine schedules are compatible with the release and due dates of jobs q and p. Specifically, the following conditions are required in order for the swap to be feasible:

- $t_{\Gamma q} = t_{\Lambda p} = 1$ (machines eligible for jobs)
- $r_p < e_\Lambda$ and $r_q < e_\Gamma$ (release dates compatible with shift end times)
- $d_p > b_\Lambda$ and $d_q > b_\Gamma$ (due dates compatible with shift begin times)

If these preliminary feasibility conditions are met, methods such those contained in **AddSchedule()** are used to determine whether "gaps" of sufficient width can be created within the machine schedules for swapping the jobs. Specifically, steps 1 and 2 of **AddSchedule()** are applied for position y in the schedule of machine Γ and position z in the schedule of machine Λ to move preceding start times as early as possible and subsequent start times as late as possible. This determines the resulting gap widths g_y and g_z. Gap width g_y must be long enough for job q to be moved there, and g_z must be long enough for job p to be

Algorithm 2 Local Search Improvement Algorithm

Require: An initial solution S

Machines $i = 1, \ldots, |\mathcal{M}|$ and jobs $j = 1, \ldots, |\mathcal{N}|$

hasImprovement = True

while hasImprovement = True **do**

  hasImprovement = False

  for $\Gamma = 1$ to $|\mathcal{M}|$

    for $\Lambda = \Gamma + 1$ to $|\mathcal{M}|$

      for $p = 1$ to β_Γ (the number of positions in the schedule of Γ)

        for $q = 1$ to β_Λ (the number of positions in the schedule of Λ)

          if ! **ViolateConstraint**(Γ, p, Λ, q)

            hasImprovement = **FindImprovement**(Γ, p, Λ, q)

            if hasImprovement = True, implement $swap(p, q)$ in S and **Go to** 2

        end for

      end for

    end for

  end for

end while

return S

moved there. Let $c_{\Gamma y} - s_{\Gamma y}$ denote the duration of job p in its original position y of machine Γ's schedule and $c_{\Lambda z} - s_{\Lambda z}$ denote the duration of job q in its original position z of machine Λ's schedule. Then, the following conditions are required for swap feasibility:

- $(c_{\Gamma y} - s_{\Gamma y})\frac{\rho_{\Lambda p}}{\rho_{\Gamma p}} \leq g_z$,
- $(c_{\Lambda z} - s_{\Lambda z})\frac{\rho_{\Gamma q}}{\rho_{\Lambda q}} \leq g_y$.

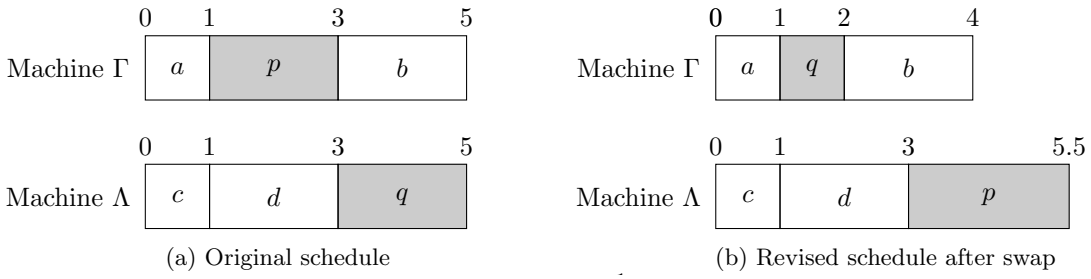
Once $swap(p, q)$ has been determined to be feasible, whether or not it improves the solution is considered. A swap only improves the solution if it reduces the total amount of time required to complete the work. Specifically, if Equation (2.21) is satisfied, it is faster to complete jobs p and q on machines Λ and Γ, respectively, than vice versa:

$$(c_{\Gamma y} - s_{\Gamma y})\frac{\rho_{\Lambda p}}{\rho_{\Gamma p}} + (c_{\Lambda z} - s_{\Lambda z})\frac{\rho_{\Gamma q}}{\rho_{\Lambda q}} \leq (c_{\Gamma y} - s_{\Gamma y}) + (c_{\Lambda z} - s_{\Lambda z}). \tag{2.21}$$

The right hand side of Equation (2.21) provides the total duration of jobs p and q in the original schedule, with job p in position y on machine Γ and job q in position z on machine Λ. The left hand side adjusts those durations according to the machine productivity ratios for jobs p and q. For example, if the duration of job p was 2 hours on machine Γ and $\rho_{\Gamma p} = 0.5$ and $\rho_{\Lambda p} = 1$, then its duration on machine Λ will be 4 hours $(2 * \frac{1}{0.5})$. If Equation (2.21) is satisfied, then the swap reduces the total duration of work. The swap will be implemented and the algorithm will return to line 2 to search for additional swaps.

Figure 2.4 illustrates a swap using a very simple example. The original schedule is depicted in (a). Job p is the second job on machine Γ and lasts from time 1 to 3. Job q is the third job on machine Λ and lasts from time 3 to 5. Assume the following productivity values: $\rho_{\Gamma,p} = 1$, $\rho_{\Lambda,p} = 1.25$, $\rho_{\Gamma,q} = 0.5$, $\rho_{\Lambda,q} = 1$. Figure (b) depicts the schedule that will

result if $swap(p, q)$ is implemented. The duration of q will be reduced from 2 to 1 while the duration of job p will be increased from 2 to 2.5. The reduction in q is greater than the increase in p so the swap is improving and will be implemented, reducing the total schedule duration by 0.5 hours.



(a) Original schedule

(b) Revised schedule after swap

Figure 2.4: Example of a swap between jobs p and q

2.6 Experiments and Results

The performance of the proposed heuristic is investigated using the experimental design summarized in Table 2.6. Five different factors are included in the design. Numerical values and distributions used for each level of each factor are based on real data from a company with a large network of warehouses. The first factor, warehouse type, has three levels representing small, medium, and large warehouses, which differ according to the number of employees (machines, $|\mathcal{M}|$), tasks (jobs, $|\mathcal{N}|$), and task duration (U). A larger warehouse implies more employees and more tasks. Arena Input Analyzer was used to fit task duration distributions for more than 1200 historical tasks from 15 work days. The distributions fit the historical data well and the p-values from the Kolmogorov-Smirnov test are greater than 0.05. The second factor, machine schedule, has two levels representing two different employee work shift options. In the first level, all employees work between 8:00 am and

4:00 pm. We refer to this as a "tight" machine schedule because there is no flexibility for employees to complete work later in the day. Alternatively, in the "loose" schedule, 30% of employees work this schedule, and the remaining 70% work between 12:00 pm and 8:00 pm. The third factor describes task release times, which can either be "early" with all tasks available at 8:00 am or "late" with 50% of tasks available at 8:00 am, 20% at 10:00 am and 30% at 11:00 am. Analogously, the task due date factor can either be "late" with no tasks due until 8:00 pm or "early" with 30% due at 6:00 pm, 20% due at 7:00 pm, and the rest due at 8:00 pm. Therefore, the test instances with the tighest windows for task completion are those having late release date with early due date. Furthermore, the machine eligibility factor describes the proportion τ of machines (employees) eligible to perform each task, on average. When $\tau = 0.5$, the expected proportion of employees eligible for each task is 50%, therefore eligibilities are less restrictive. Alternatively, eligibilities are more restrictive when $\tau = 0.3$. Finally, the distribution of machine productivities is a single-level factor that is derived using Arena Input Analyzer. Historical data from 15 work days were used to fit the distribution, which is normal with a mean of 0.9 hours and a variance of 0.1 hours. The p-value from the Kolmogorov-Smirnov test is greater than 0.05.

Table 2.6: Experimental Design

Factors	Level	Level Description
Warehouse Type	3	**(1) Small:** $\lvert\mathcal{M}\rvert = 15, \lvert\mathcal{N}\rvert = 85$, U$\sim Weibull(0.516, 0.809)$, $E(\text{U}) = 1.69$ **(2) Medium:** $\lvert\mathcal{M}\rvert = 35, \lvert\mathcal{N}\rvert = 130$, U$\sim 12 \cdot Beta(0.773, 3.47)$, $E(\text{U}) = 2.19$ **(3) Large:** $\lvert\mathcal{M}\rvert = 95, \lvert\mathcal{N}\rvert = 190$, U$\sim Gamma(9.36, 0.687)$, $E(\text{U}) = 6.43$
Machine Schedule	2	**(1) Tight:** 100% employees work shift $[8, 16]$ **(2) Loose:** 30% employees work shift $[8, 16]$, 70% employees work shift $[12, 20]$
Ready Time	2	**(1) Early:** 100% tasks available at 8 **(2) Late:** 50% tasks available at 8, 20% at 10, 30% at 11
Due Date	2	**(1) Late:** 100% tasks due at 20 **(2) Early:** 50% tasks du at 20, 20% at 19 and 30% at 18
Machine Eligibility	2	**(1) Less Restrictive:** $\tau = 0.5$ **(2) More Restrictive:** $\tau = 0.3$

For each of the 48 factor-level combinations, 10 random replicates are generated, yielding a total of 480 test instances. Each test instance is solved once using the heuristic described in Section 4 and once using CPLEX with a 60 minute runtime limit imposed. All computations are performed on a super computer with 12 CPUs and 24-GB RAM.

Table 2.7: Experimental Results

Factors	CPLEX		Heuristic		
	Results	RunTime(s)	Results	Gap to CPLEX	RunTime(s)
(1,1,1,1,1)	46.6	31.6	47.1	1.03%	4.6
(1,1,1,1,2)	49.0	20.1	49.4	0.67%	5.0
(1,1,1,2,1)	51.7	46.7	52.3	1.14%	4.2
(1,1,1,2,2)	42.1	22.5	42.5	0.88%	5.1
(1,1,2,1,1)	51.2	165.9	51.9	1.30%	4.0
(1,1,2,1,2)	45.1	107.5	45.6	1.18%	4.5
(1,1,2,2,1)	53.9	180.2	55.8	3.36%	5.3
(1,1,2,2,2)	50.0	155.0	50.6	1.22%	4.8
(1,2,1,1,1)	43.4	38.3	44.4	1.84%	5.1
(1,2,1,1,2)	50.4	20.0	50.9	0.98%	4.9
(1,2,1,2,1)	48.6	40.9	49.1	1.10%	4.3
(1,2,1,2,2)	53.2	19.9	53.8	1.23%	4.8
(1,2,2,1,1)	46.8	158.9	47.7	1.71%	4.0
(1,2,2,1,2)	48.9	53.3	49.5	1.11%	5.3
(1,2,2,2,1)	46.8	169.7	47.5	1.32%	4.7
(1,2,2,2,2)	49.6	40.6	50.2	1.24%	5.2
(2,1,1,1,1)	86.6	314.3	87.9	1.40%	9.8
(2,1,1,1,2)	99.1	124.4	100.5	1.41%	11.8
(2,1,1,2,1)	85.6	308.0	86.9	1.46%	11.8
(2,1,1,2,2)	100.1	180.4	101.3	1.27%	11.7
(2,1,2,1,1)	87.9	326.5	89.2	1.48%	11.5
(2,1,2,1,2)	94.9	344.1	96.1	1.26%	12.2
(2,1,2,2,1)	91.2	358.4	92.0	0.90%	10.7
(2,1,2,2,2)	97.2	283.9	98.2	1.01%	11.0
(2,2,1,1,1)	84.7	291.4	86.2	1.76%	12.0
(2,2,1,1,2)	97.1	149.0	98.8	1.69%	10.8
(2,2,1,2,1)	88.2	293.7	89.9	1.91%	11.4
(2,2,1,2,2)	97.9	210.9	99.8	1.90%	10.8
(2,2,2,1,1)	86.7	305.6	88.3	1.89%	11.7
(2,2,2,1,2)	100.0	300.3	101.9	1.90%	12.0
(2,2,2,2,1)	84.3	405.4	85.7	1.57%	11.0
(2,2,2,2,2)	93.2	402.5	94.7	1.70%	11.9
(3,1,1,1,1)	580.0	1077.9	600.1	3.47%	14.5
(3,1,1,1,2)	595.6	355.7	616.8	3.56%	14.7
(3,1,1,2,1)	582.2	1078.6	602.3	3.45%	13.8
(3,1,1,2,2)	593.8	361.3	613.8	3.37%	14.7
(3,1,2,1,1)	584.5	1078.0	605.0	3.52%	14.1
(3,1,2,1,2)	594.5	360.1	614.8	3.41%	13.3
(3,1,2,2,1)	*581.4	2595.6	601.8	3.84%	15.2
(3,1,2,2,2)	592.9	360.5	613.6	3.49%	12.9
(3,2,1,1,1)	582.3	1076.6	602.1	3.39%	13.5
(3,2,1,1,2)	592.5	359.5	613.3	3.50%	14.4
(3,2,1,2,1)	579.8	1078.2	600.3	3.53%	15.3
(3,2,1,2,2)	596.5	361.0	618.1	3.62%	14.3
(3,2,2,1,1)	576.4	1075.4	597.2	3.61%	14.4
(3,2,2,1,2)	592.7	358.4	613.2	3.46%	14.2
(3,2,2,2,1)	579.0	1078.1	598.9	3.44%	14.9
(3,2,2,2,2)	594.7	360.0	616.2	3.61%	15.6

The full set of computational results are given in Table 2.7. The first column denotes

the factor levels used to create the instance. For example, instance (1,1,1,1,1) in the first row is for a small warehouse with a tight machine schedule, early release dates, late due dates, and less restrictive machine eligibility. Remaining columns report the objective values of the best solutions obtained by CPLEX and the heuristic, the percent gap between the heuristic solution and the best lower bound from CPLEX, and the runtime for each. Note that each value reported is an average across 10 replicates of the indicated instance type. The heuristic performance gaps range from 0.67% to 3.62%. The average runtime of CPLEX per instance is 393.1 seconds while average runtime is 10.1 seconds per instance for the heuristic. One row of the table, corresponding to instance type (3,1,2,2,1), has an asterisk next to the CPLEX solution value to denote that optimal solutions were not obtained within a 60 minute runtime limit for some of the 10 replicates of this instance type. For all other replicates of all other instances, optimal solutions were obtained. Therefore, the values in the Gap to CPLEX column are optimality gaps for every row except (3,1,2,2,1); in that case, the gap is to the best lower bound obtained by CPLEX.

Table 2.8 provides additional details regarding the instances that CPLEX failed to solve to optimality within a 60 minute runtime limit. The first column provides the replicate number. It can be observed that optimal solutions were only found for 4 of the 10 replicates. However, for those instances that CPLEX did not solve to optimality, the optimality gaps are at most 1.01%. These instances represent large warehouses with tight machine schedules, late release dates, early due dates, and less restrictive machine eligibilities. Therefore, with the exception of the machine eligibility factor, they are the most constrained instances overall.

Average, minimum, and maximum heuristic performance gaps are provided per instance class in Table 2.9. An *instance class* refers to a set of instances for which one of the

Table 2.8: Experimental Results for Instance Class (3,1,2,2,1)

Factors	CPLEX Result	Lower Bound	Gap	CPLEX RunTime(s)
(3,1,2,2,1-1)	580.64	580.64	0.00%	1084.1
(3,1,2,2,1-2)	583.25	582.08	0.20%	3600.0
(3,1,2,2,1-3)	583.45	583.45	0.00%	1081.4
(3,1,2,2,1-4)	583.09	583.09	0.01%	3600.0
(3,1,2,2,1-5)	568.88	566.60	0.40%	3600.0
(3,1,2,2,1-6)	587.65	587.65	0.00%	1076.5
(3,1,2,2,1-7)	578.47	575.00	0.60%	3600.0
(3,1,2,2,1-8)	580.40	580.40	0.00%	1076.6
(3,1,2,2,1-9)	589.49	584.19	0.91%	3600.0
(3,1,2,2,1-10)	578.71	572.92	1.01%	3600.0

factors in the experimental design is fixed at a particular level. For example, the first row (1,*,*,*,*) provides results averaged across all instances where the warehouse type is fixed to level 1 (small). The performance gap is computed as the difference between the heuristic solution the best solution obtained by CPLEX within a 60 minute runtime limit. Recall from the previous paragraph that the best solution obtained by CPLEX is also an optimal solution for 474 out of 480 test instances. Using Table 10, it can be observed that the heuristic performance degrades as warehouse size increases, with the average gap increasing from 1.45% in instances with small warehouses to 3.52% in instances with large warehouses. However, the average gap remains relatively stable at approximately 2% across all other experimental design factors. Thus, we can conclude that warehouse type has a larger impact than other factors on heuristic performance.

Average, minimum, and maximum computation times for both CPLEX and the heuristic are provided in Table 2.10 by instance class. We can observe a drastic increase

Table 2.9: Heuristic Performance Gaps (% deviation from best known lower bound) per Instance Class

Factors		Avg. Gap	Max Gap	Min Gap
Warehouse	(1,*,*,*,*)	1.45	4.78	0.10
Types	(2,*,*,*,*)	1.53	3.40	0.10
	(3,*,*,*,*)	3.52	6.52	0.10
Machine	(*,1,*,*,*)	2.12	4.78	0.10
Schedule	(*,2,*,*,*)	2.21	6.52	0.10
Ready	(*,*,1,*,*)	2.06	6.52	0.10
Times	(*,*,2,*,*)	2.27	4.78	0.10
Cutoff	(*,*,*,1,*)	2.10	6.52	0.10
Times	(*,*,*,2,*)	2.23	4.78	0.10
Eligibility	(*,*,*,*,1)	2.31	4.78	0.10
	(*,*,*,*,2)	2.03	6.52	0.10
Overall	(*,*,*,*,*)	2.17	5.11	0.10

in CPLEX average solution time as warehouse size increases, from an average of 79.4 seconds in instances with small warehouses to 286.5 seconds in instances with large warehouses. We can also observe a maximum CPLEX solution time of 3600 seconds for instance class (3,*,*,*,*), indicating the 60-minute runtime limit was reached without finding an optimal solution for some of the instances within this class. The heuristic computation times also increase as warehouse size increases, from an average of 4.7 seconds to 14.4 seconds. This is reasonable, as both CPLEX and the heuristic must address a larger number of variables as warehouse size increases. Furthermore, CPLEX needs approximately twice as much time to solve instances in class (*,*,*,*,1) than in class (*,*,*,*,2). For two problems with the same warehouse type, machine schedule, release dates and due dates but different eligibilities, the one with less restrictive eligibility has a larger feasible region. The heuristic, on the other hand, still shows no significant difference between these two classes of problems.

In Table 2.11, computation times are compared for instance classes describing three factors at a time instead of one. For example, class (*,1,2,2,*) represents all instances for which the machine schedule is tight (at level 1), release dates are late (level 2) and due

Table 2.10: Computation Time for Experiments

Factors		CPLEX Solving Time (s)			Heuristic Solving Time (s)		
		Avg.	Max	Min	Avg.	Max	Min
Warehouse Types	(1,*,*,*,*)	79.4	286.4	13.6	4.7	8.1	1.2
	(2,*,*,*,*)	286.5	493.1	95.5	11.4	17.8	5.5
	(3,*,*,*,*)	813.4	3600.0	301.4	14.4	19.4	9.8
Machine Schedule	(*,1,*,*,*)	426.5	3600.0	13.6	10.1	18.8	1.3
	(*,2,*,*,*)	359.7	1084.9	15.4	10.3	19.4	1.2
Ready Times	(*,*,1,*,*)	326.9	1089.0	13.6	10.1	18.8	1.2
	(*,*,2,*,*)	459.3	3600.0	18.4	10.2	19.4	1.3
Cutoff Times	(*,*,*,1,*)	353.0	1089.0	13.6	10.1	18.8	2.3
	(*,*,*,2,*)	433.6	3600.0	15.4	10.3	19.4	1.2
Eligibility	(*,*,*,*,1)	566.7	3600.0	25.8	10.1	18.4	1.2
	(*,*,*,*,2)	221.0	493.1	13.6	10.3	19.4	1.3
Overall	(*,*,*,*,*)	393.1	3600.0	13.6	10.2	19.4	1.2

dates are early (level 2). It can be observed that instances in this class require the most time to solve optimally, with an average of 655 seconds. This is approximately 200 seconds more than other groups. However, the heuristic computation time and performance are not impacted by this group of factor levels.

Table 2.11: Computation Time for Experiments

Groups	Avg. CPLEX (s)	Avg. Heuristic (s)	Avg. Solution Gap
(*,1,1,1,*)	320.7	10.1	1.9%
(*,1,1,2,*)	332.9	10.2	1.9%
(*,1,2,1,*)	397.0	9.9	2.0%
(*,1,2,2,*)	655.6	10.0	2.3%
(*,2,1,1,*)	319.6	10.1	2.2%
(*,2,1,2,*)	334.8	10.1	2.2%
(*,2,2,1,*)	375.3	10.3	2.3%
(*,2,2,2,*)	409.4	10.5	2.1%

2.7 Conclusions

In this chapter, we introduced an unrelated parallel machine scheduling problem with release dates, due dates, limited machine availabilities and job splitting for some jobs. To the best of our knowledge, these problem characteristics have not been simultaneously considered in the literature. We also introduced a new objective function to minimize the total job processing time on all machines, which is different than minimizing makespan or tardiness. In order to solve the problem variant where all jobs can be split, both a constructive heuristic with local search improvement and a commercial optimization solver are used.

The commercial solver found optimal solutions for 474 out of 480 test instances within a 60 minute runtime limit. The six instances for which an optimal solution was not identified all fall within the most constrained class of instances, with tight machine schedules, late release dates and early due dates. The heuristic provides solutions within an average gap of 2.17% to the best solutions obtained by CPLEX. Though heuristic performance degrades as warehouse size increases, it is not impacted by how tightly constrained an instance is. Overall, the heuristic is shown to be fast and effective.

One limitation of our research is the estimation of employee performance. This work assumes that employees, like machines, have non-degrading performance throughout the day. This is unlikely to be true, as factors such as fatigue are likely to influence productivity levels. Furthermore, the performance vector values used in our computational study represent a three-week average of historical data. This potentially masks impacts such as employee learning curves for new tasks. However, our modeling of employee productivity mirrors actual practice at the company DC and is therefore deemed sufficient for the purposes of

this application.

Future work could include addressing additional practical considerations for this problem. For example, employee lunch breaks (machine downtime) could be pre-scheduled instead of just reserving $\alpha\%$ of the schedule. This will split each employee schedule into two parts. In that case, not only the shift begin/end times, but also the break begin/end times need to be considered when assigning tasks. Furthermore, a transfer time between two tasks could be considered. Employees potentially need to make preparations (for example, walk to the area, pick up tools) before shifting work to a different type of task. This would require modeling and solving a problem that includes setup times.

3 Scheduling Unrelated Parallel Machines Subject to Setup Times and Scheduled Machine Availability

3.1 Introduction

Similar to Chapter 2, this chapter is motivated by a large medical product supplier seeking to optimize labor costs within their warehouses. The unrelated parallel machine scheduling problem from Chapter 2 is extended to account for two additional practical considerations faced by the company. First, a delay is introduced to model the preparatory work that occurs when warehouse staff shift from one type of task to another. The medical product supplier groups warehouse tasks based on type, such as order picking, pallet picking, sorting and replenishment. Switching from a task in one group to another may require specialized equipment. For example, a lift truck is required for pallet picking but not for order picking. The warehouse staff will require time to retrieve the necessary equipment. Switching from a task in one group to another may also require traveling to a different part of the warehouse. Therefore, delays are introduced between tasks from different groups but not between tasks within the same group.

The second extension in this chapter is the explicit consideration of employee breaks. The problem in Chapter 2 reserved $\alpha\%$ of an employee's schedule for breaks, effectively reducing each employee's effective shift duration. Alternatively in this chapter, the break time is introduced as a fixed time window during an employee's schedule during which they are unavailable to complete tasks. For example, an employee that works from 8:00 am to 5:00 pm may take a lunch break from 12:00 pm to 1:00 pm. This employee would not be available

to complete tasks during the lunch break. This concept appears in the machine scheduling literature as limited machine availability [18]. Instead of machines operating continuously, their availability is described by a set of time intervals. In this problem, preemption is allowed. If an employee does not have enough time to complete an entire task before their break, the task can be resumed by the employee after break, or by a different employee at any time. We also assume that an employee will encounter no setup time for their first task after break. The problem is to minimize the labor required to complete all work within the warehouse, including setup times. Thus, the sum of all task processing times plus setup times is minimized.

The contributions of this chapter are as follows. We formally introduce a new problem to the scheduling literature; its primary novelties are its unique objective function and its simultaneous consideration of release dates, due dates, setup times and limited machine availabilities in an unrelated parallel machine scheduling environment. Next we provide a MIP formulation for it. Then, a simulated annealing (SA) metaheuristic is developed for its solution. A computational study compares the performance of the SA approach to the MIP via commercial optimization software. The study is comprised of 480 test instances that vary according to the number of machines, tasks, timing of release and due dates, and machine productivity and availability.

The remainder of this chapter is organized as follows. In the next section, we review related literature. In Section 3.3, a formal problem statement is provided and a mixed integer programming formulation (MIP) is introduced. The details of the simulated annealing metaheuristic used to solve the problem are presented in Section 3.4. Section 3.5 describes a computational study used to compare the developed heuristic with a commercial optimization

solver. Finally, results and directions for future work are provided in Section 3.6.

3.2 Literature Review

Chapter 2 provides a review of the unrelated parallel machine scheduling literature, with emphasis on papers that address due dates, release dates, limited machine availability and job splitting. In this chapter, we include additional information about setup times and limited machine availability in the context of unrelated parallel machine scheduling problems. We also discuss the use of metaheuristics for solving parallel machine scheduling problems.

Allahverdi (2015) provides a recent review of scheduling problems with setup times [19]. Combined with two other surveys, Allahverdi et al. (1999) and Allahverdi et al. (2008), over 500 papers on this topic from 1960 to 2014 are reviewed [20, 21]. Here we focus on those papers from the reviews that address unrelated parallel machine scheduling problems with setup times. That is, we include papers on $R|ST_{sd}|\gamma$ problems, where R denotes unrelated parallel machines and ST_{sd} indicates sequence-dependent setup times, the latter as denoted in Allahverdi et al. (2015) [19]. Pereira Lopes and de Carvalho (2007) propose a branch-and-price algorithm for a $R|ST_{sd}, a_k, r_j|\sum w_j T_j$ problem, where a_k indicates that machines have limited availabilities, r_j indicates the presence of job release dates and $\sum w_j T_j$ denotes a total weighted tardiness objective [22]. Lin and Hsieh (2014) solve a related $R|ST_{sd}, r_j|\sum w_j T_j$ problem with the same weighted tardiness objective; the difference is their problem does not include limited machine availabilities [23]. To solve the problem they propose an iterated hybrid metaheuristic which integrates an electromagnetism-like algorithm (EMA) with local search. Chen (2009) discusses a $R|ST_{sd}, d_j|\sum T_j$ problem with a tardiness objective and job due dates d_j [9]. A SA algorithm and two additional heuristics

are presented for its solution. A few additional papers in the literature address unrelated parallel machine scheduling problems with setup times and total completion time ($\sum C_j$) objectives. For example, Joo (2015) uses a hybrid genetic algorithm with three dispatching rules to solve a $R|ST_{sd}, r_j|\sum C_j$ problem that additionally has machine-dependent processing times [24]. Avalos (2015) and Wang (2016) also study a $R|ST_{sd}, r_j|\sum C_j$ problem [25, 26]. The former proposes a metaheuristic based on a multi-start algorithm with variable neighborhood descent while the latter uses a hybrid algorithm with iterated greedy search. The $R|ST_{sd}, a_k, r_j|\sum C_j$ problem studied by Afazalirad (2016) additionally includes machine eligibility and precedence constraints. Both a genetic algorithm (GA) and artificial immune system (AIS) are proposed for finding optimal or near optimal solutions [27]. Finally, Yilmaz (2014) studies a $R|ST_{sd}|C_{max}$ problem with a makespan objective and proposes a genetic algorithm (GA) with local search for its solution [28].

Like scheduling problems with setup times, those with machine availability constraints have also been discussed for decades. Schmidt (2002) reviews deterministic scheduling problems where machines are not continuously available for processing [18]. The paper also provides an update for the machine availability portion of the three-field notation to describe the availability pattern of machines, which may be constant, zigzag, decreasing, increasing, staircase and arbitrary. Ma (2010) provides another survey of deterministic machine scheduling problems with machine availability constraints, with an emphasis on discussing algorithmic advances and complexity results [29]. There are a few papers in the literature that address machine availability specifically in the context of parallel machine scheduling environments. Logendran *et al.* (2004) solves a $R|a_k|\sum w_jT_j$ problem with machine availability constraints a_k and a weighted tardiness objective [16]. This problem has an arbi-

trary machine availability pattern, as the machine availabilities are generated from a Poisson distribution. Hu (2010) uses a heuristic to solve a $R|a_k|C_{max}$ problem with precedence constraints and a makespan objective, motivated by shipyard operations [30]. This problem also has an arbitrary machine availability pattern. Zhao (2011) solves a $P|a_k|\sum w_j C_j$ problem where machines will not be available in a specified time period in identical parallel machines. This problem has randomly generated availability pattern [31]. Hashemian (2014) considers an identical parallel machines scheduling problem minimizing the makespan with multiple planned nonavailability periods in the case of resumable jobs ($P|a_k|C_{max}$). The machine availability pattern is zigzag, as each machine has a single unavailable period starting at time 0 for odd index and time 15 for even index machines [32].

Table 3.1 summarizes the papers reviewed above and indicates the objectives and problem characteristics included in each paper. Of the papers included, two are most similar to our work, as release dates, setup times and machine availabilities are considered simultaneously in the context of unrelated parallel machine scheduling in both Pereira Lopes and de Carvalho (2007) [22] and Afazalirad (2016) [27]. Our work includes four distinguishing features: (i) due dates, (ii) job splitting, (iii) machine-task eligibility restrictions, and (iv) a unique objective function which minimizes the total time required to complete all work in the warehouse (i.e. total labor hours).

Simulated annealing (SA) is a metaheuristic that has shown promise for parallel machine scheduling problems. Here we review a number of papers that apply SA to such problems. Koulamas (1997) uses SA in an identical parallel machine scheduling problem [33]. His SA is used to exchange jobs assigned to machines using a polynomial decomposition heuristic. Kim (2002) uses SA to solve an unrelated parallel machine problem

Table 3.1: Summary of reviewed unrelated parallel machine scheduling papers with setup time times and machine availability

Authors	$\boldsymbol{R}$	ST_{sd}	a_k	**Objective**	**Other**
Pereira Lopes and de Carvalho (2007)	✓	✓	✓	$\sum w_j T_j$	r_j
Chen (2009)	✓	✓		$\sum T_j$	d_j
Lin and Hsieh (2014)	✓	✓		$\sum w_j T_j$	
Joo (2015)	✓	✓		$\sum C$	
Yilmaz (2014)	✓	✓		C_{max}	
Avalos (2015)	✓	✓		C_{max}	r_j
Afazalirad (2016)	✓	✓		$\sum C_j$	r_j, eligibility
Wang (2016)	✓	✓		$\sum C_j$	r_j
Logendran *et al.* (2004)	✓		✓	$\sum w_j T_j$	
Hu (2010)	✓		✓	C_{max}	
Zhao (2011)			✓	$\sum w_j C_j$	
Hashemian (2014)			✓	C_{max}	

minimizing total tardiness with sequence-dependent setup times [34]. Neighborhood solutions are generated using six job or item rearranging techniques in the proposed SA. Lee (2006) studies an identical parallel machine scheduling problem minimizing the makespan [35]. The SA he proposed generates near-optimal solution. Damodaran (2012) solves another identical parallel machine scheduling problem minimizing the makespan [36] involving arbitrary job processing times, non-identical job sizes, and non-zero ready times. His developed SA is less computationally costly and the solution quality is comparable to the greedy

randomized adaptive search procedure (GRASP) [37]. Lin (2015) considers a multi-objective multi-point simulated annealing algorithm to solve an unrelated parallel machine scheduling problem minimizing makespan, total weighted completion time and total weighted tardiness simultaneously [38]. Our SA considers historical performance when selecting operators for improvement. Operators that are frequently successful in finding improving solutions are more likely to be selected.

3.3 Problem Description

There are a set of $\mathcal{N}$ jobs that must be scheduled. Each job $j \in \mathcal{N}$ can begin no earlier than its release date r_j and must be finished before its due date d_j. The standard duration of job j on a machine with baseline productivity is known and is denoted u_j. There is a set $\mathcal{M}$ of unrelated parallel machines available for processing jobs. The availability of each machine $i \in \mathcal{M}$ is described by a shift start time b_i and shift end time e_i; a machine cannot process jobs before b_i or after e_i. Each machine also has scheduled downtime from f_i to h_i, where $[f_i, h_i]$ is an interval of time within $[b_i, e_i]$. The parameter t_{ij} describes whether machine i is eligible to process job j. For every job j for which machine i is available, the productivity ρ_{ij} of machine i for job j is known. If $\rho_{ij} < 1$, machine i has faster than baseline productivity for job j; if $\rho_{ij} > 1$, machine i has slower than baseline productivity for job j. Therefore, the time required for machine i to complete all of job j is $\rho_{ij}u_j$. However, it should be noted that job splitting is allowed and multiple machines can work simultaneously on portions of the same job. Some pairs of jobs require a setup time if they appear in consecutive positions in a machine's schedule. This is indicated by the binary parameter p_{qr}, which takes value 1 if a setup is required between jobs q and r and 0 otherwise. The duration

of the setup time is a fixed value P. We assume that setup times are never required before the job in the first position of a machine's schedule, nor before the first job in the machine's schedule following its break. Let X_{ij} represent the fraction of job j assigned to machine i. Then, the time machine i will spend on job j is $\rho_{ij}u_jX_{ij}$. The problem is to assign each job to an eligible machine, such that job release date and due date constraints and machine availability constraints are not violated. The objective is to minimize the total amount of time required to process all jobs, including required setup times.

Using the $\alpha|\beta|\gamma$ classification scheme for scheduling problems, this is classified as follows. In the α field, there is Rm for m unrelated parallel machines and NC_{win}, which refers to the machines having an arbitrary availability pattern (i.e., not increasing, decreasing, zigzag, etc. as defined in [18]). In the β field, there is r_j and d_j for release dates and due dates, $pmtn$ to indicate the schedule is preemptive and ST to indicate setup times as in [21]. In the γ field, which indicates the performance criterion, the objective function of our problem does not fit with the predefined options in the literature. Letting Y_{ik} be a binary variable indicating whether a setup is incurred for the job in position k of machine schedule i, the objective is to minimize $\sum_{ijk}\rho_{ij}u_jX_{ijk} + \sum_{ik}Y_{ij}P$.

Here we include a simple example to illustrate the impact of setup times on the total duration of a schedule. For simplicity, we assume a single machine is available and is eligible to process jobs 1-3. Setup time is required between jobs 1 and 3 and 2 and 3, but not between jobs 1 and 2. That is, $p_{13} = p_{23} = 1$ and $p_{12} = 0$. Figure 3.1 shows two possible machine schedules for these three jobs. The job sequence in schedule 1 requires only 1 setup time while the sequence in schedule 2 requires 2 setup times; therefore schedule 2 is P time units longer than schedule 1 and schedule 1 is preferred.

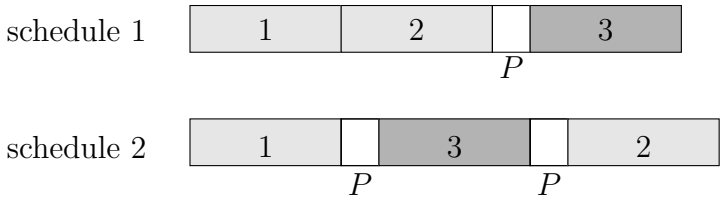


Figure 3.1: Different job sequences when considering job preparation times

A second example is included here to demonstrate the impact of the machine schedule (fixed employee break), and also the interrelationship between breaks and setup times. There is a single machine available to process two jobs, $j = 1$ and $j = 2$. A setup is required between jobs 1 and 2, i.e., $p_{12} = 1$. However, recall that setup times are not required for the first job following a machine's break. In schedule 1, the start time of job 1 is such that it cannot be finished prior to the scheduled break. It is preempted, and the machine resumes working on job 1 after the break. Then, there is a setup time before job 2 begins. In schedule 2, the start time of job 1 is such that it can be finished prior to the break. After break, the machine can immediately begin working on job 2 with no setup time. Thus, the total duration of schedule 1 is P units longer than schedule 2, so schedule 2 is preferred.

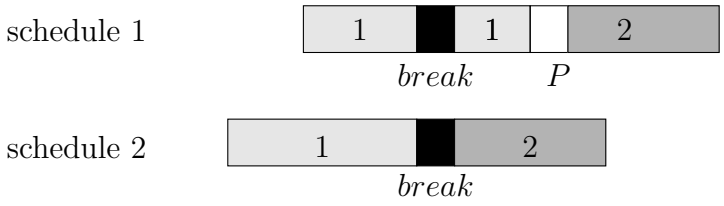


Figure 3.2: Different job sequences when considering fixed breaks

Before presenting the MIP for this problem, we first discuss two methods for handling the fixed break time within the model. The first is to introduce m new jobs to be scheduled; one for each machine (i.e. employee). The only machine eligible to complete the

job corresponding to its break is itself. That is, if job $n+1$ is introduced to model the break for machine 1, then the eligibility parameter $t_{1,n+1} = 1$ and $t_{i,n+1} = 0$ for all $i \neq 1$. Furthermore, the release date of a machine's break "job" is set to the beginning of the break interval and the due date is set to the end of the break interval. That is, for break job $n+1$ for machine 1, $r_{n+1} = f_1$ and $d_{n+1} = h_1$. Finally, the binary parameters indicating whether setup times are required between a break job and other jobs are set to zero ($p_{n+1,j} = 0 \; \forall \; j$). This method requires the introduction of $|\mathcal{M}|$ new jobs and is illustrated in the second row of Figure 3.3. The second method is to split one machine into two: one representing the portion of availability before the break, and the other representing the remaining availability after break. To accomplish this, machine i with shift $[b_i, e_i]$ and break $[f_i, h_i]$ is replaced with two machines i' and i''. The schedule for machine i' becomes $[b_i, f_i]$ and for machine i'' becomes $[h_i, e_i]$. Machines i' and i'' are both eligible for every job for which i is eligible, with the same productivity as i. Neither machine i' nor i'' will require a scheduled break. This method requires the introduction of $|\mathcal{M}|$ new machines and is illustrated in the third row of Figure 3.3. We choose the former method because it results in fewer added decision variables. Decision variables $\mathbf{X}$ and $\mathbf{Z}$ are indexed across jobs, machines and positions but $\mathbf{Y}$ is indexed only across machines and positions.

The MIP formulation for the problem in Chapter 2 can be modified for the problem in this chapter. Recall that X_{ijk} indicates the fraction of job j assigned to machine i in position k, Z_{ijk} is a binary variable indicating whether or not any portion of job j is assigned to machine i in position k, and Y_{ik} is a binary variable indicating whether a setup time is required for the job in position k of machine i's schedule. Instead of providing the full MIP model here, much of which would duplicate what is already given in Chapter 2, we discuss

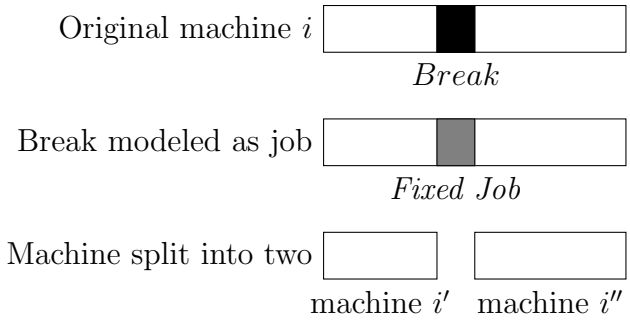


Figure 3.3: Two Methods for Introducing the Fixed Break Time

only the changes required to the MIP. We define the set $\mathcal{N}'$ to indicate the expanded job set that now includes the fixed "break job" for each machine. Any set that is indexed over $\mathcal{N}$ in the MIP in chapter 2 should be indexed over $\mathcal{N}'$ in the new model. Constraint set (2.10) is not needed in the new model and should be removed. The remaining changes are as follows. Any constraint sets that are not mentioned are left unchanged. The objective function below replaces objective 2.1:

$$\min \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{N}'} \sum_{k \in \mathcal{K}} \rho_{ij} u_j X_{ijk} + \sum_{i \in \mathcal{M}} \sum_{k \in \mathcal{K}} Y_{ik} P, \tag{3.1}$$

where P is the constant setup time. This objective minimizes the total time required to process all jobs, including any required setups. Constraint (3.2) replaces constraint set (2.7) to ensure job j in the k^{th} position on machine i begins no earlier than the time the previous job ended, plus a setup time, if necessary:

$$s_{ik} \geq c_{i(k-1)} + Y_{ik} P, \ \forall\ i \in \mathcal{M}, k \in \mathcal{K}\ (k \neq 1). \tag{3.2}$$

Constraint set (2.8) and constraint set (2.9) are set remained the same. Constraint set (3.3) is added to ensure a penalty is considered when jobs between position k and $k-1$

require penalty ($k > 1$). Where $p_{j,h}$ in this constraint is a binary parameter shows whether there is a setup time between job j and h. If there requires a setup time, then $p_{j,h} = 1$.

$$Y_{ik} \geq p_{jh} \left(Z_{ih(k-1)} + Z_{ijk} - 1 \right), \ \forall \ i \in \mathcal{M}, j, h \in \mathcal{N}, k \in \mathcal{N}(k \neq 1) \tag{3.3}$$

Constraint (3.4) is introduced to ensure that no setup time occurs before the first job of each machine:

$$Y_{i1} = 0, \forall \ i \in \mathcal{M}. \tag{3.4}$$

Constraint set (3.5) is used to ensure positions in a machine's schedule are filled consecutively and no positions are skipped. For example, for a schedule containing 3 jobs, this constraint would not allow the jobs to appear in positions 1, 3 and 5, but instead would require them to be placed in positions 1, 2 and 3:

$$\sum_{j \in \mathcal{N}'} Z_{ijk} \geq \sum_{j \in \mathcal{N}'} Z_{ij(k+1)}, \ \forall \ i \in \mathcal{M}, k \in \mathcal{K}. \tag{3.5}$$

Finally, constraint set (3.6) enforces binary restrictions on the $\mathbf{Y}$ variables:

$$Y_{ik} \in \{0, 1\} \ \forall \ i \in \mathcal{M}, k \in \mathcal{K}. \tag{3.6}$$

A simple illustrative example including 3 jobs and 2 machines is presented. Table 3.2 provides the job details. Each job has a release date, due date, and a standard duration. For example, job 1 is released at 8:00, is due at 14:00 and requires 5.00 standard hours. There is a setup between jobs 1 and 2 and between jobs 2 and 3 but not between jobs 1 and 3. The value of setup time, when required, is $P = 0.25$ hours in this example. Table 3.3 gives the

machine details. The productivity information indicates how fast (or slow) each machine is for each job. For example, machine A requires 1.1 times the standard duration of job 1 to process job 1. Missing productivity values indicate the machine is not eligible for the job. For example, machine B is not eligible for job 2. The machine availability information indicates the shift begin and end time for each machine. For example, machine A is available between 8:00 and 17:00. The break time indicates jobs cannot be performed during that time. For example, machine A has a break from 12:00 to 13:00.

				Setup Times p_{jl}		
j	r_j	c_j	u_j	$l=1$	$l=2$	$l=3$
1	8.00	14.00	5.00	-	1	0
2	10.00	20.00	4.00	1	-	1
3	12.00	20.00	5.00	0	1	-

Table 3.2: Example Job Information

i	Productivity(ρ_{ij})			Machine Availability		Break Time	
	$j=1$	$j=2$	$j=3$	b_i	e_i	f_i	h_i
A	1.1	1.0	0.9	8:00	17:00	12:00	13:00
B	0.8	–	1.2	12:00	21:00	17:00	18:00

Table 3.3: Example Machine Information

The optimal solution is described in Table 3.4 and depicted in Figure 3.4. It is obtained by solving the MIP presented above. Job 1 is split into 2 equal pieces between machines A and B. However, machine A spends more time than machine B completing its assigned portion of job 1 because it is less productive. Job 2 is performed by machine A since it is the only eligible machine. A setup time of 15 minutes occurs between jobs 1 and 2 on machine A. Also, job 2 is preempted from 12:00 to 13:00 during the break of machine A. Job

3 is split between machines A and B as well. There is a setup time before job 3 on machine A because job 2 precedes it. There is no setup time before job 3 on machine B because job 1 precedes it. Job 3 is preempted from 17:00 to 18:00 during the break of machine B. The total job processing time and setup time associated with this solution is 15 hours.

	Job 1	**Job 2**	**Job 3**
Machine A	0.5	1	0.17
Machine B	0.5	0	0.83

Table 3.4: Fraction of Each Job Assigned to Each Machine in Optimal Solution

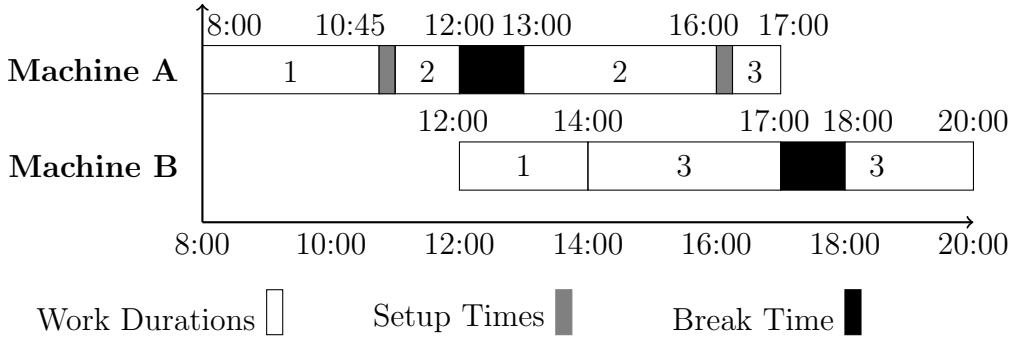


Figure 3.4: Example: The Optimal Schedule Considering Setup Times

3.4 Methods

The Simulated Annealing (SA) metaheuristic was first introduced in Kirkpatrick *et al.* (1983) to solve combinatorial optimization problems [39]. The algorithm iteratively moves from current feasible solutions to generated neighborhood solutions until stopping criteria are met. Moves to neighborhood solutions with improved objective values are always allowed, while moves to non-improving solutions are only allowed according to a probability that decreases as the algorithm progresses. Section 3.4.1 describes a method for obtaining

an initial solution, which is required by SA. Section 3.4.2 describes the move operators used to explore solution neighborhoods, and Section 3.4.3 presents the details of the SA metaheuristic, as implemented in this research.

3.4.1 Initial Solution Generation

To construct an initial solution to pass as input to SA, the greedy constructive heuristic from Chapter 2 is updated to account for the limited machine availabilities and setup times present in this variant of the problem. In the updated algorithm, functions **FindJobs()** and **FindMachine()** remain unchanged from their descriptions in Chapter 2. They are used to find a job $\hat{j}$ to insert and to identify a machine $\hat{i}$ to explore for feasible insertion locations for job $\hat{j}$. The function **AddSchedule()** is used to find an insertion location for $\hat{j}$ in the machine schedule of $\hat{i}$ and decide the appropriate amount of $\hat{j}$ to place in the selected position in the schedule of machine $\hat{i}$. The primary updates to **AddSchedule()** required in this chapter to Steps 1-3 are presented below.

Step 1 Assume positions $1, 2, \ldots \beta$ in the schedule of machine $\hat{i}$ have been filled. Then, there are $\beta + 1$ possible insertion locations for job $\hat{j}$: one preceding each of the β positions, and one following the β^{th} position. Choosing any one of these locations will result in a modified alternative schedule for machine $\hat{i}$. The alternative schedule results when job start times are shifted, without changing the job sequence, so that maximum duration "gaps" are created between jobs in two consecutive positions. This frees up as much time as possible for inserting a new job. Therefore, when a new job is inserted into a machine schedule, not only will a new job fill a new position in the schedule, but the start and completion times of other jobs already in the schedule may change.

Figure 3.5 illustrates this idea. Suppose there is a machine for which two tasks are currently assigned in the initial schedule: a and b, in that order. No setup time is required between a and b, but setup times are required between the new job and both a and b. Alternative 1 is to place the new job before a. Moving jobs a and b to begin as late as possible without violating the due date of either generates a large "gap" of available time prior to job a, but a setup time (indicated in black) must occupy part of that time. Alternative 2 is to place the new job between a and b. Job a cannot be moved any earlier due to its release date and the schedule begin time, but job b can be moved as late as possible without violating its due date. This generates a large window of time between job a and b, but setup times must be introduced twice in this gap: after a and before b. The final alternative is to place the new job after b. Ideally, both jobs a and b would be moved to begin as early as possible without violating the release date of either. In this example, there is actually no room to move the start times of a or b to be earlier. However, there is still a window of time between the completion of b and the end of the machine's schedule. A setup time will occupy part of the window.

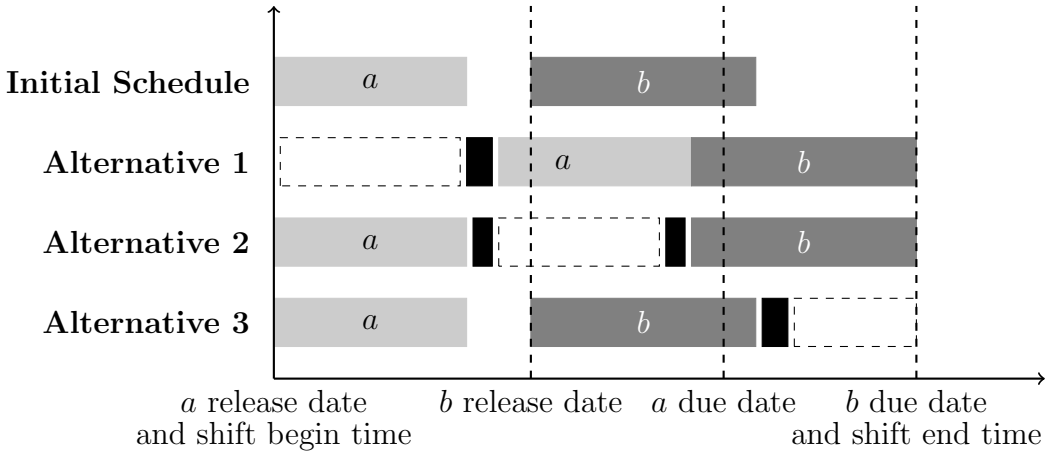


Figure 3.5: Illustration of Alternative Schedules and Insertion Locations

To create the maximum duration "gap" associated with potential insertion location L on machine $\hat{i}$, equations 2.16 and 2.17 in Chapter 2 describe how to update modified completion times and start times for all jobs in positions that precede L. In this chapter, the break is treated as a job with a fixed start and completion time; it cannot be modified. Therefore, when inspecting jobs that precede L, it must be considered whether L comes before of after the break. If L comes before the break, then start and completion times for jobs in positions $1, 2, \ldots, L-1$ must be inspected for modification. However, if L comes after the break, then only positions $\tau, \tau+1, \ldots, L-1$ can be inspected for modification, where τ indicates the first position after the fixed break. Then, an additional update to Step 1 is required for the inclusion of setup times when finding the earliest possible start times for jobs. Equation 2.17 must be replaced with:

$$\sigma_l = \max\{\gamma_{l-1} + p_{j_l, j_{l-1}} P, r_{j_l}\}. \tag{3.7}$$

This sets the earliest possible start time for the job in position l to the maximum of its release date and the modified completion time of the job in the previous position, plus a setup time if required.

Equations 2.18 and 2.19 in Chapter 2 describe how to update modified start and completion times for all jobs in positions that come after L, up through the last occupied position β in the machine's schedule, to begin and complete as late as possible. Again, it must be considered whether L is before or after the fixed break. If it is after the fixed break, positions $L+1, L+2, \ldots, \beta$ are inspected, whereas only positions $L+1, L+2, \ldots, \tau-1$ are inspected if it is before the break. Then, Equation 2.19 must

be replaced with:

$$\gamma_l = \min\{\sigma_{l+1} - p_{j_l,j_{l+1}}P, d_{j_l}\}. \tag{3.8}$$

This sets the latest possible completion time for the job in position l to the minimum of its due date and the modified start time of the job in the next position, minus a setup time if required.

Step 2 Update the starting times $s_{\hat{i}l}$ and completion times $c_{\hat{i}l}$ for all $l = 1, 2, \ldots, \beta$ except $l = L$ and $l = \tau$ to the times σ_l and γ_l identified above. Then, the maximum duration "gap" g_L associated with potential insertion location L is $s_{\hat{i},L+1} - c_{\hat{i},L-1} - p_{j_{L-1},\hat{j}}P - p_{\hat{j},j_{L+1}}P$; the latest possible start time of the job in the next position minus the earliest possible completion time of the job in the previous position, minus any setup times that are required (between the job in position $L-1$ and job $\hat{j}$, or between job $\hat{j}$ and the job in position $L+1$, or both).

Step 3 Step 3 uses information from Steps 1 and 2 to pick the insertion location $\hat{k}$ for job $\hat{j}$ in the schedule of machine $\hat{i}$ and then determine how much time the machine should spend working on job $\hat{j}$, denoted D^*. In this chapter, no changes are required to how the location $\hat{k}$ is selected. The feasible location with maximum duration gap is still chosen. However, D^* is computed differently than in Chapter 2. Equation 2.20 limits D^* according to the amount of job $\hat{j}$ that still needs to be processed, the gap duration associated with position $\hat{k}$, and the total amount of work in the schedule of machine $\hat{i}$, which cannot exceed $(1-\alpha)\%$ of its shift duration. The fixed breaks in this chapter

replace the need to limit a machine's work hours to $(1-\alpha)\%$ of its shift duration, so that term is dropped and Equation 2.20 is replaced with:

$$D^* = \min\left\{\rho_{\hat{i}\hat{j}} v_{\hat{j}},\ g_{\hat{k}}\right\}. \tag{3.9}$$

The amount of job $\hat{j}$ to assign to position $\hat{k}$ of machine $\hat{i}$ is the minimum of the remaining duration of $\hat{j}$ to be assigned and the maximum gap length for position $\hat{k}$.

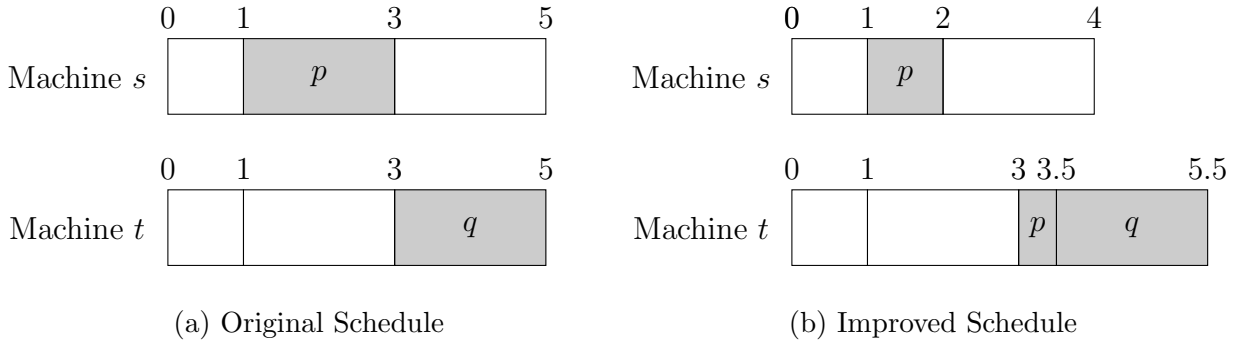
3.4.2 Neighborhood Solution Generation

Neighborhood solutions are generated via three different operators in our algorithm. The first operator is job insertion. It searches for improving solutions that involve two different machines. Specifically, it attempts to move a job on one machine (or a portion of it) to another machine. Figure 3.6 depicts an example of this operation. Let p be a job on on machine s and q be a job on machine t. Machine t is twice as productive as machine s for job p, so some of job p is moved from machine s to machine t. Specifically, 1 hour of job p is moved from s to t, and machine t only requires 0.5 hours to process this amount of job p. The total duration of work on machines s and t improves from 10 hours to 9.5 hours with this move. We refer to this operator as **Insert(p,q,t)**, which indicates job p is moved to the position before q in machine t. Let $D_s(p)$ represent the duration of job p on machine s and denote the position before task q in the schedule of machine t as l. Let γ_l be the latest possible completion time of p if inserted in position l, and let σ_l be the earliest possible start time of p. The productivities of machines s and t for p are ρ_{ps} and ρ_{pt}, respectively. Then the amount of p that will be moved from s to the l^{th} position of machine t ($D_s(\Delta p)$)

is determined by:

$$D_s(\Delta p) = \min\left\{D_s(p), (\gamma_l - \sigma_l)\,\frac{\rho_{ps}}{\rho_{pt}}\right\}. \tag{3.10}$$

This equation indicates that if the available space in position l of machine t is large enough to insert all of job p there, then p will be moved to machine t in its entirety. Otherwise, the amount of p that will be moved to t is the difference between the latest time p can be completed and the earliest time it can be started, adjusted for machine productivities.

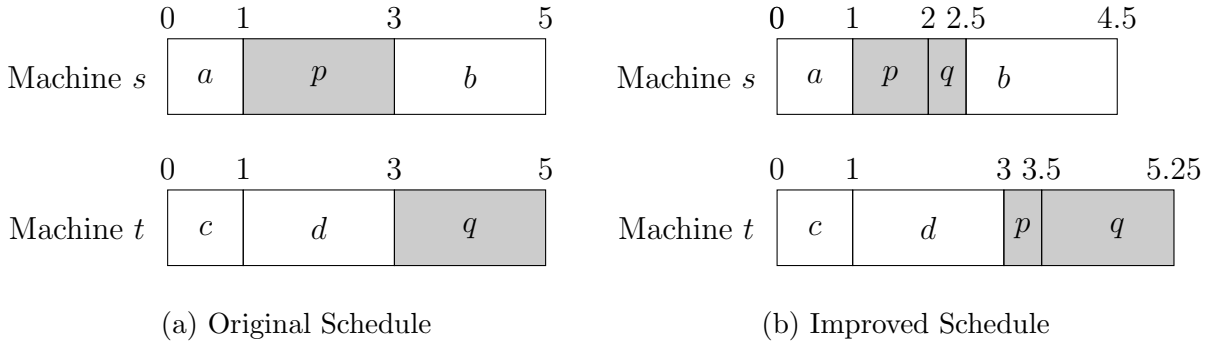


(a) Original Schedule (b) Improved Schedule

Figure 3.6: Example of job insert

The second operator is job interchange. It searches for two jobs from two different machines and attempts to swap them (or portions of them). Figure 3.7 depicts an example of this operation. Let p be a job on machine s and q be a job on machine t. Machine t is twice as productive as machine s for both jobs p and q. Moving 1 hour of job p from s to t adds 0.5 hours of work to t. Moving 0.25 hours of job q from t to s adds 0.5 hours of work to machine s. Therefore, the net decrease in total work across both schedules is 0.25 hours. This operator is denoted **Interchange(p,s,q,t)**, which indicates job p from machine s and job q from machine t are entirely or partially interchanged. In this operator, if a job is partially removed from a machine schedule, then the positions immediately preceding it

and following it will be considered for inserting the new job. For example, in Figure 3.7, a portion of p is removed from machine s. Some of p remains in s so the positions before and after what remains of p will be considered as potential insertion locations for job q. If a job is fully removed from a machine schedule, then the other job involved in the interchange will be considered for insertion only in the position vacated by the job that was removed. Two temporary schedules are generated to decide how much of jobs p and q will be interchanged. Temporary schedule 1 is generated by first removing q from t and then inserting as much of p as possible into the position vacated by q. Note that, if only some of p was moved to t, then some of p will remain in s. Next, as much of q as possible is inserted to s. If some portion of q remains, it is inserted back to q. If this is not possible, the temporary schedule is considered infeasible. Temporary schedule 2 is generated by following the converse process of that used to generate temporary schedule 1. That is, it begins with the entire removal of p from s. If both temporary schedules are feasible, the one with lesser total duration is returned. If only one is feasible, it is returned. Otherwise if neither are feasible, the interchange returns the original schedule with no change.



(a) Original Schedule

(b) Improved Schedule

Figure 3.7: Example of job interchange

The last operator is job combination. It searches for improving solutions that involve only a single machine. Specifically, it inspects a machine schedule and determines whether it contains the same job in more than one position. If so, it attempts to combine them into a single position (with longer duration) without violating any constraints. This also has the potential to reduce setup time. See the two alternative schedules for machine s in Figure 3.8 for example. In the original schedule, job b appears in the schedule twice and there is a setup time between b and d. In the revised schedule, the two portions of job b have been combined, and the setup time between b and d has been eliminated because they are no longer adjacent in the schedule. We refer to this operator as **Combine(s)** moving forward, where the function is applied on a machine s.

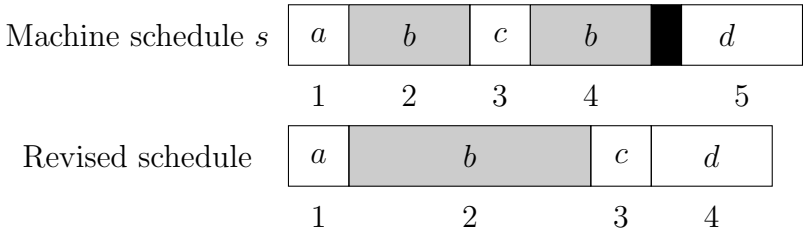


Figure 3.8: Example of a job combination

3.4.3 Neighborhood Exploration

The following steps describe how the neighborhood of a current solution is explored via the move operators defined in the previous section. The functions **SelectOperator()**, **StopCriteria()**, **AcceptWorseSolution()** that are referred to in these steps vary according to which stage the SA is in at a particular point in time. These details are explained in Section 3.4.4.

Let R be a list of all job-machine pairs that appear in the current schedule, sorted in

order from the worst productivity to the best productivity of the job-machine pairs. Define λ as an inspection range parameter and let λ=0. Mark all job-machine pairs in R as unavailable for inspection. Let i be the iteration number and set $i = 1$, and let $U(s)$ denote the duration of a machine schedule s.

1. Mark the job-machine pairs (p, s) in the top $10\lambda\%$ to $10(\lambda + 1)\%$ of list R as available for inspection.

2. Randomly select a job-machine pair (p, s) from R that is available for inspection.

    a. If no job-machine pair can be selected (no pair is available in R), increase the inspection range by letting $\lambda = \lambda + 1$.

        i. If $\lambda <= 4$, go back to Step 1.

        ii. Else, go to Step 3.

    b. Else if job-machine pair (p, s) has been selected, construct a list Q_p of machines that are eligible for job p, sorted in order from the best productivity for p to the worst. Let Q'_p be a sorted list that only contains the top 20% of Q_p. Mark all machines in Q'_p as available for inspection.

    c. Select the machine available for inspection that is highest in list Q'_p.

        i. If no machine can be selected (all machines have been marked as unavailable), mark this job-machine pair (p, s) as unavailable. Return to Step 2.

        ii. Else if a machine has been successfully selected from Q'_p, denote it t and mark it as unavailable for future inspection. Use **SelectOperator()** to select an operator.

iii. Search machine t for a feasible move (of the selected type) for job p.

- If the selected operator is **Interchange()**, inspect the jobs in machine schedule t in sequential order for the first job q that can feasibly be interchanged with pair (p, s).
- If no feasible move is found with the selected operator, select the other operator and return to 2.c.iii.
- If no feasible move is found with either operator, go to Step 2.
- If a feasible move is found for p in t, use the selected operator to generate temporary schedules s' and t'.
- If $U(s) + U(t) - U(s') - U(t') > 0$ (this is a feasible and improving move) then implement the move by replacing s with s' and t with t'. Then call **Combine(s)** and **Combine(t)**. Update "Successful Operator Counter" and go to Step 3.
- Else if $U(s) + U(t) - U(s') - U(t') \leq 0$ and **AcceptWorseSolution()** is true (this is a feasible non-improving move but the SA criterion indicates it should be accepted), then implement the move by replacing s with s' and t with t'. Then call **Combine(s)** and **Combine(t)**. Update "Successful Operator Counter" and go to Step 3.
- Else do not implement the move. Retain schedules s and t. Go to Step (2.c).

3 Let $i = i + 1$. Call **StopCriteria()** (this can potentially result in terminating the SA or lowering the temperature). Regenerate sorted list R and let $\lambda = 0$. Mark all

job-machine pairs in R as unavailable for inspection. Return to Step 1.

Note that in the above, Combine() is applied anytime an insert or interchange move is implemented. Consider first an insert move. After an insert move, a job is removed (entirely or partially) from machine s and inserted into machine t. A combine move can be required for either of these machines. For example, by removing some or all of job p, machine schedule s becomes less "tight". A job combination that was previously infeasible due to the schedule's "tightness" might be feasible now. Figure 3.9 depicts this situation. For machine schedule s with job sequence $b - a - b$, job b is split because the release date of a is later than the release date of job b and a also has an earlier due date than b. Let neighborhood schedule s' be the schedule generated after half of job a is removed from s. Revised schedule s' can now accommodate a combination for job b, where its two portions become adjacent. Consider now machine schedule t, into which p is being inserted. A pre-processing step is used within the combine operator to inspect whether schedule t contains any jobs that appear in more than one position. This requires traversing the job array (go through every job in the schedule and use a count array to record) with a complexity of $O(n)$, where n is the number of jobs in the schedule. If the pre-processing step reveals repeating jobs in t, **Combine(t)** will seek the opportunity to improve t. Otherwise, **Combine(t)** will stop. For an interchange, both machines involved in the move experience the removal of some or all of one job and the insertion of some or all of another. Therefore, Combine() is required for both s and t after an interchange.

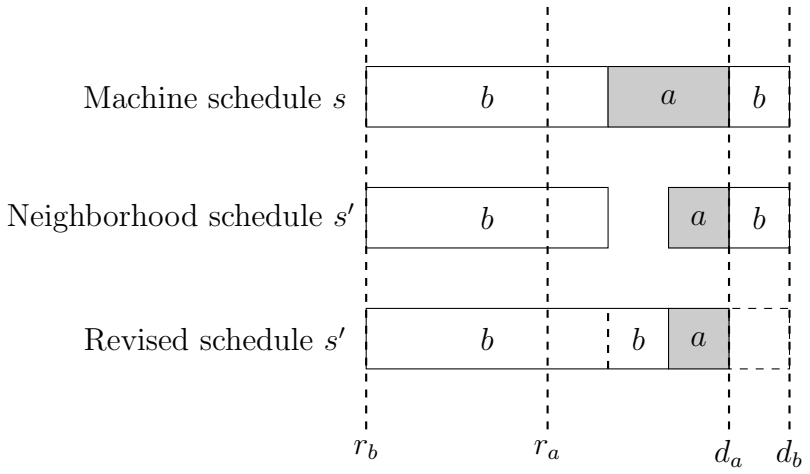


Figure 3.9: Example of **Combine(s')** when a job portion is removed from a schedule s'

3.4.4 Simulated Annealing Process and Parameters

There are a total of two stages in the SA: a warm-up stage and a simulated annealing stage. The warm-up stage randomly selects move operators for consideration and only implements feasible moves that improve the current solution. The simulated annealing stage randomly selects between the two operators according to a probability that is based on their historical performance and uses the simulated annealing criterion to determine whether to accept a feasible move that does not improve the current solution. Figure 3.10 the overall flow of the SA.

The warm-up stage begins with the initial solution generated using the process outlined in Section 3.4.1. Then in each iteration, **SelectOperator()** randomly selects either insert or interchange with equal probability. If the first selected operator fails to find improvement, the other operator will be selected and tried. An operator counter "Successful Operator Counter" is used to record the number of times each operator finds an improving solution. In this stage, **AcceptWorseSolution()** is always set to false to avoid implement-

ing a non-improving move. The **StopCriteria()** function will terminate the SA during this stage if no improving solutions are found in 1000 consecutive iterations or if $\lambda > 4$.

The simulated annealing stage begins with the final solution from the warm-up stage. Instead of selecting between the move operators with equal probability, **SelectOperator()** uses historical performance of the operators to influence their selection. Specifically, the probability of picking an operator is the ratio of the success counter for that operator to the total success counter for both operators. If the first selected operator fails to find a feasible move, the other operator is subsequently selected. Updates to "Successful Operator Counter" continue during this stage anytime an improving solution is found.

The initial temperature T_0 for the simulated annealing stage is 100. The probability of accepting a non-improving solution is:

$$p = e^{-\Delta \cdot \frac{100}{T}}, \tag{3.11}$$

where Δ is the difference between the new and old objective values. This probability p is used in **AcceptWorseSolution()** to decide if a non-improving solution will be accepted. For example, in the initial temperature stage, if the difference between a current solution and a candidate solution is 0.5 hours, then the probability of accepting the (worse) candidate solution is $p = e^{-0.5} = 0.60$. This probability decreases as the temperature stages progress.

In the literature, epoch length is often proportional to the number of neighborhood solutions. For example, in Kim et al. (2002), epoch length (L)= number of jobs (N) $\times$ number of machines (M) $\times$ beta (β) [34]. In this research we experiment with $\beta \in \{.25, 0.50, 0.75, 1.00, 1.25\}$ for each instance and use the value for each instance that provides the best performance. The cooling down ratio T_k in this algorithm is calculated using a

geometric ratio:

$$T_k = \alpha T_{k-1}, k = 0, 1, 2, \ldots, 0 < \alpha < 1, \tag{3.12}$$

where k is the temperature stage number and α is the cooling down ratio parameter. The value of k is increased by 1 whenever the temperature drops. We have selected $\alpha = 0.9$ after tests and experiments, which encourages slow convergence. Let i' record the number of iterations in each temperature stage. The **StopCriteria()** in the simulated annealing stage includes two parts: (i) If $\lambda > 4$ or $i' \geq L$, move to the next temperature stage and set $i' = 0$; (ii) If no improvement is found in this temperature stage or if $T < 1$, STOP.

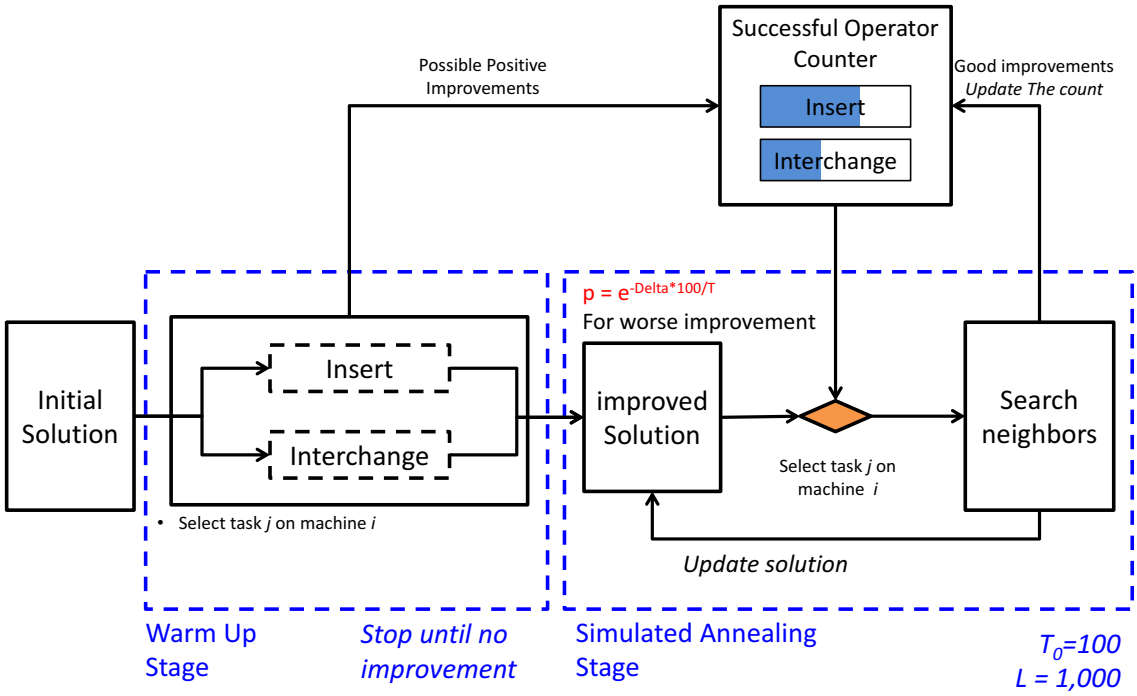


Figure 3.10: Illustration of the simulated annealing algorithm

3.5 Performance Evaluation

The performance of the proposed SA is investigated in two primary ways. First, it is employed with some minor modifications to solve the problem instances from Chapter 2. Recall that the problem variant in Chapter 2 did not have setup times or limited machine availabilities. The SA is compared to a commercial optimization software and the heuristic introduced in Chapter 2 on the basis of the quality of solutions produced for the instances from Chapter 2. Second, a new set of instances including setup times and fixed machine break times is developed for the problem defined in this chapter. The performance of the SA heuristic and a commercial optimization software are compared for this new set of instances.

Figure 3.5 provides a summary of the experimental design used in Chapter 2. It consists of five factors, including warehouse size, machine schedule, task ready time, task due date and machine eligibility. Because these instances do not include setup times or fixed machine breaks, changes to the methodology presented in this chapter are required before it can be used to solve them. Specifically, P from Equation 3.1 is set to zero as no setup time will be considered, and the parameter ρ_{ij}, which indicates if a setup is required between consecutive jobs (i, j), is set to 0 for all job pairs. The maximum shift duration is set to 6.8 hours so that no employee will be busy more than 85% of their entire shift. Furthermore, in function **AddSchedule()**, the consideration of fixed break position τ is ignored, so that all positions between 1 and L will be considered in the search for the "gap" of longest duration.

The computational results are given in Table 3.6. The first column denotes the factor levels used to create the instance. The next two columns present the percent gap between Heur-1 (the heuristic from Chapter 2) and Heur-2 (the SA heuristic from the current chapter)

Table 3.5: Experimental Design from Chapter 2

Factors	Level	Level Description
Warehouse Type	3	**(1) Small:** $\|\mathcal{M}\| = 15, \|\mathcal{N}\| = 85$, U$\sim Weibull(0.516, 0.809)$, $E(\text{U}) = 1.69$ **(2) Medium:** $\|\mathcal{M}\| = 35, \|\mathcal{N}\| = 130$, U$\sim 12 \cdot Beta(0.773, 3.47)$, $E(\text{U}) = 2.19$ **(3) Large:** $\|\mathcal{M}\| = 95, \|\mathcal{N}\| = 190$, U$\sim Gamma(9.36, 0.687)$, $E(\text{U}) = 6.43$
Machine Schedule	2	**(1) Tight:** 100% employees work shift $[8, 16]$ **(2) Loose:** 30% employees work shift $[8, 16]$, 70% employees work shift $[12, 20]$
Ready Time	2	**(1) Early:** 100% tasks available at 8 **(2) Late:** 50% tasks available at 8, 20% at 10, 30% at 11
Due Date	2	**(1) Late:** 100% tasks due at 20 **(2) Early:** 50% tasks du at 20, 20% at 19 and 30% at 18
Machine Eligibility	2	**(1) Less Restrictive:** $\tau = 0.5$ **(2) More Restrictive:** $\tau = 0.3$

and the best lower bound (LB) obtained from CPLEX with a 30 minute runtime limit. Note that CPLEX was able to find optimal solutions for all but six of the instances in this experimental design; therefore the LB is equivalent to the optimal solution in most cases (details are available in Table 2.8). The fourth column presents the percent gap between Heur-1 and Heur-2 and the fifth provides the runtimes of the two heuristics. The results indicate that Heur-2 (the SA) produces slightly better solutions than Heur-1 in most of the groups, but with much longer runtimes. The solutions from Heur-2 are 0.11% to 1.58% better, on average, than from Heur-1. But, the average runtime for Heur-2 is approximately 10 minutes compared with 1 to 5 seconds for Heur-2. To examine the significance of the performance difference between Heur-1 and Heur-2, two-sided paired t-tests were conducted for each instance group (the sample size for each group is 10). The null hypothesis for each test is that the mean objective values produced by Heur-1 are equal to the mean objective

values produced by Heur-2 for the instance group in question. At a significance level of 0.05, the null hypothesis is rejected for all of the medium and large instance groups, indicating a significant difference in performance between the two heuristics. For the 16 small instance groups, the null hypothesis is rejected for 9 of them (indicating a significant difference in performance) and accepted for 7 (indicating no significant difference in performance).

Table 3.6: Results comparison for instances from Chapter 2

Factors	**Average Percent Difference Between Methods**			**Run Time (s)**	
	Heur-1,LB	Heur-2,LB	Heur-1,Heur-2	Heur-1	Heur-2
(1,*,*,*,*)	1.11%	0.43%	1.58%	1.43	322.66
(2,*,*,*,*)	0.87%	0.62%	0.40%	1.82	625.16
(3,*,*,*,*)			1.28%	4.90	868.79
(*,1,*,*,*)	0.75%	0.50%	0.51%	2.81	605.80
(*,2,*,*,*)	0.98%	0.79%	0.24%	2.62	605.28
(*,*,1,*,*)	0.83%	0.79%	0.05%	2.88	606.13
(*,*,2,*,*)	0.90%	0.50%	0.80%	2.55	604.94
(*,*,*,1,*)	0.90%	0.81%	0.11%	2.82	606.36
(*,*,*,2,*)	0.83%	0.48%	0.72%	2.61	604.71
(*,*,*,*,1)	0.93%	0.47%	0.97%	2.74	605.49
(*,*,*,*,2)	0.80%	0.72%	0.12%	2.69	605.58

A new set of experiments including setup times and fixed machine breaks time is developed to investigate the performance of the SA for the problem variant introduced in this chapter. The experiments are based on 20 days of historical data from 5 warehouses of the partner company. The problems in this chapter are more realistic and complex, according to the company. For example, jobs are released throughout a 24-hour period, compared with a 16 hour period (from 08:00 to 24:00) in the previous chapter. Additionally, employee (machine) schedules are more flexible and diverse. The experimental design is summarized in Table 3.8. Seven factors are included. Warehouse type has three levels representing small, medium, and large warehouses, which differ according to the number of employees

(machines, M) and tasks (jobs, N). A larger warehouse implies more employees and more tasks. The sizes selected are representative of real warehouses in the partner company's network. Machine schedule has two levels representing employee shift alternatives. Details of the shift alternatives are provided in Table 3.7. For example, following the tight schedule alternative in a small warehouse, 25% of the employees work a shift from 00:00 to 08:00, 60% work from 08:00 to 16:00, and the remaining 15% work from 16:00 to 24:00. These are denoted "tight" because the shift changes do not contain any overlapping time. The break always occurs during the fifth hour of the schedule (e.g., 12:00 to 13:00 for a shift from 08:00 to 16:00). The third factor, ready time, describes the job release time. A distribution for ready time was developed using Arena Input Analyzer. The p-value from the Kolmogorov-Smirnov test for this distribution is greater than 0.05. Task due dates are decided by task ready times plus time window durations, which are captured by the the fourth factor. It has two levels: "normal" and "tight", for which time windows are shorter, on average, when the level is "tight". Machine eligibility is the fifth factor and describes the proportion τ of machines (employees) eligible to perform each task, on average. When $\tau = 0.3$, the expected proportion of employees eligible for each task is 30%. The sixth factor, setup time, describes the proportion p of task pairs that will require a setup time. When $p = 0.75$, the expected proportion of task pairs requiring a setup time is 75%. A distribution for the final factor, task duration, was determined using Arena Input Analyzer. The p-value from the Kolmogorov-Smirnov test is greater than 0.05 for this distribution.

For each of the 48 factor-level combinations, 10 random replicates are generated, yielding a total of 480 test instances. Each test instance is solved once using the SA and once using CPLEX with a 60 minute runtime limit imposed. All computations are performed

Table 3.7: Machine Schedule Design Detail

Warehouse Type	Tight Schedule	Loose Schedule
Small	25%: 0:00 - 8:00 60%: 8:00 - 16:00 15%: 16:00 - 24:00	25%: 0:00 - 8:00 45%: 8:00 - 16:00 15%: 12:00 - 20:00 15%: 16:00 - 24:00
Medium	80%: 8:00 - 16:00 20%: 16:00 - 24:00	20%: 4:00 - 12:00 40%: 12:00 - 20:00 20%: 16:00 - 24:00 5%: 20:00 - 24:00
Large	5%: 0:00 - 8:00 30%: 8:00 - 16:00 65%: 16:00 - 24:00	5%: 0:00 - 8:00; 5%: 4:00 - 12:00, 20%: 8:00 - 16:00; 15%: 12:00 - 20:00 40%: 16:00 - 24:00; 15%: 16:00 - 24:00

Table 3.8: Experimental Design

Factors	Level	Level Description
Warehouse Type	3	**(1) Small:** $M = 30, N = 80$ **(2) Medium:** $M = 50, N = 130$ **(3) Large:** $M = 70, N = 190$
Machine Schedule	2	**(1) Tight:** Tight schedule from Table 3.7 **(2) Loose:** Loose schedule from Table 3.7
Ready Time	1	10% Uniform Distribution (0,4) 90% 4.5+19*Beta(2.16,2.23)
Time Windows	2	**(1) Normal:** 40% 4 hours, 50% 12 hours, 10% 22 hours **(2) Tight:** 60% 4 hours, 30% 12 hours, 10% 22 hours
Machine Eligibility	2	**(1) Less Restrictive:** $\tau = 0.3$ **(2) More Restrictive:** $\tau = 0.15$
Setup Time	2	**(1) More Restrictive:** $p = 0.75$ **(2) Less Restrictive:** $p = 0.5$
Task Duration	1	Weib(129,0.729)

on a super computer with 12 CPUs and 24-GB RAM. Computational results for the instances with small warehouses are given in Table 3.9. The first column denotes the factor levels used to create the instance. The second column provides the average value for K (number of positions on each machine) used for the instances in each set. The value of K is determined via two passes through the binary search algorithm [40]. The first pass through the binary search algorithm establishes a range of values for K for which CPLEX does not terminate with an out-of-memory result before producing a feasible solution. The second pass finds the minimum value of K, within the established range, required for the instance to solve optimally. The third column provides the number of instances per set for which CPLEX is successful in finding a feasible solution. The next four columns provide the objective values, runtimes, percentage gaps to the best lower bounds, and the best lower bounds, all from CPLEX. The third column represents the number of solutions that CPLEX successfully find feasible solutions in this model. The next four columns present the CPLEX objective values, runtimes, percent gaps to the best lower bounds from CPLEX, and the lower bounds values from CPLEX. The final three columns present the analogous information for the SA. CPLEX successfully solved all 10 instances for small instances. Each value reported in the small instances is an average across 10 replicates of the indicated instance type. On average, SA produces solutions that are close to the ones that produced by CPLEX. The average runtime of CPLEX across all instances is around 3600 seconds while average runtime is 672.1 seconds for SA. Furthermore, CPLEX can only solve some of the medium warehouses instances. Each value reported in the medium instances is an average across the solved replicates of the indicated instance type. SA produces solutions that have an average of 2.35% gap to the CPLEX lower bounds. The average runtime of CPLEX across all instances

is still 3600 seconds while average runtime is 787.15 seconds for SA. Finally, CPLEX failed to find feasible solutions within the 60-minute runtime limit for the large instances. However, SA finds solutions in the large instances with an average runtime of 909.98 seconds.

Table 3.10 compares SA and CPLEX for specific levels of each design factor. Here each row refers to a set of instances for which one of the factors in the experimental design is fixed at a particular level. The gap referred to in this table is the percent difference between the SA solution and the best lower bound from CPLEX within a 60-minute runtime limit. From these results it is apparent the SA performs better on instances with tighter parameters. The average gap of group (*,1,*,*,*) is less than the average gap of group (*,2,*,*,*), as group (*,1,*,*,*) has a tight machine schedule. Similarly, the performance of group (*,*,2,*,*) is better than group (*,*,1,*,*) because group (*,*,2,*,*) is tight in the time window parameter. Group (*,*,*,2,*) performs better than (*,*,*,1,*) as group (*,*,*,2,*) has a more restrictive machine eligibility. And finally, group (*,*,*,*,1) has a better performance because group (*,*,*,*,1) has a more restrictive setup time than group (*,*,*,*,2). This may be because neighborhoods that must be explored are smaller for these instances.

Computation times for SA are summarized in Table 3.11 by instance size and in Table 3.12 by temperature stage. In Table 3.11, it can be observed that the time spent in warm-up is relatively low; only approximately 5% of total runtime. During warm-up, the time spent on Insert() is only slightly higher than on Interchange(). The majority of the runtime is spent in the SA stage (91% of total runtime). This is by design, to allow plenty of time for neighborhood exploration at each temperature stage. In the SA stage, the time spent on Insert() is approximately 6% higher than the time spent on Interchange(). One possible explanation is that Insert() is being called more frequently as it more often results

Table 3.9: Experiment Results

Factors	Avg. $\|K\|$	# CPLEX solved	CPLEX Total Work (hrs)	CPLEX Runing Time(s)	CPLEX Gap to LB (%)	CPLEX LB	Heuristic Total Work (hrs)	Heuristic Runing Time(s)	Heuristic Gap to LB (%)
(1,1,1,1,1)	16.8	10	103.13	544.82	0.00	103.13	103.43	650.46	0.29
(1,1,1,1,2)	14.9	10	117.02	3631.95	1.42	115.38	117.26	603.30	1.63
(1,1,1,2,1)	16.6	10	108.87	549.72	0.00	108.87	109.66	768.30	0.73
(1,1,1,2,2)	16.6	10	114.97	3600.00	1.58	113.18	113.58	736.56	0.35
(1,1,2,1,1)	16.5	10	114.56	3600.00	3.25	110.95	113.06	695.75	1.90
(1,1,2,1,2)	15.1	10	109.46	3600.00	2.73	106.55	108.39	732.78	1.73
(1,1,2,2,1)	17.1	10	114.09	3600.00	1.85	112.02	112.98	608.68	0.86
(1,1,2,2,2)	17.7	10	112.47	3600.00	2.38	109.86	111.17	626.22	1.20
(1,2,1,1,1)	16.7	10	113.72	3600.00	2.31	111.15	112.54	745.95	1.25
(1,2,1,1,2)	14.9	10	114.14	3600.00	1.93	111.98	112.61	658.29	0.56
(1,2,1,2,1)	14.7	10	111.80	3600.00	2.95	108.60	110.49	737.88	1.74
(1,2,1,2,2)	16.1	10	112.95	3600.00	2.87	109.80	111.40	632.58	1.46
(1,2,2,1,1)	18.5	10	112.71	3600.00	2.48	109.98	111.25	614.52	1.15
(1,2,2,1,2)	16.0	10	113.21	3600.00	2.25	110.72	112.33	625.98	1.46
(1,2,2,2,1)	14.1	10	113.68	3600.00	1.97	111.48	112.70	603.36	1.09
(1,2,2,2,2)	14.5	10	112.75	3600.00	2.97	109.50	111.20	723.54	1.55
(2,1,1,1,1)	32.6	9	209.89	3600.00	2.53	204.71	210.53	879.25	2.84
(2,1,1,1,2)	36.2	6	217.70	3600.00	1.97	213.49	218.09	783.77	2.16
(2,1,1,2,1)	25.2	5	242.17	3600.00	2.71	235.79	240.60	861.26	2.04
(2,1,1,2,2)	21.2	9	255.30	3600.00	2.06	250.16	256.27	893.55	2.45
(2,1,2,1,1)	13.9	3	211.38	3600.00	1.92	207.39	211.72	788.93	2.08
(2,1,2,1,2)	38.7	5	243.26	3600.00	1.53	239.59	244.68	748.92	2.12
(2,1,2,2,1)	25.3	7	235.15	3600.00	3.06	228.17	234.41	800.20	2.73
(2,1,2,2,2)	23.4	5	231.06	3600.00	2.26	225.96	232.57	755.20	2.93
(2,2,1,1,1)	37.5	3	199.92	3600.00	3.37	193.39	198.55	886.28	2.67
(2,2,1,1,2)	36.0	3	196.32	3600.00	2.03	192.41	196.72	749.00	2.24
(2,2,1,2,1)	23.4	9	246.34	3600.00	3.02	239.11	245.54	850.24	2.69
(2,2,1,2,2)	15.0	9	234.05	3600.00	1.69	230.16	234.82	899.60	2.03
(2,2,2,1,1)	20.6	8	223.49	3600.00	1.93	219.26	225.13	797.46	2.68
(2,2,2,1,2)	26.9	7	186.63	3600.00	2.84	181.47	185.78	719.94	2.38
(2,2,2,2,1)	18.1	3	246.26	3600.00	2.15	241.08	246.59	896.09	2.29
(2,2,2,2,2)	28.9	8	191.48	3600.00	2.38	187.02	191.91	884.36	2.61
(3,1,1,1,1)	-	0					370.23	952.73	
(3,1,1,1,2)	-	0					323.26	866.65	
(3,1,1,2,1)	-	0					381.61	890.53	
(3,1,1,2,2)	-	0					322.63	995.93	
(3,1,2,1,1)	-	0					393.80	891.71	
(3,1,2,1,2)	-	0					313.77	883.42	
(3,1,2,2,1)	-	0					313.17	902.81	
(3,1,2,2,2)	-	0					358.38	845.35	
(3,2,1,1,1)	-	0					399.39	832.09	
(3,2,1,1,2)	-	0					349.13	908.56	
(3,2,1,2,1)	-	0					334.79	889.43	
(3,2,1,2,2)	-	0					366.87	913.94	
(3,2,2,1,1)	-	0					326.71	814.42	
(3,2,2,1,2)	-	0					298.18	878.73	
(3,2,2,2,1)	-	0					313.47	948.34	
(3,2,2,2,2)	-	0					328.24	826.42	

Table 3.10: Heuristic Performance Gaps (% deviation from best known lower bound) per Instance Class

Factors		Avg. Gap	Max Gap	Min Gap
Warehouse Types	(1,*,*,*,*)	1.12	2.71	0.10
Machine	(*,1,*,*,*)	1.03	2.71	0.10
Schedule	(*,2,*,*,*)	1.21	2.02	0.34
Time	(*,*,1,*,*)	1.23	2.71	0.23
Windows	(*,*,2,*,*)	1.01	2.14	0.10
Machine	(*,*,*,1,*)	1.26	2.71	0.23
Eligibility	(*,*,*,2,*)	0.98	2.14	0.10
Setup Time	(*,*,*,*,1)	1.24	2.71	0.10
	(*,*,*,*,2)	1.00	2.52	0.34
Overall	(*,*,*,*,*)	1.12	2.71	0.10

in improving solutions than does Interchange().

Table 3.11: Heuristic Solving Time for Stages and Operators

Model Size	Total Avg. Time (s)	Init. Sol. Time (s)	Warm Up Stage (s) (%)		Simulated Annealing Stage (s) (%)	
			Insert	Interchange	Insert	Interchange
Small (1,*,*,*,*)	672.76	21.03 (3.13)	20.13 (2.99)	16.60 (2.47)	338.04 (50.25)	276.96 (41.17)
Medium (2,*,*,*,*)	787.15	23.01 (2.92)	22.27 (2.83)	18.10 (2.30)	375.69 (47.73)	348.09 (44.22)
Large (3,*,*,*,*)	909.98	29.12 (3.20)	25.17 (2.77)	21.79 (2.39)	443.45 (48.72)	390.46 (42.91)

* Percentage value in the brackets is percentage of total average running time.

Table 3.12 provides the runtimes attributed to various temperature stages. For all instance sizes (i.e. columns), it can be observed that, in general, as temperature decreases, runtime per temperature stage also decreases. This occurs because as the temperature is reduced, the probability of accepting a non-improving solution is also reduced. Fewer

solution updates occur, leading to less time spent in the temperature stage. An example of the convergence plot is illustrated in Figure 3.11. From this plot, it can be observed that many non-improving solutions are explored when the temperature is high, and the algorithm converges to a good solution once the temperature drops.

Table 3.12: Heuristic Solving Time for Temperature Stages

Temperature Stage ($T =$)	**Avg. Time for Small (1,*,*,*,*)(s) (%)**	**Avg. Time for Medium (2,*,*,*,*)(s) (%)**	**Avg. Time for Large (3,*,*,*,*)(s) (%)**
100.00	31.11 (4.62)	36.11 (4.59)	42.37 (4.66)
90.00	33.47 (4.98)	36.57 (4.65)	42.24 (4.64)
81.00	33.40 (4.96)	37.37 (4.75)	41.43 (4.55)
72.90	33.18 (4.93)	35.72 (4.54)	40.37 (4.44)
65.61	29.24 (4.35)	33.39 (4.24)	41.19 (4.53)
59.05	30.52 (4.54)	36.39 (4.62)	41.10 (4.52)
53.14	31.48 (4.68)	33.76 (4.29)	39.49 (4.34)
47.83	27.87 (4.14)	35.09 (4.46)	41.12 (4.52)
43.05	26.61 (3.96)	33.36 (4.24)	40.67 (4.47)
38.74	26.23 (3.90)	33.22 (4.22)	36.79 (4.04)
34.87	26.55 (3.95)	31.40 (3.99)	39.26 (4.31)
31.38	29.40 (4.37)	34.20 (4.34)	34.58 (3.80)
28.24	27.35 (4.07)	29.28 (3.72)	38.37 (4.22)
25.42	27.24 (4.05)	28.83 (3.66)	35.45 (3.90)
22.88	27.40 (4.07)	32.30 (4.10)	34.41 (3.78)
20.59	22.51 (3.35)	30.97 (3.93)	34.23 (3.76)
18.53	23.83 (3.54)	28.08 (3.57)	32.28 (3.55)
16.68	24.43 (3.63)	27.30 (3.47)	35.68 (3.92)
15.01	22.46 (3.34)	26.11 (3.32)	32.61 (3.58)
13.51	23.00 (3.42)	26.94 (3.42)	35.20 (3.87)
12.16	22.13 (3.29)	28.36 (3.60)	31.66 (3.48)
10.94	20.82 (3.09)	26.54 (3.37)	31.84 (3.50)
$\leq$10	14.78 (2.19)	22.49 (2.85)	11.57 (1.27)

* Percentage value in the brackets is percentage of total average running time.

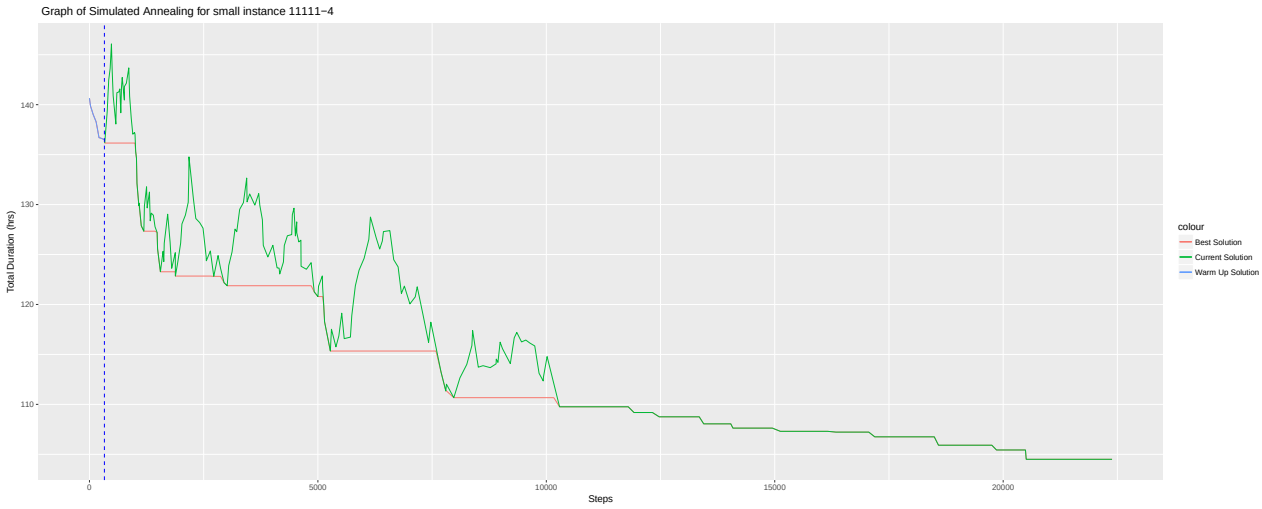


Figure 3.11: Convergence Plot of Instance (1,1,1,1,1-4) from Experiment Design

3.6 Conclusion

This chapter continues the exploration of the unrelated parallel machine scheduling problem from Chapter 2 with two additional practical considerations: setup time and limited machine availability. Similar to Chapter 2, these problem characteristics have not been simultaneously considered in the literature. The setup time discussed in our problem is sequence dependent. A few mathematical model modifications have been made for the problem of this chapter. The limited machine availability is introduced as fixed employee breaks and can be handled by introducing m new jobs to be scheduled (for m employees); one for each machine (i.e. employee), where the only machine eligible to complete the job corresponding to its break is itself.

CPLEX has limited ability in solving the instances we created: for a computer server with 12 CPUs and 24 GB RAM, only the small instances (with 30 machines and 80 jobs) could

be solved within 1 hour running time. For the medium and large instances, CPLEX fails to find any feasible solution before running out of memory. Therefore, a simulated annealing heuristic is developed to solve the problem more efficiently. For the small instances, the heuristic provides solutions that are, on average, 1.12% worse than the lower bound obtained by CPLEX. For the medium and large instances, the heuristic was able to solve them in a relatively reasonable time while CPLEX was not.

Our work has limitations. First of all, some of the machine schedules generated from the heuristic contain many jobs with short durations. This is a result of the job splitting allowed in the problem; move operators can arbitrarily split tasks into smaller and smaller portions. It is not clear whether schedules like this are desirable from a practical perspective. The operator Combine(s) at least partially alleviates this problem. A second limitation is that there is a lack of evidence regarding the performance of the SA for medium and large instances. The SA can be better evaluated if other heuristic methods are introduced and compared.

One area for future work includes possible improvements to the SA. First, operators involving more than two machines could be introduced (e.g., an interchange operator between three different machines). Second, the neighborhood exploration may be more efficient if a tabu list is introduced. Currently the SA selects job-machine pairs for inspection randomly, without considering whether that job-machine pair has been explored in recent iterations.

4 Integrating Uncertain User-Generated Demand Data when Locating Facilities for Disaster Response Commodity Distribution

This chapter presents a new facility location problem variant with application in disaster relief. The problem is unique in that both verified data and unverified user-generated data are available for consideration during decision making. The problem is motivated by the recent need of integrating unverified social data (e.g., Twitter posts) with data from more traditional sources, such as on-the-ground assessments and aerial flyovers, to make optimal decisions during disaster relief. Integrating social data can enable identifying larger numbers of needs in shorter amounts of time, but because the information is unverified, some of it may be inaccurate. This paper seeks to provide a "proof of concept" illustrating how the unverified social data may be exploited. To do so, a framework for incorporating uncertain user-generated data when locating Points of Distribution (PODs) for disaster relief is presented. Then, three decision strategies that differ in how the uncertain data is considered are defined. Finally, the framework and decision strategies are demonstrated via a small computational study to illustrate the benefits user-generated data may afford across a variety of disaster scenarios.

4.1 Introduction

Disasters are prevalent today and the rapid delivery of food, water and medical attention is critical in minimizing suffering for those impacted. Over 6900 disasters were reported during the ten year period from 2002 to 2011, causing over 1.2 million total deaths and affecting almost 2.7 billion people worldwide [41]. Mega-disasters resulting in massive death

tolls occurred both in 2010 with the earthquake in Haiti and in 2011 with the earthquake and tsunami in Japan [42]. Disasters leave impacted populations in need of food, water, shelter, and medical attention, among other things. Major relief organizations such as the International Federation of Red Cross and Red Crescent Societies recognize the critical roles of logistics and the optimized use of scarce resources in saving lives in disasters [43]. The academic community is also aware of the importance of logistics in disaster operations. The authors of a 2011 survey paper on disaster relief routing state "much of successful and rapid relief relies on the logistics operations of supply delivery" [44].

The Federal Emergency Management Association (FEMA) describes three primary methods of issuing supplies after a disaster [45]. Using *mobile delivery*, vehicles deliver supplies directly to drop locations and points where needs have been identified. This method is useful in rural areas and where transportation infrastructure damage has occurred. Using *direct delivery*, supplies are delivered to a specific location such as a shelter or hospital. The types of supplies to be delivered and quantities of each are predetermined. Lastly, in the *Points of Distribution* (POD) method, commodities are delivered to centralized points (i.e., PODs) and impacted populations come to the PODs to pick them up. The available commodities typically include food and water and may also include ice, tarps and blankets [45]. To summarize, supplies are taken to the end users in the mobile and direct delivery methods, while the end users travel to the supplies when PODs are used. POD location decisions are the focus of this paper; specifically, those that must be made after a disaster occurs.

The timeline for POD operation depends on the scope of the disaster and is relatively short when compared with facilities in commercial settings. For example, the timeline

outlined in one POD location paper suggests PODs will be established within four days post-disaster and operate until approximately one week post-disaster when there is a transition from response to recovery operations [46]. Within this context, required decisions in practice may include where and when to open PODs, when to close and/or relocate them, and what demand each will serve. The type of each POD may also need to be determined. The United States Army Corps of Engineers identifies three POD types, distinguishable primarily by their size and the number of persons they are intended to serve [45]. In this paper, we restrict our focus to only a subset of these POD-related decisions. Specifically, we study where to open uncapacitated PODs at a single point in time and what demand each will serve. By doing so, we are able to build preliminary insights regarding the incorporation of social data during the planning process.

In order to make good POD location decisions, information regarding the locations and needs of impacted groups is essential. Situational awareness refers to possessing an understanding of the situation at hand in order to inform better decisions. It is critical in planning logistics activities to support disaster response. Traditionally this information has become available as on the ground assessments – time consuming efforts – are completed. However, social media usage during emergencies is accelerating the pace at which information becomes available to emergency managers [47]. Surveys conducted by the American Red Cross in 2011 and 2012 concluded individuals within the U.S. are increasingly using social media to post information relevant to emergencies [48]. Social data that enhance situational awareness provide "tactical, actionable information that can aid people in making decisions" [49]. To inform location planning, a specific need and precise location should be communicated. Consider for example the Twitter post from Hurricane Sandy depicted

in Figure 4.1. It provides an address, states that a house is flooding and alerts the Fire Department of New York (FDNY) to the need. A response from FDNY demonstrates they were monitoring social data for this type of information. While the flooding example in this figure may not influence the placement of PODs, it is clear how social data does have the potential to inform such decisions. For example, if a large group of persons reports having no access to food and water, it may be reasonable to locate a POD near them. If emergency managers had no knowledge of these individuals, the individuals may need to travel farther to reach the nearest POD, or they may not be able to reach a POD at all.

Figure 4.1: Twitter post during Hurricane Sandy [50].

The emergency management community is beginning to adapt to the new demand for social data integrated response activities. Three-quarters of emergency agencies participating in a 2012 survey conducted by CNA Analysis & Solutions and The National Emergency Management Association (NEMA) indicated their agencies use social media in some capacity [47]. The American Red Cross launched the first social-media monitoring platform dedicated to disaster relief in 2012 [51]. But despite movement in the emergency management community to adopt social data usage, two primary barriers to seamlessly integrating social data with large-scale logistics response planning are evidenced. One such barrier is limited ana-

lytics infrastructure. Less than 12% of respondents to the CNA/NEMA survey indicate the social media capability at their agency includes data collection, aggregation, and analysis that are robust for large-scale events [47]. Concerns regarding the accuracy of social data constitute another barrier. Because the data is user-generated, it is initially not verified. A majority of respondents report trusting social media less than traditional sources, and indicate their agency would take action on social data only after verifying it [47]. It is true social data has the potential to be inaccurate. For example, a Twitter user in New York City intentionally spread alarming misinformation about Hurricane Sandy [52]. However, not all social data is inaccurate, and waiting to take action until it is verified contradicts one of its primary advantages – timeliness. Thus integrating unverified user-generated data in emergency planning yields a key tradeoff:

- serve more people in a shorter amount of time by allocating resources to needs that may otherwise be discovered later or not at all, or
- waste precious resources by allocating them to false needs.

In the context of POD location decisions, the specific tradeoff is between locating facilities such that more demand (both verified and unverified) can reach them in a short amount of time, versus potentially placing PODs farther away from the verified points than necessary. This paper presents a framework for evaluating this tradeoff.

The primary research question addressed in this chapter is whether there is value in acting on user-generated data prior to its absolute verification in the context of POD location decisions. Given the limited analytics infrastructure in place at the majority of the agencies surveyed in 2012, the value of a social-data integrated approach must be demonstrated

in order to spur technology development and diffusion throughout the disaster response enterprise. This paper makes a pivotal, albeit simple, first step towards answering this question. The rationale is as follows: each element of user-generated content (e.g., a Tweet, a youTube video) can be viewed as potentially admissible information in the planning process. The user-generated content specifies information about a demand point (e.g, the location and quantity of people in need of the supply or service). Based on this rationale, emergency managers need to choose a strategy regarding how the uncertain data will be incorporated into the planning process. For example, they may choose to ignore it completely, give it equal weight to the verified data, or pursue a compromise policy (where they, for instance, only consider a subset of the user-generated content). Through the models and a small computational study presented in this paper, the performance of such strategies across a range of disaster situations is evaluated. Specifically, the following three research questions are addressed: how do decision strategies that (i) ignore all unverified data, (ii) include all unverified data, and (iii) use scenario planning to account for unverified data perform across a range of disaster situations?

The contributions of this chapter are as follows. A framework for incorporating uncertain user-generated data in disaster relief POD location decisions is presented. To the best of our knowledge, this is the first paper to simultaneously consider two classes of verified and unverified demand when placing facilities. Section 4.2 will describe how this is different from other literature on facility location under uncertainty and multi-commodity facility location. Second, the paper proposes three strategies that can be used by an emergency manager faced with a POD location decision for which both verified and unverified data are available. One of these reflects current practice, in which only verified data is considered,

and the other two reflect ways in which the unverified data can additionally be considered. Third, the framework and strategies are demonstrated via a computational study comprised of (i) a traditional facility location data set from the literature and (ii) a realistic case study based on a large-magnitude earthquake scenario with an existing transportation network infrastructure. The results suggest that POD location planning strategies that incorporate uncertain user-generated data, even in a naïve way, provide competitive results across a broad range of test instances studied. This provides initial insights into the usefulness of user-generated data for POD location planning.

The remainder of the chapter is organized as follows. Section 4.2 provides a literature review. In Section 4.3, the problem addressed in this chapter is defined and its formulation presented. Section 4.4 describes the computational study design and results are given in Section 4.5. Finally, conclusions are provided in Section **??**.

4.2 Literature Review

This section provides brief reviews of the literature addressing disaster logistics, POD location problems, facility location problems under uncertainty, multi-commodity facility location problems and the use of user-generated content in disaster relief.

Disaster relief supply chains are characterized by unpredictable, suddenly occurring, large-magnitude demands and high stakes associated with timely delivery [53]. Disaster logistics problems can be classified according to whether the decisions considered occur before or after the disaster. Those operations carried out prior to disaster occurrence are *preparation* activities and they play an instrumental role in strategic planning (e.g., stock pre-positioning) [54]. The operations that take place after a disaster include *immediate response* and *recon-*

struction [54]. Immediate response involves activities such as relief distribution, evacuation of displaced people, and transportation and treatment of disaster victims. Reconstruction includes activities carried out across a longer horizon post-disaster, such as debris removal. The scope of this paper lies within immediate response. More specifically, we focus on the location of disaster relief facilities to be used for the distribution of relief commodities in the immediate response phase, like the PODs described in Section **??**. We only review papers with this same focus. Thus, papers addressing facility location for pre-positioning (e.g., [53, 55, 56]) and the location of emergency medical services for large-scale emergencies and disasters (e.g., [57, 58]) are not reviewed because operationally, those problems are quite different from commodity distribution during immediate response.

The literature addressing facility location problems for commodity distribution during the immediate response phase of disaster relief includes only a few known papers at the time of this writing. Jaller and Holguín-Veras (2011) develop a model that considers facility congestion in determining how many PODs should be located and what their capacities should be [59]. Widener and Horner (2011) employ a hierarchical capacitated median problem to place distinct facilities providing different levels of assistance [60]. Gormez et al. (2011) study both pre-positioning and immediate response facility location decisions as part of a two-tier distribution system for disaster response [61]. The facilities located for immediate response are conceptualized as temporary shelters for refugees. A variant of the p-median model is used to determine these locations. The primary difference between these papers and the work described here is that none of the papers in the literature consider user-generated data to describe uncertainty in situational awareness when making POD location decisions.

While POD location planning is not well studied the literature, POD location prob-

lems can be modeled as variants of the classic facility location problem, for which the literature is rich. Owen and Daskin (1998) review the literature addressing static, dynamic and stochastic location models [62]. Snyder (2006) provides a more recent review of these topics and discusses robust formulations as well [63]. Dynamic models assume demand is known or varies deterministically over time. In contrast, randomness in problem characteristics such as demand, cost and travel times exists in uncertainty situations. In the Snyder review paper, facility location problems with randomness in problem input are classified into two types: those for which probability distributions are available for the uncertain parameters and thus stochastic models are used, versus those for which probabilistic information is not available and thus robust optimization models are employed [63]. As will be shown in the next section, one of the decision approaches presented in this paper is modeled using a form of robust optimization. Thus, our formulation of a facility location problem using robust optimization is not novel. The contribution lies in the unique application, and in how the problem input is divided into two distinct classes (verified and unverified). Individual decision makers will have some freedom in determining how to use the unverified information. That is, a decision maker can choose to consider only one or both classes of demand when locating facilities. From a modeling perspective, this is distinct from the multi-commodity facility location literature that also considers multiple classes of demand. Warszawski (1973) was one of the first papers to introduce the multi-commodity location problem. In it, each warehouse can be assigned at most one commodity [64]. Later papers relax this restriction, such as Shen (2005), but still require all of the demand of each customer for each commodity to be assigned to a facility [65]. In this paper, a facility can serve demand from more than one commodity (both verified and unverified demand). And, a decision maker can choose to make location

decisions without considering demand for a commodity (i.e., without considering unverified data).

Social data that contain actionable information can benefit emergency response. But, 500,000 Instragram photos and 20 million tweets were generated during Hurricane Sandy alone [66]. Finding this type of information among the huge volumes of data posted to social media during disasters has been compared to finding the proverbial needle in the haystack. However, tools and technology are being introduced to rectify this need. Capabilities for filtering, categorizing, displaying and verifying social data for disaster response are being developed. For example, the Artificial Intelligence for Disaster Response (AIDR) engine leverages both human participation and machine learning to collect and classify messages posted to Twitter during disasters in real-time into categories such as *infrastructure damage* and *needs of those affected* [67]. AIDR is part of the MicroMappers platform, created with the vision to "combine human computing (smart crowdsourcing) with machine computing (artificial intelligence) to filter, fuse and map a variety of different data types such as text, photo, video, and satellite/aerial imagery" [68]. MicroMappers is comprised of web-based crowdsourcing apps people can use to filter and geo-tag various data types which will then be displayed on maps or in spreadsheets. Other examples of systems for leveraging and visualizing crowdsourced disaster data include the Arizona State University Coordination Tracker (ACT) and TweetTracker [69]. Furthermore, researchers are actively designing methods for effective extraction and classification of microblogged data. Tweak the Tweet, an effort originating from within Project EPIC, is in this domain. It asks users to format tweets with specific hashtags so computers can easily extract relevant information [70]. More recently, machine learning techniques such as Bayes classifiers and Support Vector Machines have been

used to automate social data extraction and classification (e.g, [71, 72, 73, 74, 75]). Finally, tools for social data verification include humanitarian platforms such as Verily and Peta-Jakarta and journalist platforms such as BBC's User-Generated Content Hub [76, 77, 66]. Pheme is a related research project, aimed at identifying lies on Twitter in real-time [78]. This paper does not advance knowledge in the areas of social data extraction, classification and verification. Instead, the contribution of this work is to demonstrate how decision support systems for disaster response facility location based on such technologies may be useful.

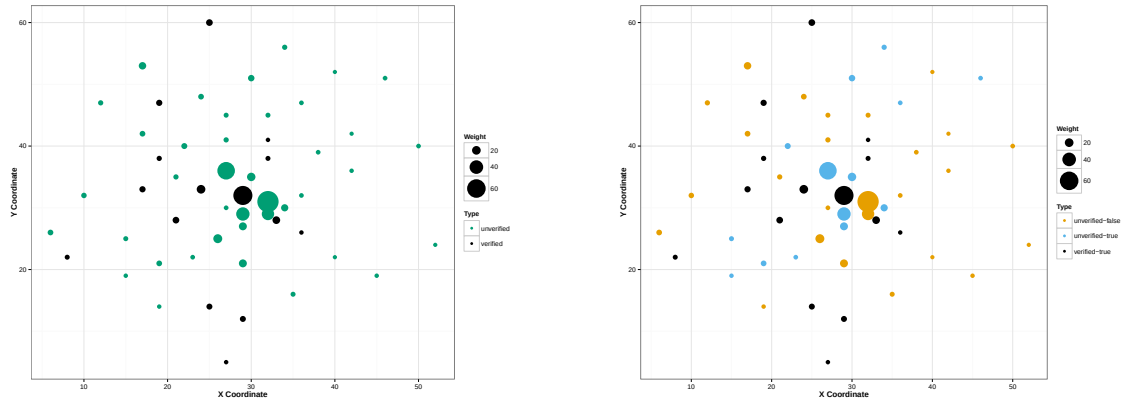
4.3 Problem Description and Formulation

The problem under consideration is as follows. There is a set of demand locations $\mathcal{I}$, each associated with a request for disaster relief supplies. At the time of planning, the demand locations are partitioned into two sets: (i) verified data from traditional sources $\mathcal{V}$, and (ii) unverified user-generated data $\mathcal{U}$. Each request in $\mathcal{I}$ specifies a demand magnitude (d_i) and location. This information is known with certainty for requests in $\mathcal{V} \subseteq \mathcal{I}$ but not in $\mathcal{U} \subseteq \mathcal{I}$. For those requests in $\mathcal{U}$, both the demand and location are taken from a post to a social platform. Either demand exists at the stated location in the stated magnitude, and thus the request is *true*, or it does not, and it is *false*. The classification of requests in $\mathcal{U}$ as true or false is not known at the time of planning. Furthermore, there is no probabilistic information to describe the likelihood a request in $\mathcal{U}$ is true. Requests in $\mathcal{V}$ are true by default because they are verified. There is a set of potential facility locations $\mathcal{J}$ and the facilities to be opened have unlimited capacity. The distance c_{ij} between each demand point $i \in \mathcal{I}$ and candidate facility location $j \in \mathcal{J}$ is known. The problem is to determine how many

and which facilities to open and assign all *true* demand to them, such that the total demand weighted distance between true demand points and their assigned facilities is minimized. Note that this objective cannot be fully evaluated until the unverified demand locations, $\mathcal{U}$, can be partitioned into two sets $\mathcal{T}$ (true) and $\mathcal{F}$ (false). It is assumed that this partitioning takes place some time after the facilities have been located. This assumption is plausible given the application. PODs are opened, and then affected persons begin visiting them, regardless of whether those persons' needs were discovered through traditional or social data sources.

Figures 4.2a and 4.2b depict the demand information available at two different points in time for this planning problem, using data from Swain (1971) for illustrative purposes [79]. Each circle in the two figures represents a demand element. The coordinate of the circle represents its location and the size of the circle represents its magnitude (a larger circle has higher demand than a smaller circle). The two figures are comprised of the same set of demand elements. Figure 4.2a illustrates the information available at the time of planning. Each circle is classified as either verified (black) or unverified (green). The decision maker must decide where to place PODs using only the information in Figure 4.2a. Figure 4.2b represents the information that becomes available some time after facilities have been placed. The set of unverified demands have been partitioned into false (yellow) and true (blue) demands. The set of verified demands has not changed.

Two-stage programming is proposed for the study of this problem. Whereas deterministic optimization problems are formulated with known parameters, real world problems often include parameters which are not known at the time when decisions should be made. Two-stage programs are used to model these situations in which a decision maker takes



(a) Information available before planning (b) Information available after planning

Figure 4.2: Illustration of information available before and after planning

some action in the first stage (prior to uncertainty being resolved) and then a random event affecting the first-stage decisions occurs. In a second stage, a recourse decision is chosen by the decision maker to compensate for the effects of the random event [80]. The basic idea of two-stage programming is that optimal decisions should be based on data available at the time of making the decision and should not depend on future observations. During the first stage one needs to make a "here-and-now" decision before the realization of the uncertain data. During the second stage, after the true scenario is realized, one performs a subsequent optimization which describes the optimal behavior when the uncertain data is revealed [80].

A two-stage model can be conceptualized for this problem as follows. Following disaster occurrence, verified traditional and unverified user-generated data describing demand becomes available. In the first stage, an emergency manager chooses POD locations and allocates demand points to them before the accuracy of the unverified data can be investigated. The location-allocation decisions are made so as to optimize an objective function for a set of demand points the manager selects. For example, if the objective is to minimize

demand-weighted distance, the manager may decide to optimize this distance only for the verified locations, or for all verified and unverified locations. As the true scenario is realized, each unverified data element is classified as true or false. The second stage is an observational stage. We assume that demand not accounted for in the first stage will appear at the nearest open facility. Similarly, false demand that was accounted for in the first stage will not appear at its designated (or any other) facility. Thus, the second stage in our model is not a decision making stage; the decision maker does make reallocation decisions after learning which demands are true and false. Instead, it is an evaluation stage. The total demand weighted distance is computed under the assumption that each true demand point will visit the nearest facility that was opened in stage one, and false demands will not visit a facility. While factors such as facility congestion and damaged transportation network infrastructure may influence behavior in practice, those extensions are saved for future work.

The two-stage model for this problem is more formally described as follows. The first stage problem is to determine how many and which facilities to locate and how to allocate demand to them. Based on their decision making preference, the emergency manager selects the subset of demand points $\mathcal{I}' \subseteq \mathcal{I}$ to consider. For example, if the emergency manager would like to focus only on the verified demand, they will choose the subset $\mathcal{I}' = \mathcal{V}$. If instead they prefer to focus on both the verified and unverified demand, they will select the subset $\mathcal{I}' = \mathcal{V} \cup \mathcal{U}$. This problem is different from a two-stage stochastic programming problem because in our problem the probability of whether a demand is accurate or not is unknown before decisions are made. Let Y_j be a binary variable indicating whether facility j is opened, X_{ij} be a binary variable indicating whether demand point i is assigned to facility j, d_i be the demand of i and c_{ij} be the distance between demand point i and potential facility

location j. A P-median problem is solved, denoted Problem 1a:

Problem 1a:

$$\text{Min} \sum_{i \in \mathcal{I}'} \sum_{j \in \mathcal{J}} d_i c_{ij} X_{ij}, \tag{4.1}$$

$$\text{Subject to} \sum_{j \in \mathcal{J}} X_{ij} = 1 \quad \forall\ i \in \mathcal{I}', \tag{4.2}$$

$$X_{ij} \leq Y_j \quad \forall\ i \in \mathcal{I}',\ j \in \mathcal{J}, \tag{4.3}$$

$$\sum_{j \in \mathcal{J}} Y_j = P, \tag{4.4}$$

$$X_{ij} \in \{0, 1\} \quad \forall\ i \in \mathcal{I}',\ j \in \mathcal{J}, \tag{4.5}$$

$$Y_j \in \{0, 1\} \quad \forall\ i \in \mathcal{J}. \tag{4.6}$$

Objective function (4.1) minimizes total demand-weighted distance between the subset of considered demand locations and their assigned facilities. Constraint set (4.2) ensures each demand point in the subset of considered locations is assigned to a facility. Constraint set (4.3) ensures those demand points are assigned only to open facilities. Constraint (4.4) requires that exactly P facilities are opened. Finally, constraint sets (4.5) and (4.6) describe the binary nature of the decision variables. Note that depending on the strategy adopted by the emergency manager, some demand points may not be allocated to facilities during stage one (i.e., those demand points in $\mathcal{I} \backslash \mathcal{I}'$).

The second stage problem is to determine the total distance from all *true* demand points to their nearest facilities once the uncertainty has been resolved and the set of unverified social data requests $\mathcal{U}$ is partitioned into true ($\mathcal{T}$) and false ($\mathcal{F}$) requests (recall that by

default, verified requests in $\mathcal{V}$ are also true). Let Z_{ij} be a binary decision variable indicating whether demand point i is assigned to facility j in the second-stage problem. Then, the following Problem 2 is solved. Note that $\boldsymbol{Y}$, the vector of aggregated Y_j values, is now an input parameter instead of a decision variable, and the requirement that exactly P facilities be opened is no longer needed because these decisions carry over from stage one:

Problem 2:

$$\text{Min} \sum_{i \in \mathcal{V} \cup \mathcal{T}} \sum_{j \in \mathcal{J}} d_i c_{ij} Z_{ij}, \tag{4.7}$$

$$\text{Subject to:} \sum_{j \in \mathcal{J}} Z_{ij} = 1 \quad \forall\ i \in \mathcal{V} \cup \mathcal{T}, \tag{4.8}$$

$$Z_{ij} \leq Y_j \quad \forall\ i \in \mathcal{V} \cup \mathcal{T},\ j \in \mathcal{J}, \tag{4.9}$$

$$Z_{ij} \in \{0, 1\} \quad \forall\ i \in \mathcal{V} \cup \mathcal{T},\ j \in \mathcal{J}. \tag{4.10}$$

Objective function (4.7) minimizes total demand-weighted distance between the subset of true demand locations and their assigned facilities. Constraint set (4.8) ensures each true demand point is assigned to a facility. Constraint set (4.9) ensures those demand points are assigned only to facilities opened during stage one. Finally, constraint set (4.10) describes the binary nature of the allocation decision variables. A facility could potentially exist at a *false* demand point, if that demand point were included in the set $\mathcal{J}$ and selected as a facility in stage one.

This two-stage framework can be used to represent and evaluate decision maker *strategies*. A decision maker *strategy* is defined as a rule describing the subset of demand points

to consider during stage one. Two plausible strategies to consider in this initial case study include one representing a decision manager who is optimistic regarding the potential of unverified social data, and another representing a decision manager who is pessimistic. Strategies *Consider All* and *Consider Only Verified* model these two perspectives. These strategies are detailed below, along with a description of how their performance can be evaluated in the context of the two-stage problem.

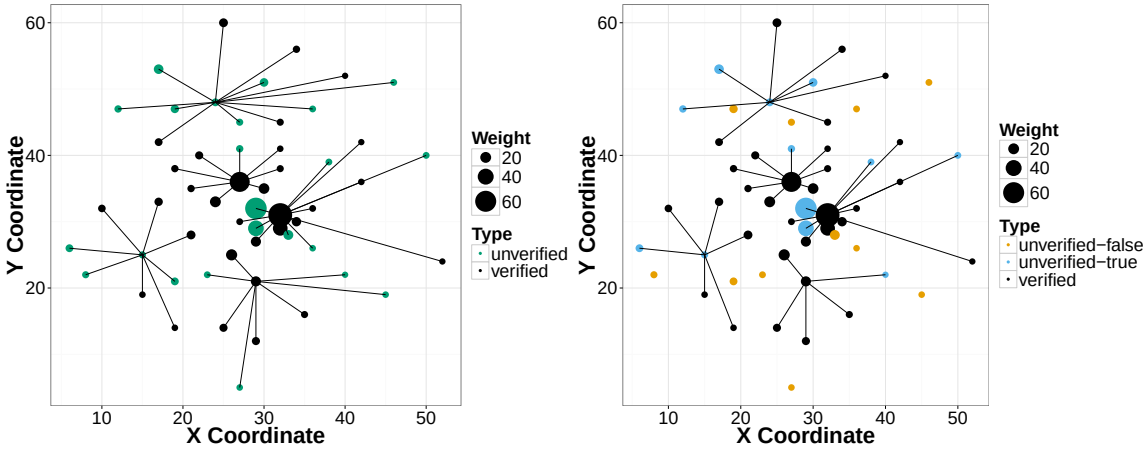
1. *Consider All (CA)*: Consider all verified and unverified data when making location decisions in stage one. That is, solve Problem 1a with $\mathcal{I}' = (\mathcal{V} \cup \mathcal{U})$. Then, solve Problem 2 using the $\boldsymbol{Y}$ decision variables from the optimal solution to Problem 1a as input. The impact of stage two decisions will be to de-allocate unverified demand points that are now known to be false ($\mathcal{F}$) from their assigned facilities. While this strategy will benefit from accounting for the true unverified demand in stage one, a downside is that the location decisions made during stage one are affected by false demand.

2. *Consider Only Verified (COV)*: Consider only the verified data when making location decisions in stage one. Solve Problem 1a with $\mathcal{I}' = \mathcal{V}$. Then, solve Problem 2 using the $\boldsymbol{Y}$ decision variables from the optimal solution to Problem 1a as input. The impact of stage two decisions will be to allocate unverified demand points now known to be true ($\mathcal{T}$) to the nearest facility that is already open. An advantage of this strategy is that false demand does not impact stage one location decisions. However, the disadvantage is that true unverified demands, which were not considered, may need to travel longer distances to reach the nearest facility.

Note that for both CA and COV, the determination of stage two allocation decisions does not require solving the integer program denoted *Problem 2*. According to both strategies, the true ($\mathcal{T}$) and verified ($\mathcal{V}$) demand points must be allocated to the PODs opened in stage one. Each of these demand points will always be assigned to the nearest open POD because there are no capacity constraints on facilities and the objective is to minimize total weighted distance. However, we include the generalized model here because it allows for explicitly noting the information considered in stages one and two. Also, the generalized model provides the building blocks needed for the study of future problem variants, such as capacitated facilities.

Figures 4.3 and 4.4 illustrate the CA and COV strategies for an example instance where the number of facilities to be located is five. Each circle in each figure is a demand, and the demand at each location is proportional to the size of the circle. The total weighted distance associated with the facility assignments in each figure is provided in brackets in the figure caption. For this example instance, the set of potential facility locations was chosen as the demand points under consideration during Stage 1 (that is, $\mathcal{J} = \mathcal{I}$ for each strategy). Consider first Figure 4.3a depicting stage one for the CA strategy. Five facilities are located. In this example, verified black locations are chosen for all five facilities (locations 13, 14, 24, 41, 43), but this will not necessarily always be true for the CA strategy. All verified and unverified demand points are allocated to the selected facilities. Then, in the second stage in Figure 4.3b, the selected facilities are evaluated against the demand realization. The uncertain demands are now classified as either true (blue) or false (yellow), and the false demand points have been de-allocated from their assigned facilities. The total weighted distance is lower in stage two than stage one because it is computed across fewer demand

points. Now consider Figure 4.4a, corresponding to stage one of the COV strategy. All five facilities are opened at verified demand locations (13, 14, 18, 23, 24) and only verified demands are allocated to them. In stage two depicted in Figure 4.4b, the unverified demands newly classified as true (blue) are now assigned to their nearest facility. In this case the total weighted distance is higher in stage two than in stage one because more demand points are included in the computation. Comparing CA and COV, the former offers better performance for this particular instance, as noted by the smaller stage two distance (2467.0 versus 2471.3). In summary, under both strategies, the facility location decisions and the facility assignments for the verified demand points do not change between stage one and stage two. However, total weighted distance changes between the stages because some unverified demand points are either newly allocated to or de-allocated from facilities in stage two. Therefore, strategies should be compared on the basis of stage two distance, not stage one.

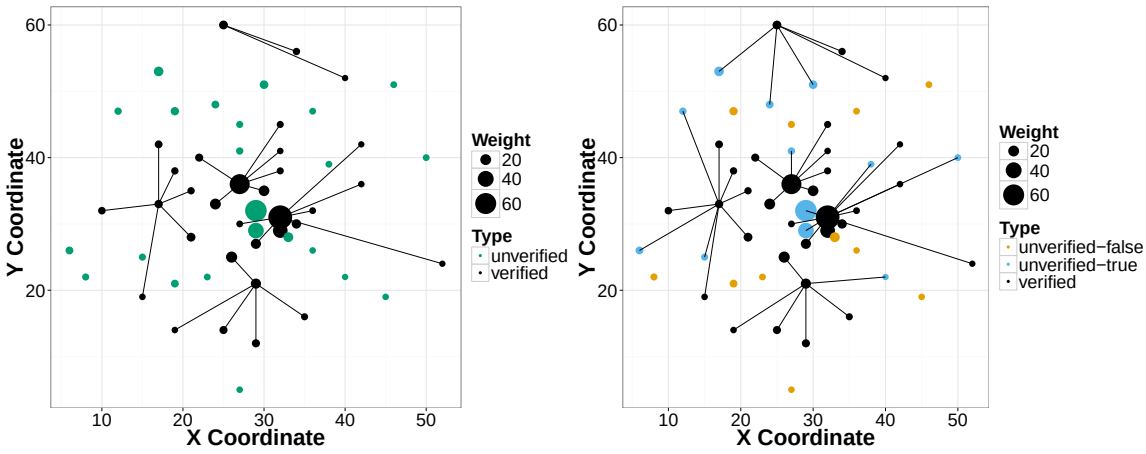


(a) Stage one [2950.4]

(b) Stage two [2467.0]

Figure 4.3: Illustration of two-stage model for CA strategy

In addition to the optimistic and pessimistic decision maker strategies described



(a) Stage one [1358.3]

(b) Stage two [2471.3]

Figure 4.4: Illustration of two-stage model for COV strategy

above, a third plausible strategy to consider utilizes scenario planning. In this case, a decision maker seeks to hedge against multiple possible demand realizations when placing facilities. Here, a scenario r is a demand realization comprised of all verified demand points in $\mathcal{V}$, plus a subset $\mathcal{U}_r \subseteq \mathcal{U}$ of unverified demand points. The subset $\mathcal{U}_r$ is constructed by randomly sampling from the unverified set $\mathcal{U}$, where each demand point in $\mathcal{U}$ is either *true* in scenario r and thus included in $\mathcal{U}_r$, or false and thus not included. Using a set $\mathcal{R}$ of potential scenarios, a minimize maximum regret model can be formulated. Regret is the difference between the objective value of the optimal solution for a realized scenario and the objective value of a compromise. The minimax regret measure is attractive because it does not require the estimation of scenario probabilities. Without probabilistic information describing the potential for each unverified location to be true (or false), scenario probabilities cannot be computed. The minimax regret formulation from Owen and Daskin (1998) is adapted below [62]. Here, the distance between locations c_{ij} does not vary by scenario. We refer to this as Problem

1b, as it must be solved during stage one. Let X_{ijr} be a binary variable indicating whether demand point i is assigned to facility j in scenario r and define W_r^* as the objective value of the optimal P-median solution for scenario r (i.e., the objective value of the optimal solution to Problem 1a when $\mathcal{I}'$ is comprised of $\mathcal{V} \cup \mathcal{U}_r$). Then, the minimax regret formulation is:

Problem 1b:

$$\text{Minimize} \max_{r \in \mathcal{R}} \sum_{i \in \mathcal{V} \cup \mathcal{U}_r} \sum_{j \in \mathcal{J}} d_i c_{ij} X_{ijr} - \mathcal{W}_r^* \tag{4.11}$$

$$\text{subject to} \sum_{j \in \mathcal{J}} X_{ijr} = 1 \quad \forall\ i\ \in \mathcal{I}, r\ \in \mathcal{R}, \tag{4.12}$$

$$X_{ijr} \leq Y_j \quad \forall\ i \in \mathcal{I}, j \in \mathcal{J}, r \in \mathcal{R}, \tag{4.13}$$

$$\sum_{j \in \mathcal{J}} Y_j = P, \tag{4.14}$$

$$X_{ijr} \in \{0, 1\} \quad \forall\ i \in \mathcal{I},\ j \in \mathcal{J},\ r \in \mathcal{R}, \tag{4.15}$$

$$Y_j \in \{0, 1\} \quad \forall\ j \in \mathcal{J}. \tag{4.16}$$

Objective function (4.11) minimizes the maximum regret across all scenarios. Constraint sets (4.12) and (4.13) control the assignment of demand points to open facilities. Constraint (4.14) requires that exactly P facilities are opened. Finally, constraint sets (4.15) and (4.16) describe the binary nature of the decision variables. Using this formulation, the Consider Minimax Regret strategy is evaluated as follows.

3. *Consider Minimax Regret (CMR)*: First determine the optimal solution for each scenario $r \in \mathcal{R}$ by solving Problem 1a with $\mathcal{I}' = \mathcal{V} \cup \mathcal{U}_r$. Next, solve the minimize maximum regret problem (Problem 1b), using W_r^* for each $r \in \mathcal{R}$ as input. Evaluate

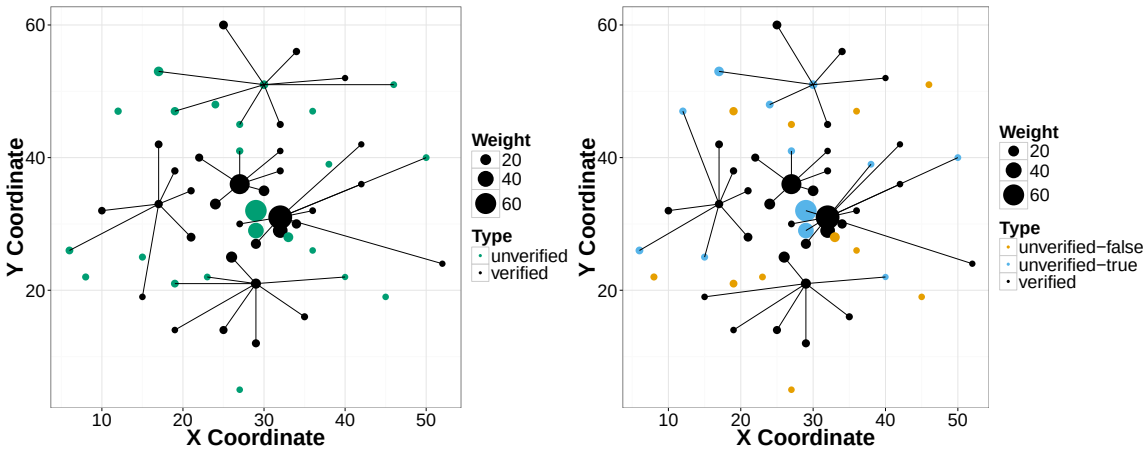
Table 4.1: POD locations selected by each strategy in example instance derived from Swain [79]

Strategy	Facilities opened					Tot. weighted dist. from Problem 2
COV	13	14	18	23	24	2422.3
CA	13	14	24	41	43	2467.0
CMR	13	14	24	31	42	2471.3

the quality of the minimax regret solution by using the optimal location variables $\boldsymbol{Y}$ from Problem 1b as input parameters when solving Problem 2. The impact of stage two decisions may include (i) de-allocating false demand points ($\mathcal{F}$) from their assigned facilities, and/or (ii) allocating user-generated demand points newly classified as true ($\mathcal{T}$) to the nearest facility that is already open.

Figure 4.5 illustrates the CMR strategy. As depicted in Figure 4.5a for a scenario r, all of the verified demand points (black) are allocated to facilities in stage one, and a subset of the unverified demand points (green) are also. The unverified demand points that are allocated to facilities in stage one are those comprising the set $\mathcal{U}_r$. Then in stage two the true demand realization is revealed and the solution is depicted in Figure 4.5b. Green points that had facility assignments in stage one and are now blue ($\mathcal{U}_r \cap \mathcal{T}$) retain those assignments, while green points that had facility assignments in stage one and are now yellow ($\mathcal{U}_r \cap \mathcal{F}$) are de-allocated. Green points that did not have facility assignments in stage one ($\mathcal{U} \backslash \mathcal{U}_r$) and are now blue are newly allocated to facilities. The total weighted distance of 2422.3 in stage two is lower than the distances from the CA and COV strategies for the same instance (2467.0 and 2471.3, respectively).

Table 4.1 summarizes which PODs were opened according to each strategy for the example instance, in order to highlight differences in Stage 1 decisions among the three



(a) Stage one [2180.5]

(b) Stage two [2422.3]

Figure 4.5: Illustration of two-stage model for CMR strategy

approaches. Note that PODs were opened at demand locations 13, 14 and 24 in each strategy, but the additional two POD locations chosen vary.

The next section describes a computational study designed to illustrate the framework presented in this paper and evaluate the performance of the three strategies discussed above for a variety of test instances.

4.4 Computational Study

The test instances designed to illustrate and evaluate the CA, COV and CMR decision strategies are described in this section. A case study is derived from a real Arkansas disaster scenario, and other test instances are derived from the Swain dataset. The Swain data set is one of the standard datasets used in the literature for analyzing and comparing location algorithms [79]. It includes 55 demand points of varying magnitudes. For all test instances based on the Swain data set, we assume the set of potential facility locations is identical to

the set of demand points considered by a particular strategy during Stage 1 (that is, $\mathcal{J} = \mathcal{I}'$).

The case study we develop reflects a large-magnitude earthquake along the New Madrid fault line and focuses specifically on damages projected to occur in the state of Arkansas [46]. A list of subdivisions in a nineteen county region in Northeast Arkansas was developed based on a census report that divided each county into numerous subdivisions [81]. The latitude-longitude coordinates and population of the centroid of each subdivision were collected from census 2000 U.S Gazetteer files. Demand estimates were generated for case study instances using the shelter seeking population by county, as identified in the Gazetteer report [82]. The shelter seeking population estimates were chosen as the basis for demand instead of total population because perhaps not all households in the at risk population would seek mass care commodities from a POD. Ratios of shelter seeking population estimates to total population estimates were computed to approximate the percentage of people in each county presenting demand for mass care commodities at PODs. All demand of each subdivision is assumed to be located at the centroid [82]. Demand locations and magnitudes are summarized in Tables A.1 - A.4 in the Appendix. Given that many schools in Arkansas have facilities that are centrally located and include large stable structures, the schools in the impacted counties comprise the set of candidate POD locations. A list of 127 schools is developed based on EducationBug, a complete listing of educational resources available online [83]. The list of POD locations is provided in Table A.5 and Table A.6 in the Appendix. The road network for the case study region is created using "StreetMap North America - Detailed Streets" in ArcGIS software. Travel distances between locations are also obtained from ArcGIS [84].

Test instances are created from the Swain and Arkansas data sets as follows. First, it

is reasonable to suppose the performance of each decision strategy will depend on the relative proportions of verified and unverified data, and within the unverified data set, the relative proportions of true and false data. For example, a strategy that ignores user-generated data (like COV) may not perform well when most of the data is unverified, and is later discovered to be true. In that situation, only a very small number of demand points that will actually be visiting the facilities are considered when choosing the locations of those facilities. In order to capture the differences in decision strategy performance across a broad range of disaster situations, test instances are created such that the relative proportions of verified versus unverified and true versus false data vary. To do so, a Bernoulli random variable with parameter α is first associated with each demand point in $\mathcal{I}$. The parameter α represents the probability a demand point will be designated as verified (conversely, a demand point is designated as unverified with probability $1-\alpha$). To create a test instance, the probability distributions for each demand point are sampled, effectively partitioning the set $\mathcal{I}$ into the verified and unverified sets $\mathcal{V}$ and $\mathcal{U}$, respectively. Next, an additional Bernoulli random variable with parameter ω is associated with each demand point in $\mathcal{U}$. The parameter ω represents the probability an unverified demand point will be classified as true (and $1-\omega$ the probability it will be classified as false). The probability distributions for each unverified demand point are then sampled, partitioning the set $\mathcal{U}$ into the true and false sets $\mathcal{T}$ and $\mathcal{F}$, respectively. Using this method, the expected numbers of verified demand points in the Swain and Arkansas data sets are 55α and 335α, respectively, while the expected numbers of true unverified demand points are $55\omega(1-\alpha)$ and $335\omega(1-\alpha)$, respectively. Both α and ω are varied within the set $\{0.25,0.50,0.75\}$ in the computational study; thus, 9 combinations of (α,ω) are considered. Instance groups from the Swain data set and the Arkansas data

set are indexed from $S1$ to $S9$ and $A1$ to $A9$ respectively based on increasing values of α and ω. For each of the nine instance groups in each data set, ten random replicates are generated. Within each instance group, replicates are numbered 1 through 10, for example, $S1$-1 through $S1$-10 represent the ten replicates of instance group $S1$.

Table 4.2 summarizes the test instances. For each instance, the table provides the values of α and ω and the expected numbers of verified, unverified true and unverified false demand points. Figure 4.6 provides a graphical representation of instance groups $S1$-$S9$. Each graph corresponds to the first replicate of each instance group. The header for a graph provides the instance group number and value of α and ω. To visually highlight the differences among the instance groups, consider for example the first column in the figure. The proportion of verified data (α) increases moving from top to bottom within the column. This can be seen from the increasing proportion of black points.

Table 4.2: Test instance summary

(α, ω)	Instance group	Veri. $\mathcal{V}$	Unv. true $\mathcal{T}$	Unv. false $\mathcal{F}$	Instance group	Ver. $\mathcal{V}$	Unv. true $\mathcal{T}$	Unv. false $\mathcal{F}$
(0.25,0.25)	S1	13.75	10.31	30.95	A1	83.75	62.81	188.44
(0.25,0.50)	S2	13.75	20.62	20.62	A2	83.75	125.62	125.62
(0.25,0.75)	S3	13.75	30.95	10.31	A3	83.75	188.44	62.81
(0.50,0.25)	S4	27.50	6.88	20.62	A4	167.50	41.88	125.63
(0.50,0.50)	S5	27.50	13.75	13.75	A5	167.50	83.75	83.75
(0.50,0.75)	S6	27.50	20.62	6.88	A6	167.50	125.63	41.88
(0.75,0.25)	S7	41.25	3.44	10.31	A7	251.25	20.94	62.81
(0.75,0.50)	S8	41.25	6.88	6.88	A8	251.25	41.88	41.88
(0.75,0.75)	S9	41.25	10.13	3.44	A9	251.25	62.81	20.94

Each of the three strategies presented in Section 4.3 are applied to obtain solutions for each test instance for each possible value of P (number of facilities). Note that the range

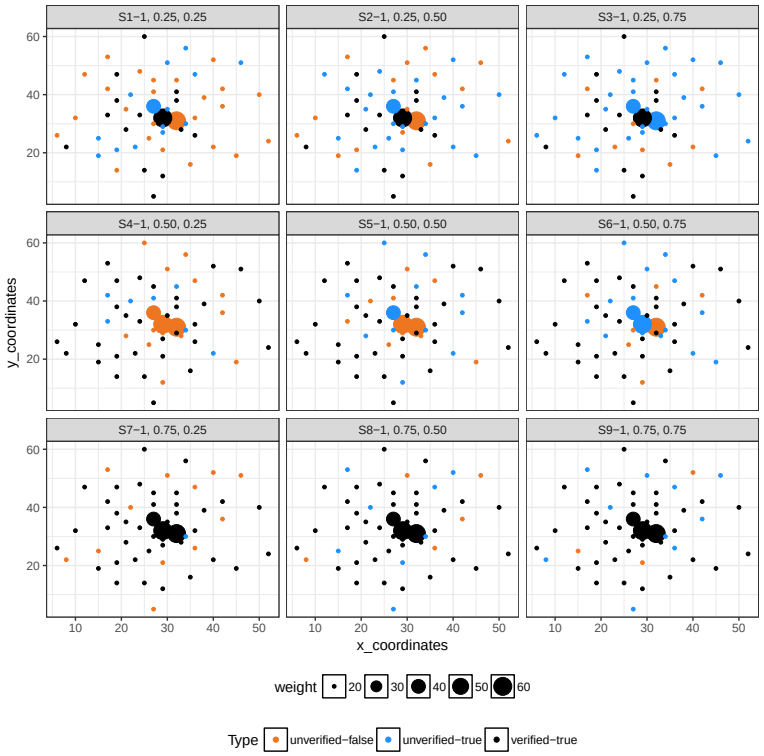


Figure 4.6: Graphical representation of Swain data set instances. Grid labels: instance, α and ω.

of possible values for P in the Swain data set depends on the strategy employed, because the set of candidate facility locations differs for each strategy. The set of candidate facility locations for a particular strategy is the set of demand points considered by the strategy in stage one (i.e., $\mathcal{J} = \mathcal{I}'$). For example, using the CA strategy, in the Swain dataset, since all 55 demand points serve as candidate facility locations ($\mathcal{I}' = \mathcal{I}$), the range for P is $\{1, 2, \ldots, 55\}$. Using the COV strategy, the maximal size of the candidate location set is $|\mathcal{V}|$, so the range of P is $\{1, 2, \ldots, |\mathcal{V}|\}$. Finally, in the CMR case, the range of P for a particular scenario r is $\{1, 2, \ldots, |\mathcal{V} \cup \mathcal{U}_r|\}$. However, the range of P is always $\{1, \ldots, 127\}$ in the Arkansas case study instances because the set of potential facility locations is distinct from the set of demand locations. That is, the set $\mathcal{J}$ includes the same 127 locations regardless of which strategy is being employed.

It should also be noted that for the CMR strategy, the total number of distinct scenarios in a problem instance with unverified demand set $\mathcal{U}$ is $2^{|\mathcal{U}|}$. When the size of U is large, it is more tractable to carry out scenario planning using a subset of potential demand realizations, instead of all possible demand realizations. In this study, the ten random replicates associated with each (α,ω) pair comprise the subset of potential demand realizations used to evaluate the CMR strategy. While ten replicates is small compared to the total number of possible scenarios (i.e., 2^{55} in the Swain data set, 2^{335} in the Arkansas data set), it is deemed sufficient for the purpose of this study, which is simply to demonstrate the framework presented in this paper.

Solutions for all optimization problems (i.e., Problems 1a, 1b and 2) are obtained using ILOG CPLEX 12.6 via C libraries. The experiments were performed on a server with 24 GB RAM and 12 CPUs. On average, one instance (a single replicate for fixed values of α, ω and P) is solved in less than five seconds for instance groups $S1$-$S9$ and in approximately two minutes for instance groups $A1$-$A9$.

4.5 Results

This section presents the results of the computational study. First, the results are discussed for each strategy individually. Then, comparisons among the solutions obtained for the different strategies are made.

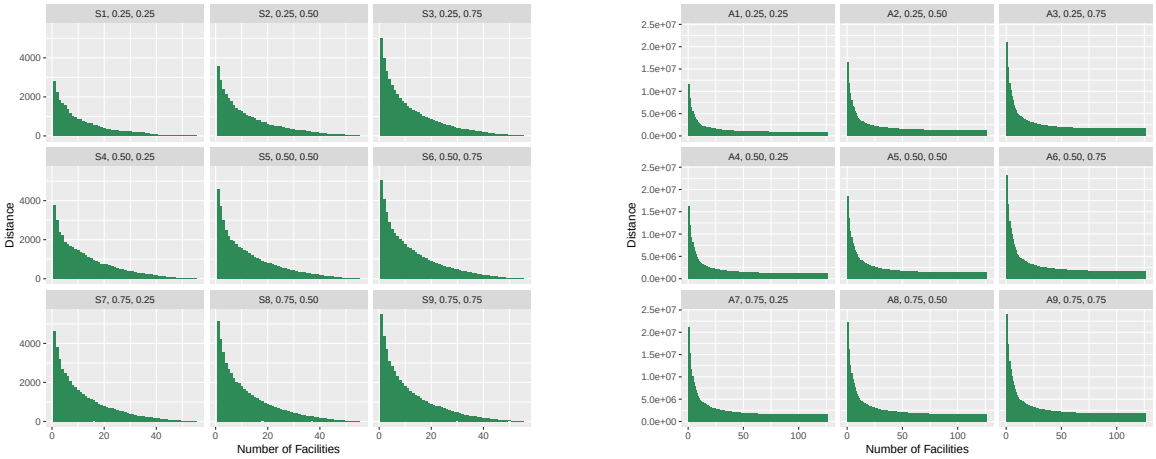
4.5.1 Individual Strategies

Figures 4.7 through 4.9 present the results for the Swain and Arkansas data sets for the strategies CA, COV and CMR, respectively. Each contains nine subfigures representing

the instance groups (the nine (α, ω) combinations). In each subfigure, the header gives the instance group and value of α and ω, the horizontal axis gives the number of facilities opened and the y-axis gives the weighted distance. The values reported are the average weighted distances over ten replications. Note that for the Swain data set, the range of P, the number of facilities opened, varies among the subfigures. This occurs because the cardinality of the set of potential facility locations $\mathcal{I}'$ in *Problem 1a* varies according to parameters α and ω and the decision strategy selected. That is, the set of potential facility locations contains $\mathcal{V} \cup \mathcal{U}$ for CA, $\mathcal{V}$ for COV, and $\mathcal{V} \cup \mathcal{U}_r$ for CMR. The parameter α controls the size of $\mathcal{V}$, as α is the proportion of demand points that are verified, and the parameters α and ω together control the size of $\mathcal{U}_r$, the proportion of unverified demand points that are accurate in scenario r. The range of the horizontal axis does not vary among the subfigures for the Arkansas data set because in this case, the sets of potential facility locations and demand points are disjoint; the number of unverified demand points in a given instance does not influence the number of potential facility locations.

A number of observations can be made from these figures. First, for a given proportion of certain data (α) and number of facilities (P), the demand-weighted distance increases as the proportion of true uncertain data (ω) increases for all three strategies. To see this, compare for example the column for $P = 1$ in each of the three subfigures in row 1 of Figure 4.7a. This behavior occurs because as ω increases, there are more demand points that actually need to visit facilities in stage two. Second, suppose ω and P are now held constant and α is varied. That is, the number of facilities and the the relative "goodness" or accuracy of the uncertain data stay constant, but the amount of verified data increases. In this case, slight increases in demand-weighted distance can be observed. To see this,

compare for example the column for $P = 1$ in each of the three subfigures in column 1 of Figure 4.7a. This also occurs because the total number of true demand points (verified and unverified true) needing facilities in stage two increases as α increases. A third observation that pertains only to the Arkansas data set is that for most instances, after approximately 100 facilities have been opened by any of the three strategies, there is no incremental benefit to opening additional facilities. The nearest facility to every demand point will have already been opened. In fact, beyond $P = 50$, the incremental benefit is very modest.



(a) Swain data set

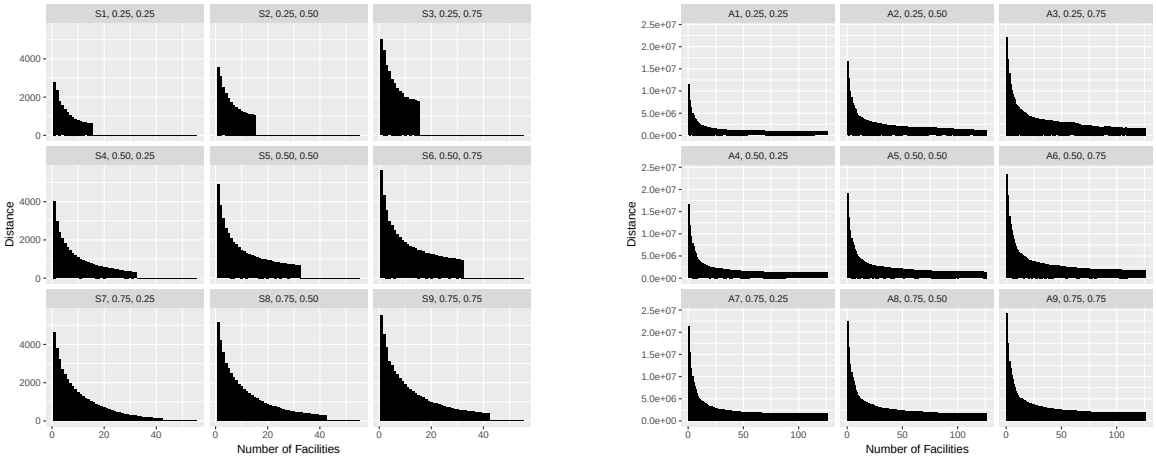
(b) Arkansas data set

Figure 4.7: Solutions obtained via CA. Subfigure headers: instance group, α, ω.

4.5.2 Comparison of Strategies

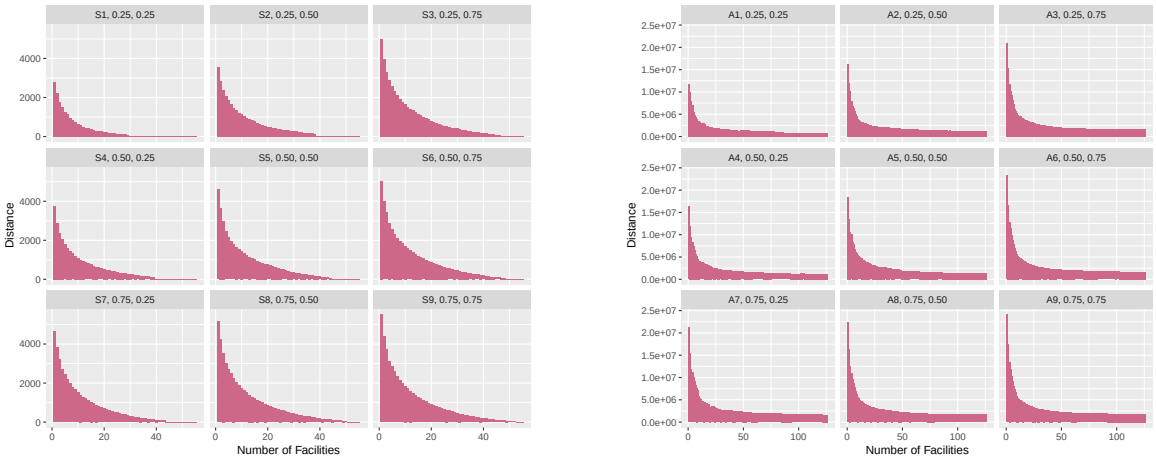
We begin this section with an illustration depicting how the solutions obtained by each strategy differ for an example instance from the Arkansas data set. We then provide a more comprehensive comparison among the strategies over all test instances from both the Swain and Arkansas data sets.

To illustrate the differences among the solutions produced by the three strategies for a single instance, consider instance A1-1 ($\alpha = 0.25$, $\omega = 0.25$) from the Arkansas data



(a) Swain data set (b) Arkansas data set

Figure 4.8: Solutions obtained via COV. Subfigure headers: instance group, α, ω.



(a) Swain data set (b) Arkansas data set

Figure 4.9: Solutions obtained via CMR. Subfigure headers: instance group, α, ω.

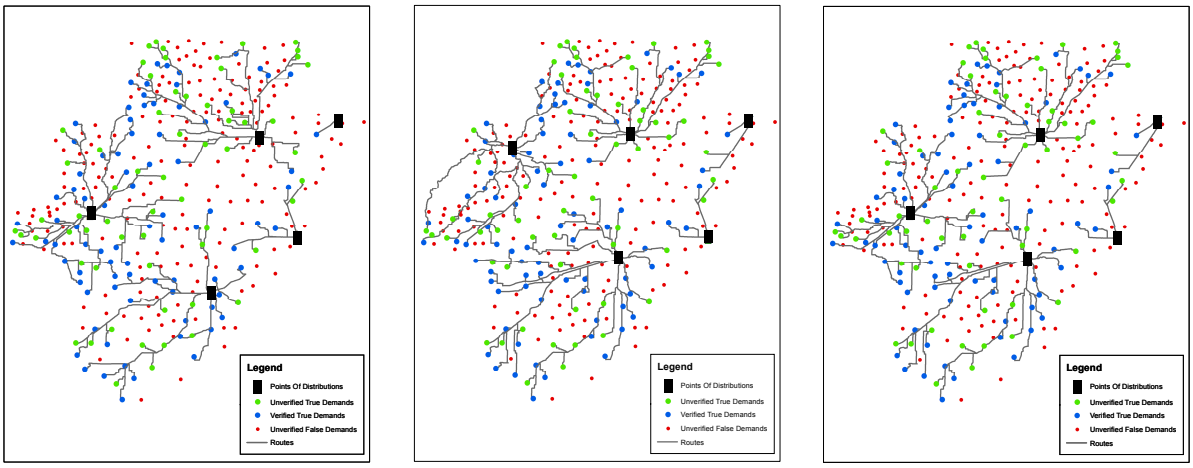
set. In this test instance, there are approximately 84 verified demand points, 63 unverified true demand points, and 188 unverified false demand points. Table 4.3 gives the facility locations opened by each of the three strategies for $P = 5$ and the total weighted distance of each of the three solutions. CMR performs best in this example with a total weighted distance of 4.6×10^7. Next is CA with total weighted distance 4.8×10^7. It is interesting that even with so much unverified data ($1 - \alpha = 0.75$) and a large proportion of it false

Table 4.3: POD locations selected by each strategy in example instance $A1$-1 from the Arkansas case study

Strategy	Facilities opened					Tot. weighted dist. from Problem 2
COV	14	28	72	109	111	5.1×10^7
CA	16	28	67	72	118	4.8×10^7
CMR	14	28	72	109	118	4.6×10^7

$(1 - \omega = 0.75)$, CA still outperforms COV. That is, considering the social data, even with a low accuracy rate, produced a better solution than ignoring it did. Figure 4.10 depicts the three solutions. The green squares with the green flags attached represent the facilities. Blue circles are verified demands, green are unverified true demands and red are unverified false demands. Connections follow the actual road network in the region and are shown for stage two allocations (explaining why no red false points are connected to facilities). It can be observed that POD locations 28 and 72 are opened by all three strategies (these are the right-most facilities in the figures), while the other three locations selected vary. Note that COV and CMR have four facilities in common, differing only according to the left-most facility (111 for COV, 118 for CMR) shown in the COV and CMR subfigures. It is reasonable to expect more similarity in facility placement between COV and CMR than with CA in this instance because the information considered during stage one is more similar between COV and CMR than with CA. During stage one, only the 84 verified demand points (blue) would have been considered by COV. Strategy CMR would have considered these 84 verified demand points (blue), plus 10 different scenarios including a random sampling of approximately 25% of the 251 unverified points (presumably, a mix of green and red), for a total of approximately 146 demand points in each of 10 scenarios. On the other hand, CA would have considered all 335 demand points (all blue, green and red) during stage one.

Figure 4.11 and Figure 4.12 provide comparisons of the average demand weighted



(a) CA (b) COV (c) CMR

Figure 4.10: Solutions from each strategy for instance A1 ($\alpha = 0.25$, $\omega = 0.25$) with $P = 5$.

distance among the three strategies for all Swain data set and Arkansas data set instances, respectively. Each contains nine subfigures representing the instance groups (the nine (α, ω) combinations). In each subfigure, the header gives the instance group and value of α and ω, the horizontal axis gives the number of facilities opened and the y-axis gives the total weighted distance. The color of each bar represents a decision strategy as indicated in the legend. The values reported are the average weighted distances over ten replications. For each instance, the bar corresponding to the best performing strategy is brought forward. For example, see the subfigure for instance group S3 in Figure 4.11. For $P = 2$, the pink bar indicates the CMR solution provides the lowest average weighted distance. The taller black bar behind it indicates how much higher the average weighted distance is for COV. Then for $P = 8$, the green bar indicates the CA solution provides the lowest average distance for 8 facilities. The fact that a pink bar isn't visible for $P = 8$ indicates CMR is nearly equivalent to one of the other two approaches in this case (the pink bar is underneath either the green

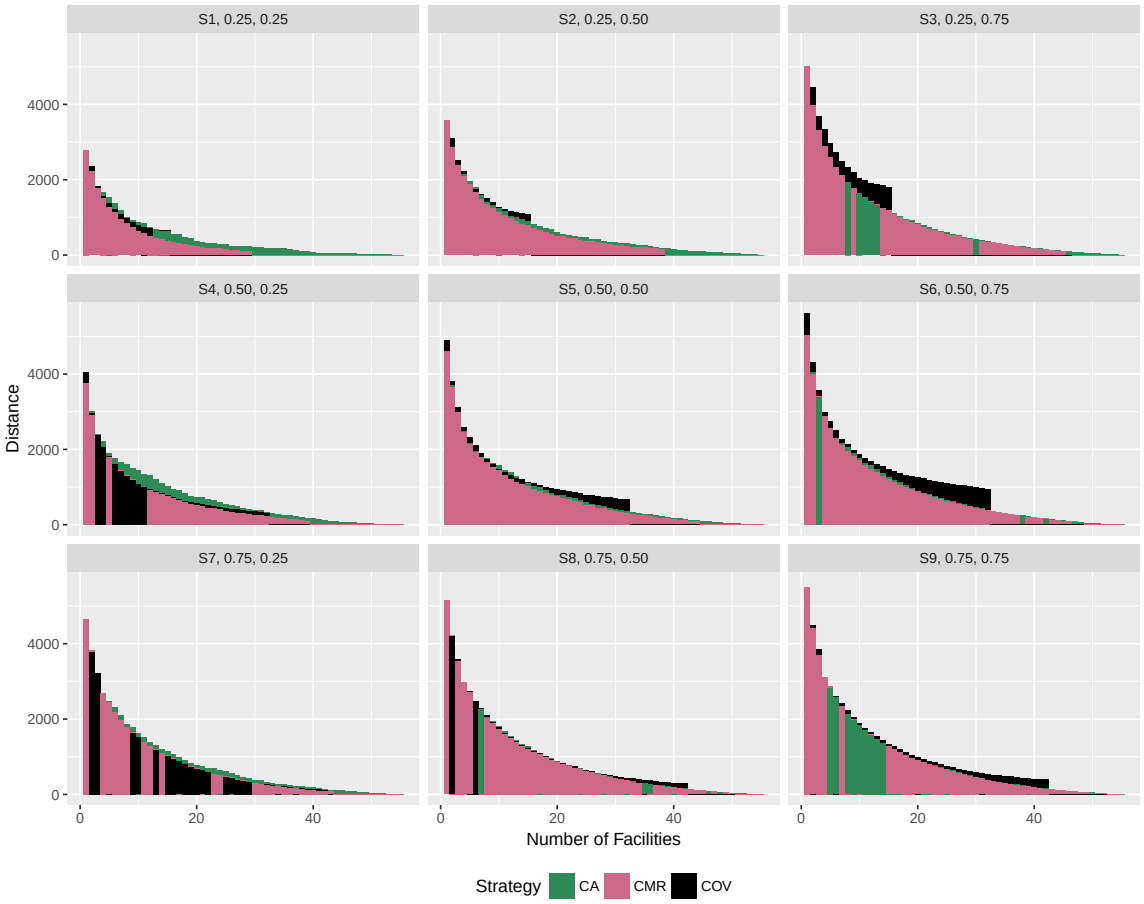


Figure 4.11: Comparison of solutions in Swain data set for the three different strategies. Subfigure headers: instance group, α ω.

or black bar). With this convention we can't be certain which one, but inspecting the trend in the subfigure would suggest it is more similar to CMR (pink) than COV (black). This graphing convention allows for a quick visual determination of the dominant approach across all instance groups. It is easily seen that CMR provides the best solution the majority of the time. This is true for both the Swain and Arkansas data sets. Because this is clear, we temporarily turn our attention to some of the minority cases where COV and CA are the top performing approaches.

When the size of the verified set is smallest ($\alpha = 0.25$), the COV strategy performs worse (or at least no better) than the other two decision strategies for most values of ω and

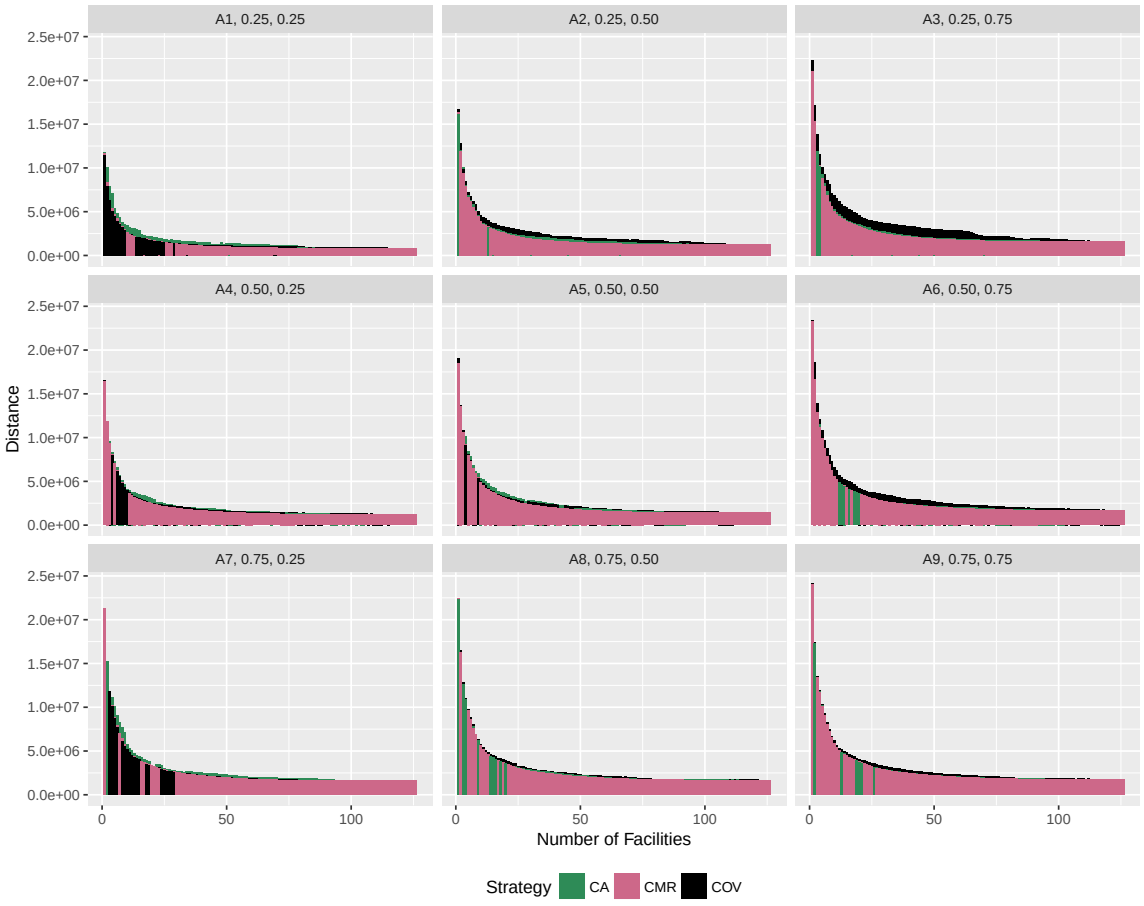


Figure 4.12: Comparison of solutions in Arkansas case study for the three different strategies. Subfigure headers: instance group, α ω.

numbers of facilities. In fact, it can be observed from the top rows of Figures 4.11 and 4.12 that the only time the black bar for COV is dominant when $\alpha = 0.25$ is for some values of P within instance group A1. In this instance group, $\omega = 0.25$, indicating that a majority of the unverified data is false. In this group, COV is the best approach for $P = 1, \ldots, 9$, $P = 14, \ldots, 25$ and $P = 29$. The weighted distance of the CMR solutions is 3.7% higher than COV, on average for these values of P in group A1. According to a paired t-test and a level of significance of 0.05, these differences are significant except for $P = 1$. For all other instance groups with $\alpha = 0.25$, CMR is the best approach for almost all P values. There are a few numbers of facilities for which CA is best (see, for example, $P = 10, \ldots, 13$ in S3). When the size of the verified set is largest ($\alpha = 0.75$), the absolute differences between the approaches tend to be less pronounced than when the verified set is small ($\alpha = 0.25$). This is likely because the information sets the three approaches are considering are most similar for these groups of instances. There are more values of P for which CA and COV produce lower weighted distances within groups with $\alpha = 0.75$ than other groups.

When the proportion of true data within the unverified set is smallest ($\omega = 0.25$), we rarely see CA be the top performing approach. In fact, it happens only once, for $P = 2$ in group A7. The size of the unverified set is small in this group ($\alpha = 0.75$), so a large proportion of false data has a lower impact than when there is more uncertain data to begin with. When the proportion of true data within the unverified set is largest ($\omega = 0.75$), we see CA be the top performing approach for at least some values of P for every applicable instance group (S7-S9 and A7-A9). In general, the absolute difference between COV and the other approaches is more pronounced for $\omega = 0.75$ than when $\omega = 0.25$ and the converse is true for CA.

Tables 4.4 and 4.5 provide summaries of the performance of each approach for each instance group. For both the Swain and Arkansas data sets, the best strategy (and therefore best solution) is identified for every instance. Then, the percent deviation in objective value is computed between each strategy and the best one for each instance. The middle group of columns in both tables reports the average percent deviation of each strategy from the best one, for each instance group. The best strategy for each instance group is indicated in bold. Additionally, for every instance group and value of P, a paired t-test is conducted over the 10 replicates to detect whether differences between the solutions produced by each pair of strategies are significant at the 0.05 level. The last set of columns in the two tables reports the percent of instances with significant differences between pairs of strategies. From these tables, it is again clear that CMR is the best approach in the majority of instances studied. It has the lowest average percent deviation from the best strategy in 17 out of 18 instances groups; the exception is S2. In 13 out of 18 instance groups, this average percent deviation is less than 1%. CA can only claim an average percent deviation under 1% in two instance groups (S3 and A9); COV cannot claim it for any instance group. Thus, a decision manager seeking a strategy that is robust across a wide variety of instance groups should adopt CMR.

While CA and COV are not robust strategies across all instance groups as is CMR, we still find it interesting to compare and contrast CA and COV with each other, as they represent the two "extreme" strategies (consider all of the unverified data or none of it). The results of this computational study suggest that among the two, COV should be used when the accuracy proportion of the unverified data is $\omega = 0.25$ and CA should be used when $\omega = 0.50$ and 0.75. That is, COV is preferred for S1, A1, S4, A4, S7 and A7, while CA is preferred for the remaining 12 instance groups. While it is no surprise that COV

Table 4.4: Summary of percent deviation from best strategy for instance groups $S1$-$S9$

				Avg % dev. from best strategy			% of differences that are significant		
Instance group #	α	ω	$\|V\|$	COV	CA	CMR	*COV,CA*	*COV,CMR*	*CA,CMR*
$S1$	0.25	0.25	15	24.66%	30.24%	**0.85%**	27%	93%	87%
$S2$	0.25	0.50	15	13.97%	7.14%	**0.18%**	60%	93%	73%
$S3$	0.25	0.75	15	25.26%	**0.49%**	0.85%	93%	93%	0%
$S4$	0.50	0.25	32	12.80%	46.85%	**6.07%**	69%	9%	94%
$S5$	0.50	0.50	32	32.45%	10.93%	**1.04%**	53%	97%	69%
$S6$	0.50	0.75	32	45.04%	3.86%	**0.43%**	94%	100%	63%
$S7$	0.75	0.25	42	7.46%	31.53%	**5.33%**	83%	5%	90%
$S8$	0.75	0.50	42	16.94%	3.90%	**0.36%**	52%	86%	69%
$S9$	0.75	0.75	42	31.51%	1.84%	**0.77%**	90%	90%	17%

performs better than CA when ω is small, we do find it interesting how tolerant the CA strategy is to inaccuracy of the uncertain data. These results suggest that as long as half of the data is accurate, it should be considered during stage one planning. And in fact, the threshold ω value for CA being the preferred approach may be less than 0.5; we did not test any values between 0.25 and 0.50. It is counterintuitive that the CA approach is not more highly dependent on data accuracy. In practical terms, there is less of a penalty for including the uncertain data up front, even if the majority of it turns out to be false, than there is for ignoring it altogether. It should be noted that the basis for recommending CA or COV in this discussion is knowledge of the value of ω. Realistically, the emergency manager will likely not have information regarding how much of the uncertain data is accurate (knowing this would probably require knowing precisely which data is true and false, negating the need for the approaches outlined in this paper). But, even though the manager may not know the precise value of ω, they may have some intuition regarding whether at least one-quarter of the data is accurate. Or, they may be able to investigate the accuracy of a small random sample of the uncertain data to approximate ω. For example, a platform such as Verily could be deployed to access the power and speed of crowdsourcing the verification process [85].

Table 4.5: Summary of percent deviation from best strategy for instance groups $A1$-$A9$

			Avg % dev. from best strategy			% of differences that are significant		
Instance group #	α	ω	COV	CA	CMR	*COV,CA*	*COV,CMR*	*CA,CMR*
$A1$	0.25	0.25	10.42%	21.88%	**1.98%**	84%	92%	93%
$A2$	0.25	0.50	26.82%	8.40%	**0.69%**	98%	97%	84%
$A3$	0.25	0.75	32.80%	4.44%	**0.24%**	100%	100%	90%
$A4$	0.50	0.25	8.43%	10.88%	**1.37%**	75%	26%	90%
$A5$	0.50	0.50	10.41%	9.61%	**0.73%**	83%	72%	90%
$A6$	0.50	0.75	17.46%	2.05%	**0.21%**	97%	97%	56%
$A7$	0.75	0.25	2.20%	10.28%	**0.96%**	43%	83%	98%
$A8$	0.75	0.50	6.54%	2.59%	**0.26%**	90%	56%	83%
$A9$	0.75	0.75	7.46%	0.79%	**0.14%**	94%	92%	40%

4.6 Conclusion And Future Work

This forth chapter presented a framework for incorporating a new type of uncertainty in POD location decisions. Specifically, the framework enables evaluating whether there is value in acting on user-generated data prior to its absolute verification when locating facilities for disaster relief. Three decision strategies that can be used by an emergency manager faced with a POD location decision for which both verified and unverified data are available were proposed. The strategies differ according to how the uncertain user-generated data is incorporated in the planning process.

The framework was illustrated via a computational study that compared the performance of the three decision strategies across a range of plausible disaster scenarios. The study yielded the following insights. First, as expected, the *Consider Only Verified* strategy is outperformed by the other strategies when the proportion of uncertain data is high, and/or the proportion of uncertain data that is true is high. Ignoring the uncertain data during the planning stage results in sub-optimal facility locations in these cases. Second, also as expected, the performance of the *Consider All* strategy suffers when a large proportion of the uncertain data is false. Considering the uncertain data in this case results in placing facilities further from the verified and true points than necessary. Finally, the *Consider Min-*

imax Regret strategy offers a tradeoff between the two extremes of considering none of the unverified data or all of it, and performs reasonably well across the test instances studied.

None of the decision strategies presented in this paper require knowledge of the likelihood that the uncertain data is true. In fact, a primary assumption in this study is that probabilistic information is *not* available for the uncertain data. Because of this, parameter ω used to describe the instances in the computational study, representing the proportion of uncertain data is true, will also not be available to the decision maker at the time of planning. The decision maker will only have knowledge of α, the proportion of the total demand set that is unverified. If the parameter ω were available, then strategy *Consider All* may be preferred in some situations (i.e., when ω is large, and in this study, greater than 0.75). However, only having knowledge of α, the *Consider Minimax Regret* strategy is recommended, with robust performance across all test instances studied.

It should be noted that the insights summarized here are not intended to comprise the basis of one-size-fits-all recommendations for user-generated data in emergency decision making. The computational study was included here primarily for demonstration purposes. The quality of insights the framework is able to provide will increase as larger studies, preferably with real disaster data sets, are conducted. We recommend such studies as an area for future work. Additionally, the use of scenario-based modeling for other types of emergency logistics planning decisions is recommended. For example, dynamic and capacitated variants of POD location problems could be studied.

5 Conclusion And Future Work

This paper presented three logistics problems. The first problem is a parallel machine scheduling problem with release dates, due dates, limited machine avibility and splitting jobs is discussed. It is an inbound continuous aid work humanitarian logistics problem. This problem is motivated by a problem faced by a large medical product supplier. They are interested in decreasing human labor costs in their warehouses through optimized employee scheduling. Specifically, an opportunity to reduce the total amount of time required to complete work inside the DCs each day exists by considering individual employee productivities for each task. The warehouse decision makers then expects to find a shchedule engine so that each day the total job processing time is minimized. The problem is unique as that it considers multiple factors at the same time when making decisions. Currently no paper has discussed all the factors at the same time.

To solve the problem, a mixed integer programming model is described. And two methods: the commerical server and a constructive heuristic are proposed to find solutions. 480 instances considering 5 different factors based on historical data are developed to verified these two methods. From the results, the commerical solver could solve the problem optimally most of the time while the heuristic only provides approximate solving that have an approximate 2% - 4% gaps compared with the optimal solutions. On the other hand, the heuristic runs much faster than the commerical server - an average of 10 seconds for heuristic to solve one instance compared with an average of 10 minutes for the commerical solver.

The second problem is a variant problem from Chapter **??** that is also motivated by

a large medical product supplier seeking to optimize labor costs within their warehouses. This problem extended to account for two additional practical considerations faced by the company: a delay that occurs when warehouse staff shift from one type of task to another and a fixed time window employee breaks during the employee's schedule. Other characteristics including release dates, due dates, limited machine avibility and splitting jobs is also involved in this discussed.

To solve this problem, a updated mixed integer programming model from chatper 2 is illustrated. A simulated anneal heuristic are proposed to find solutions. This SA heuristic includes two operators and by using the historical performance of our operators, the algorithm provide guidance on which move operators are applied in future iterations, where the more successful at finding improving solutions operators in the past are more likely to be selected. A total of 480 instances considering 7 different factors based on historical data are developed and tested. For the small instances, the heuristic provides solutions within an average 1.12% worse than the lower bound obtained by CPLEX. For the medium and large instances, the heuristic was able to solve them in a relatively reasonable time.

From the results, the commerical solver could solve the problem optimally most of the time while the heuristic only provides approximate solving that have an approximate 2% - 4% gaps compared with the optimal solutions. On the other hand, the heuristic runs much faster than the commerical server - an average of 10 seconds for heuristic to solve one instance compared with an average of 10 minutes for the commerical solver.

The last problem is a new facility location problem variant. In this problem, both verified data and unverified user-generated data are available for consideration during decision making. Three decision strategies that can be used by an emergency manager faced

with a POD location decision for which both verified and unverified data are available were proposed. The strategies differ according to how the uncertain user-generated data is incorporated in the planning process. A computational study and a case study that compared the performance of the three decision strategies across a range of plausible disaster scenarios is proposed. The study yielded the following insights. First, as expected, the *Consider Only Verified* strategy is outperformed by the other strategies when the proportion of uncertain data is high, and/or the proportion of uncertain data that is true is high. Ignoring the uncertain data during the planning stage results in sub-optimal facility locations in these cases. Second, also as expected, the performance of the *Consider All* strategy suffers when a large proportion of the uncertain data is false. Considering the uncertain data in this case results in placing facilities further from the verified and true points than necessary. Finally, the *Consider Minimax Regret* strategy offers a tradeoff between the two extremes of considering none of the unverified data or all of it, and performs reasonably well across the test instances studied.

A primary assumption in this study is that probabilistic information is *not* available for the uncertain data. Because of this, parameter ω used to describe the instances in the computational study, representing the proportion of uncertain data is true, will also not be available to the decision maker at the time of planning. The decision maker will only have knowledge of α, the proportion of the total demand set that is unverified. If the parameter ω were available, then strategy *Consider All* may be preferred in some situations (i.e., when ω is large, and in this study, greater than 0.75). However, only having knowledge of α, the *Consider Minimax Regret* strategy is recommended, with robust performance across all test instances studied.

It should be noted that the insights summarized here are not intended to comprise the basis of one-size-fits-all recommendations for user-generated data in emergency decision making. The computational study was included here primarily for demonstration purposes. The quality of insights the framework is able to provide will increase as larger studies, preferably with real disaster data sets, are conducted. We recommend such studies as an area for future work. Additionally, the use of scenario-based modeling for other types of emergency logistics planning decisions is recommended. For example, dynamic and capacitated variants of POD location problems could be studied.

Bibliography

[1] K. Vitasek, "Supply chain management terms and glossary, february 2010, published on pages of cscmp (council of supply chain management professionals)."

[2] R. A. Wilson, 21st Annual State of Logistics Report®: The Great Freight Recession. CSCMP, 2010.

[3] E. E. Blanco and J. Goentzel, "Humanitarian supply chains: a review," MIT Center for Transportation & Logistics, POMS, 2006.

[4] "Labor management systems to improve efficiency," http://www.logisticsmgmt.com/article/workforce_metamorphosis, accessed: 2015-11-20.

[5] "How do i measure employee productivity performance?" http://smallbusiness.chron.com/measure-employee-productivity-performance-1896.html, accessed: 2015-11-20.

[6] A. T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier, "An annotated bibliography of personnel scheduling and rostering," Annals of Operations Research, vol. 127, no. 1-4, pp. 21–144, 2004.

[7] M. Pfund, J. W. Fowler, and J. N. Gupta, "A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems," Journal of the Chinese Institute of Industrial Engineers, vol. 21, no. 3, pp. 230–241, 2004.

[8] M. G. Ravetti, G. R. Mateus, P. L. Rocha, and P. M. Pardalos, "A scheduling problem with unrelated parallel machines and sequence dependent setups," International Journal of Operational Research, vol. 2, no. 4, pp. 380–399, 2007.

[9] J.-F. Chen, "Scheduling on unrelated parallel machines with sequence-and machine-dependent setup times and due-date constraints," The International Journal of Advanced Manufacturing Technology, vol. 44, no. 11, pp. 1204–1212, 2009.

[10] K.-C. Ying and S.-W. Lin, "Unrelated parallel machines scheduling with sequence-and machine-dependent setup times and due date constraints," International Journal of Innovative Computing, Information and Control, vol. 8, no. 5, pp. 3279–3297, 2012.

[11] J. Bank and F. Werner, "Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties," Mathematical and Computer Modelling, vol. 33, no. 4, pp. 363–383, 2001.

[12] J. Valente and R. A. Alves, "A note on polynomially-solvable cases of common due date early-tardy scheduling with release dates," Investigação Operacional, vol. 24, no. 1, pp. 63–71, 2004.

[13] V. Suresh and D. Ghaudhuri, "Scheduling of unrelated parallel machines when machine availability is specified," Production Planning & Control, vol. 7, no. 4, pp. 393–400, 1996.

[14] T. E. Cheng, C.-J. Hsu, and D.-L. Yang, "Unrelated parallel-machine scheduling with deteriorating maintenance activities," Computers & Industrial Engineering, vol. 60, no. 4, pp. 602–605, 2011.

[15] S.-J. Yang, "Unrelated parallel-machine scheduling with deterioration effects and deteriorating multi-maintenance activities for minimizing the total completion time," Applied Mathematical Modelling, vol. 37, no. 5, pp. 2995–3005, 2013.

[16] R. Logendran and F. Subur, "Unrelated parallel machine scheduling with job splitting," IIE Transactions, vol. 36, no. 4, pp. 359–372, 2004.

[17] B. V. S. B. H. SÜRELİ, İ. İÇEREN, İ. P. M. ÇİZELGELEME, P. İ. G. ALGORİTMA-DOKUMA, and T. ÇİZELGELEME, "A genetic algorithm for the unrelated parallel machine scheduling problem with job splitting and sequence-dependent setup times-loom scheduling," 2013.

[18] G. Schmidt, "Scheduling with limited machine availability," European Journal of Operational Research, vol. 121, no. 1, pp. 1–15, 2000.

[19] A. Allahverdi, "The third comprehensive survey on scheduling problems with setup times/costs," European Journal of Operational Research, vol. 246, no. 2, pp. 345–378, 2015.

[20] A. Allahverdi, J. N. Gupta, and T. Aldowaisan, "A review of scheduling research involving setup considerations," Omega, vol. 27, no. 2, pp. 219–239, 1999.

[21] A. Allahverdi and H. Soroush, "The significance of reducing setup times/setup costs," European Journal of Operational Research, vol. 187, no. 3, pp. 978–984, 2008.

[22] M. J. P. Lopes and J. V. de Carvalho, "A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times," European journal of operational research, vol. 176, no. 3, pp. 1508–1527, 2007.

[23] Y.-K. Lin and F.-Y. Hsieh, "Unrelated parallel machine scheduling with setup times and ready times," International Journal of Production Research, vol. 52, no. 4, pp. 1200–1214, 2014.

[24] C. M. Joo and B. S. Kim, "Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability," Computers & Industrial Engineering, vol. 85, pp. 102–109, 2015.

[25] O. Avalos-Rosales, F. Angel-Bello, and A. Alvarez, "Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times," The International Journal of Advanced Manufacturing Technology, vol. 76, no. 9-12, pp. 1705–1718, 2015.

[26] L. Wang, S. Wang, and X. Zheng, "A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times," IEEE/CAA Journal of Automatica Sinica, vol. 3, no. 3, pp. 235–246, 2016.

[27] M. Afzalirad and J. Rezaeian, "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions," Computers & Industrial Engineering, vol. 98, pp. 40–52, 2016.

[28] D. Yilmaz Eroglu, H. C. Ozmutlu, and S. Ozmutlu, "Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times," International Journal of Production Research, vol. 52, no. 19, pp. 5841–5856, 2014.

[29] Y. Ma, C. Chu, and C. Zuo, "A survey of scheduling with deterministic machine availability constraints," Computers & Industrial Engineering, vol. 58, no. 2, pp. 199–211, 2010.

[30] X. Hu, J.-S. Bao, and Y. Jin, "Minimising makespan on parallel machines with precedence constraints and machine eligibility restrictions," International Journal of Production Research, vol. 48, no. 6, pp. 1639–1651, 2010.

[31] C. Zhao, M. Ji, and H. Tang, "Parallel-machine scheduling with an availability constraint," Computers & Industrial Engineering, vol. 61, no. 3, pp. 778–781, 2011.

[32] N. Hashemian, C. Diallo, and B. Vizvári, "Makespan minimization for parallel machines scheduling with multiple availability constraints," Annals of Operations Research, vol. 213, no. 1, pp. 173–186, 2014.

[33] C. Koulamas, "Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem," Naval Research Logistics (NRL), vol. 44, no. 1, pp. 109–125, 1997.

[34] D.-W. Kim, K.-H. Kim, W. Jang, and F. F. Chen, "Unrelated parallel machine scheduling with setup times using simulated annealing," Robotics and Computer-Integrated Manufacturing, vol. 18, no. 3-4, pp. 223–231, 2002.

[35] W.-C. Lee, C.-C. Wu, and P. Chen, "A simulated annealing approach to makespan minimization on identical parallel machines," The International Journal of Advanced Manufacturing Technology, vol. 31, no. 3-4, pp. 328–334, 2006.

[36] P. Damodaran and M. C. Vélez-Gallego, "A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times," Expert Systems with Applications, vol. 39, no. 1, pp. 1451–1458, 2012.

[37] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," Journal of global optimization, vol. 6, no. 2, pp. 109–133, 1995.

[38] S.-W. Lin and K.-C. Ying, "A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems," International Journal of Production Research, vol. 53, no. 4, pp. 1065–1076, 2015.

[39] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi et al., "Optimization by simulated annealing," science, vol. 220, no. 4598, pp. 671–680, 1983.

[40] L. F. Williams Jr, "A modification to the half-interval search (binary search) method," in Proceedings of the 14th annual Southeast regional conference. ACM, 1976, pp. 95–101.

[41] I. F. of Red Cross and R. C. Societies, "World Disaster Report 2012 Data Dashboard," Available from http://www.ifrcmedia.org/assets/pages/wdr2012/data/index.html, accessed July 17, 2013.

[42] E. Ferris, D. Petz, and C. Stark, "The year of recurring disasters: a review of natural disasters in 2012," The Brookings Institution, 1775 Massachusetts Ave, NW, Washington, DC 20036, Report, March 2013, available from http://www.brookings.edu/research/reports/2013/03/natural-disaster-review-ferris. Accessed July 17, 2013.

[43] I. F. of Red Cross and R. C. Societies, "Preparing for disasters: introduction to logistics," Available from http://w3.ifrc.org/what/disasters/responding/elements/logistics.asp, accessed July 17, 2013.

[44] L. E. de la Torre, I. S. Dolinskaya, and K. R. Smilowitz, "Disaster relief routing: Integrating research and practice," Socio-Economic Planning Sciences, vol. 46, pp. 88–97, 2012.

[45] F. E. M. Agency, "Independent Study (IS)-26: Guide to Points of Distribution," Available from http://training.fema.gov/is/courseoverview.aspx?code=is-26, August 2010, accessed January 8, 2014.

[46] A. B. Milburn, C. Rainwater, O. Boudhoum, and S. Young, "Models for disaster relief shelter location and supply routing," Mack Blackwell National Rural Transportation Center, University of Arkansas, 4190 Bell Engineering Center, Fayetteville, AR 72701, Final report for MBTC DOT 3028, February 2013, available from http://ntl.bts.gov/lib/47000/47100/47185/MBTC_DOT_3028.pdf. Accessed January 8, 2014.

[47] Y. S. Su, C. W. III, and Z. Thorkildsen, "Social media in the emergency management field," CNA, Available from http://www.cna.org/sites/default/files/research/SocialMedia_EmergencyManagement.pdf, Report, June 2013, accessed July 17, 2013.

[48] A. R. C. Website, "Press release, August 31, 2012: More Americans using mobile apps in emergencies," Available from http://rdcrss.org/1i2noGo, accessed July 17, 2013.

[49] S. Vieweg, A. Hughes, K. Starbird, and L. Palen, "Microblogging during two natural hazard events: What Twitter may contribute to situational awareness," in Proceedings of the 28th International Conference on Human Factors in Computing Systems, Atlanta, GA, USA, April 2010, pp. 1079–1088.

[50] K. Stephens, "Five SMEM observations and recommendations from Hurricane Sandy [Web log comment]," Available from http://idisaster.wordpress.com/2012/11/06/five-smem-observations-and-recommendations-from-hurricane-sandy/, November 6, 2012, accessed July 18, 2013.

[51] A. R. C. Website, "Press release: The American Red Cross and Dell launch first of its kind social media digital operations center for humanitarian relief," Available from http://rdcrss.org/1fimoc8, 2013, accessed July 17, 2013.

[52] D. Gross, "Man faces fallout for spreading false Sandy reports on Twitter," Available at http://edition.cnn.com/2012/10/31/tech/social-media/sandy-twitter-hoax, October 31, 2012, accessed July 22, 2013.

[53] B. Balcik and B. M. Beamon, "Facility location in humanitarian relief," International Journal of Logistics: Research and Applications, vol. 11, no. 2, pp. 101–121, 2008.

[54] G. Kovács and K. M. Spens, "Humanitarian logistics in disaster relief operations," International Journal of Physical Distribution & Logistics Management, vol. 37, no. 2, pp. 99–114, 2007.

[55] C. Rawls and M. Turnquist, "Pre-positioning of emergency supplies for disaster response," Transportation Research Part B: Methodological, vol. 44, no. 4, pp. 521–534, 2010.

[56] W. Yushimoto, M. Jaller, and S. Ukkusuri, "A Voronoi-based heuristic algorithm for locating distribution centers in disasters," Networks and Spatial Economics, vol. 12, no. 1, pp. 21–39, 2012.

[57] H. Jia, F. Ordonez, and M. Dessouky, "A modeling framework for facility location of medical services for large-scale emergencies," IIE Transactions, vol. 39, no. 1, pp. 41–55, 2007.

[58] F. S. Salman and S. Gul, "Deployment of field hospitals in mass casualty incidents," Computers & Industrial Engineering, vol. 74, no. 1, pp. 37–51, 2014.

[59] M. Jaller and J. Holguín-Veras, "Locating points of distribution with social costs considerations," 90^{th} Transportation Research Board Annual Meeting, 2011.

[60] M. J. Widener and M. W. Horner, "A hierarchical approach to modeling hurricane disaster relief goods distribution," Journal of Transport Geography, vol. 19, no. 4, pp. 821–828, 2011.

[61] N. Gormez, M. Koksalan, and F. Salman, "Locating disaster response facilities in istanbul," Journal of the Operational Research Society, vol. 62, no. 7, pp. 1239–1252, 2011.

[62] S. H. Owen and M. S. Daskin, "Strategic facility location: a review," European Journal of Operational Research, vol. 111, pp. 423–447, 1998.

[63] L. V. Snyder, "Facility location under uncertainty: a review," IIE Transactions, vol. 38, no. 7, pp. 547–564, 2006.

[64] A. Warszawski, "Multi-dimensional location problems," Journal of the Operational Research Society, vol. 24, no. 2, pp. 165–179, 1973.

[65] Z.-J. M. Shen, "A multi-commodity supply chain design problem," IIE Transactions, vol. 37, no. 8, pp. 753–762, 2005.

[66] P. Meier, "Crowdsourced verification for disaster response [Web log comment]," Available from http://irevolution.net/2013/02/19/verily-crowdsourcing-evidence/, February 19, 2013, accessed July 18, 2013.

[67] M. Imran, C. C. J. Lucas, P. Meier, and S. Viewig, "AIDR: Artificial Intelligence for Disaster Response," in Proceedings of the Companion Volume of the 23th International Conference on World Wide Web, 2014.

[68] P. Meier, "MicroMappears: Toward next generation humanitarian technology [Web log comment]," Available from http://irevolution.net/2014/12/12/micromappers-humanitarian-technology/, December 12, 2014, accessed July 7, 2015.

[69] P. Gundecha and H. Liu, "Mining social media: A brief introduction," Tutorials in Operations Research, vol. 1, no. 4, 2012.

[70] K. Starbird, "Situational awareness in mass emergency: A behavioral and linguistic analysis of microblogged communications," University of Colorado at Boulder, PhD dissertation, 2012.

[71] M. Imran, S. Elbassuoni, C. Castillo, F. Diaz, and P. Meier, "Extracting information nuggets from disaster-related messages in social media," in Proceedings of the 10th International ISCRAM Conference. Baden-Baden, Germany: International Association for Information Systems for Crisis Response and Management, May 2013.

[72] ——, "Practical extraction of disaster-relevant information from social media," in WWW 2013 Companion. Rio de Janeiro, Brazil: International World Wide Web Conference Committee, May 2013.

[73] K. Starbird, G. Muzny, and L. Palen, "Learning from the crowd: Collaborative filtering techniques for identifying on-the-ground Twitterers during mass disruptions," in Proceedings of the 9th International ISCRAM Conference. Vancouver, Canada: International Association for Information Systems for Crisis Response and Management, April 2012.

[74] B. Truong, C. Caragea, A. Squicciarini, and A. Tapia, "Identifying valuable information from Twitter during natural disasters," in Proceedings of the 77th annual ASIS&T Meeting. Seattle, WA, USA: Association for Information Science and Technology, October 31-November 4, 2014.

[75] S. Verma, S. Vieweg, W. Corvey, L. Palen, J. Martin, M. Palmer, A. Schram, and K. Anderson, "NLP to the rescue? Extracting 'situational awareness' Tweets during mass emergency," in Proceedings of the 5th International AAAI Conference on Weblogs and Social Media, Barcelona, Spain, July 2011.

[76] V. Belair-Gagnon, Social media at BBC News: The re-making of crisis reporting. Routledge, 2015.

[77] T. Holderness and E. Turpin, "Petajakarta.org: Assessing the role of social media for civic co-management during monsoon flooding in Jakarta, Indonesia," SMART Infrastructure Facility, University of Wollongong, Wollongong, Australia, White paper, June 2015.

[78] L. Derczynski and K. Bontcheva, "Pheme: Veracity in digital social networks," in Proceedings of the 10th Joint ACL - ISO Workshop on Interoperable Semantic Association (ISA), Reykjavik, Iceland, 2014.

[79] R. W. Swain, "A decomposition algorithm for a class of facility location problems," Ph.D. dissertation, Cornell University, Ithaca, NY, USA, 1971.

[80] A. Shapiro and A. Philpott, "A tutorial on stochastic programming," http://www.isye.gatech.edu/people/faculty/Alex_Shapiro/TutorialSP.pdf, 2007.

[81] U. C. Bureau, "Arkansas: 2000 population and housing unit counts," 2013.

[82] U. D. of Commerce, "National county subdivisions gazetteer," Available from www.census.gov/tiger/tms/gazetteer/cousub2k.txt, 2012.

[83] Educationbug, "Public schools by county," Available from http://arkansas.educationbug.org/public-schools/, 2012.

[84] ESRI, "Arcgis desktop 9.3 help," Available from http://webhelp.esri.com/arcgisdesktop, 2012.

[85] P. Meier, "Crowdsourced verification for disaster response [web log comment]," Available from https://irevolutions.org/2014/07/17/live-verily/, July 2014, accessed April 2, 2017.

A Appendix

Table A.1: Demand locations and magnitudes for Arkansas case study

Twonship	Demand	Lat.	Long.	Twonship	Demand	Lat.	Long.
Arkansas	67	34.06	-91.36	Lester	11901	35.91	-90.46
Barton	204	34.35	-91.66	Little Texas	11901	35.77	-90.94
Bayou Meto	179	34.21	-91.48	Maumelle	2192	35.75	-90.55
Brewer	45	34.26	-91.61	Nettleton	493	35.81	-90.59
Chester	263	34.06	-91.23	Powell	244	35.93	-90.65
Crockett	126	34.42	-91.23	Prairie	2289	35.77	-90.44
Garland	131	34.33	-91.51	Promised Land	8158	35.84	-90.98
Gum Pond	8264	34.53	-91.54	Taylor	2060	35.73	-90.48
Henton	541	34.43	-91.67	Texas	140	35.81	-90.93
Keaton	864	34.46	-91.33	Black Oak	218	35.39	-90.42
La Grue	4134	34.28	-91.34	Bob Ward	259	35.07	-90.34
McFall	113	34.53	-91.43	Fogleman	359	35.39	-90.22
Mill Bayou	485	34.42	-91.46	Jackson	489	35.21	-90.33
Morris	556	34.43	-91.57	Jasper	1182	35.22	-90.22
Point Deluce	211	34.20	-91.28	Lucas	1535	34.90	-90.33
Prairie	571	34.27	-91.16	Mississippi	1368	35.14	-90.18
Stanley	722	34.16	-91.40	Mound City	11934	35.22	-90.14
Bennett-Lemmons	659	36.42	-90.41	Proctor	827	35.09	-90.24
Bradshaw-Haywood	187	36.34	-90.27	Tyronza	13846	35.25	-90.45
Brown-Carpenter	235	36.46	-90.70	Wappanocca	13846	35.31	-90.26
Cache-Wilson	505	36.32	-90.58	Bedford	318	35.22	-90.86
Chalk Bluff-Liddell	258	36.46	-90.20	Brushy Lake	882	35.30	-90.96
Clark	285	36.29	-90.70	Coldwater	3771	35.39	-90.63
Cleveland-North Kilgore	597	36.45	-90.55	Ellis	830	35.20	-90.93
East Oak Bluff-Blue Cane	232	36.25	-90.23	Fair Oaks	535	35.20	-91.01
Gleghorn-South Kilgore	163	36.38	-90.60	Hickory Ridge	445	35.40	-90.96
Johnson	530	36.32	-90.39	Mitchell	462	35.40	-90.78
Knob	245	36.29	-90.44	Searcy	520	35.33	-90.77
Nelson	263	36.36	-90.73	Smith	225	35.20	-90.67
North St. Francis	2611	36.40	-90.18	Twist	658	35.40	-90.52
Payne-Swain	105	36.31	-90.12	Tyronza	1251	35.25	-90.56
Pollard	579	36.44	-90.28	Wynne	1453	35.21	-90.79
South St. Francis	1709	36.36	-90.18	Blue Cane	2061	36.17	-90.26
West Oak Bluff	2710	36.27	-90.31	Breckenridge	38	36.23	-90.51
Big Creek	4303	35.92	-90.80	Bryan	1969	36.08	-90.76
Black Oak	2531	35.78	-90.35	Cache	9804	36.06	-90.66
Brookland	2528	35.92	-90.54	Clark	70	36.07	-90.47

Table A.2: Demand locations and magnitudes for Arkansas case study (cont.)

Twonship	Demand	Lat.	Long.	Twonship	Demand	Lat.	Long.
Buffalo	1989	35.93	-90.35	Collier	1238	36.00	-90.50
Gilkerson	3277	35.77	-90.80	Crowley	241	36.21	-90.62
Greenfield	2191	35.74	-90.66	Evening Shade	1112	36.14	-90.71
Herndon	1202	35.94	-90.72	Friendship	8481	36.14	-90.44
Jonesboro	11901	35.85	-90.69	Hays	8481	36.04	-90.38
Lake City	11901	35.82	-90.45	Hopewell	439	36.24	-90.38
Hurricane	306	36.20	-90.43	Cow Lake	46	35.42	-91.10
Jones	70	36.22	-90.74	Glaize	2186	35.48	-91.41
Lake	768	36.10	-90.34	Glass	526	35.84	-91.11
Main Shore	146	36.00	-90.42	Grubbs	93	35.66	-91.10
Poland	311	36.00	-90.62	Jefferson	295	35.68	-91.28
Reynolds	1699	36.17	-90.35	Richwoods	442	35.55	-91.08
St. Francis	504	35.99	-90.56	Union	795	35.59	-91.26
Salem	283	36.00	-90.71	Village	1179	35.64	-91.18
Shady Grove	283	36.00	-90.80	Annieville	616	36.15	-91.25
Spring Grove	1144	36.05	-90.56	Ashland	1035	35.96	-91.02
Sugar Creek	84	36.11	-90.63	Black River	366	36.05	91.15
Union	977	36.14	-90.54	Black Rock	7789	36.13	-91.13
Walnut Corner	984	36.04	-90.79	Boas	1637	36.03	-90.98
Ashley	199	35.85	-91.63	Cache	344	36.11	-90.84
Barren	6031	35.90	-91.54	Campbell	258	36.10	-90.93
Big Bottom-Wycough-Logan	697	35.71	-91.39	Dent	506	36.18	-91.20
Black River-Marshell	2467	35.84	-91.30	Dowell	1136	35.92	-90.93
Cushman-Union	125	35.87	-91.76	Duty	2774	36.07	-91.07
Departee	1103	35.56	-91.43	Eaton	148	36.04	-91.21
Dota	1009	35.83	-91.41	Flat Creek	5093	36.11	-91.22
Fairview	1097	35.57	-91.64	Jesup	883	36.03	-91.32
Gainsboro	175	35.82	-91.52	Lawrence	320	35.99	-91.11
Greenbrier	791	35.74	-91.75	Marion	519	35.92	-91.11
Hill	215	35.55	-91.52	Morgan	258	35.96	-91.23
Huff	641	35.65	-91.62	Promised Land	117	36.00	-90.88
Jefferson	1140	35.91	-91.67	Reeds Creek	148	35.94	-91.31
Liberty	780	35.58	-91.75	Richwoods	248	36.13	-91.00
McHue	1312	35.71	-91.65	Spring River	350	36.19	-91.30
Magness	288	35.70	-91.49	Strawberry	555	36.10	-91.30
Moorefield	508	35.77	-91.57	Thacker	502	36.24	-91.26
Oil Trough	183	35.61	-91.47	Big Creek	818	34.69	-90.99
Relief	632	35.65	-91.73	Council	120	34.82	-90.48
Rosie	3082	35.64	-91.56	Fleener	380	34.86	-91.01
Ruddell	358	35.79	-91.68	Hampton	323	34.78	-91.02
Salado	1814	35.72	-91.60	Hardy	432	34.71	-90.58
Washington	536	35.80	-91.81	Independence	87	34.76	-90.71

Table A.3: Demand locations and magnitudes for Arkansas case study (cont.)

Twonship	Demand	Lat.	Long.	Twonship	Demand	Lat.	Long.
White River	315	35.74	-91.51	Oak Forest	83	34.78	-90.89
Barren	461	35.49	-91.54	Richland	366	34.68	-90.76
Bateman	7775	35.59	-91.33	St. Francis	886	34.79	-90.56
Bird	792	35.76	-91.18	Spring Creek	4	34.70	-90.89
Breckenridge	763	35.44	-91.22	Texas	6360	34.85	-90.86
Bryan	724	35.50	-91.29	Union	412	34.87	-90.74
Cache	824	35.52	-91.17	Big Lake	864	35.85	-90.20
Bowen	1904	35.97	-90.05	Spring Creek	190	34.53	-90.77
Burdette	610	35.81	-89.96	Tappan	9306	34.34	-90.88
Canadian	472	35.92	-89.74	Bolivar	64	35.66	-90.99
Carson	413	35.61	-90.02	Dobson	626	35.66	-90.99
Chickasawba	4156	35.91	-89.89	Greenfield	372	35.66	-90.72
Dyess	4630	35.59	-90.23	Greenwood	5382	35.62	-90.37
Fletcher	324	35.78	-89.90	Little River	1660	35.51	-90.47
Golden Lake	484	35.52	-90.06	Lunsford	1759	35.57	-90.55
Half Moon Lake	335	35.91	-90.05	Owen	4429	35.51	-90.93
Hector	10121	35.84	-90.06	Scott	225	35.49	-90.72
Little River	10121	35.72	-90.20	Tyronza	1243	35.48	-90.34
McGavock	933	35.48	-90.13	West Prairie	2828	35.63	-90.91
Monroe	1754	35.69	-90.02	Willis	4232	35.66	-90.52
Neal	1085	35.95	-90.22	Belcher	404	34.54	-91.63
Scott	660	35.53	-90.15	Bullard	746	34.97	-91.63
Whitton	837	35.49	-90.25	Calhoun	1106	34.98	-91.44
Brinkley	1169	34.87	-91.23	Center	1168	34.85	-91.64
Brown	728	34.90	-91.31	Des Arc	947	35.05	-91.50
Cache	10034	34.72	-91.31	Hazen	8263	34.78	-91.59
Cleburne	2607	34.56	-91.10	Hickory Plain	83	34.99	-91.75
Cypress Ridge	645	34.75	-91.13	Lower Surrounded Hill	152	34.81	-91.41
Dixon	385	34.86	-91.14	Roc Roe	292	34.64	-91.47
Duncan	3722	34.60	-91.23	Tyler	396	34.68	-91.62
Greenfield	4	34.96	-91.18	Union	227	34.91	-91.65
Hindman	1875	34.66	-91.08	Upper Surrounded Hill	1691	34.89	-91.41
Jackson	112	34.52	-91.18	Watensaw	482	34.76	-91.47
Keevil	309	34.78	-91.24	White River	623	34.94	-91.54
Montgomery-Smalley	241	34.42	-91.08	Baker	361	36.46	-91.13
Pine Ridge	938	34.69	-91.18	Bristow	210	36.18	-91.00
Raymond	240	34.63	-91.12	Butler	73	36.17	-91.11
Richland	94	34.95	-91.24	Columbia	74	36.37	-90.93
Roc Roe	189	34.62	-91.35	Current River	1195	36.30	-90.84
Big Creek	217	34.48	-90.89	Dalton	2119	36.43	-91.18
Cleburne	156	34.61	-90.69	Demun	45	36.26	-90.98
Cleveland	91	34.48	-90.97	East Roanoke	234	36.19	-91.05
Cypress	76	34.48	-91.02	Eleven Point	39	36.36	-91.07
Hickory Ridge	160	34.59	-90.94	Foster	750	36.34	-90.98

Table A.4: Demand locations and magnitudes for Arkansas case study (cont.)

Twonship	Demand	Lat.	Long.	Twonship	Demand	Lat.	Long.
Hicksville	322	34.59	-91.02	Ingram	428	36.42	-90.99
Hornor	473	34.54	-90.68	Jackson	238	36.42	-91.09
Lake	632	34.41	-90.68	Janes Creek	6435	36.33	-91.22
Marion	194	34.60	-90.87	Little Black	419	36.45	-90.82
Mooney	175	34.19	-90.97	O'Kean	248	36.20	-90.84
St. Francis	1535	34.54	-90.60	Reyno	471	36.35	-90.77
Richardson	207	36.40	-90.87	Francure	424	35.14	-91.49
Running Lake	226	36.31	-90.88	Garner	387	35.10	-91.75
Shiloh	582	36.30	-91.06	Gravel Hill	177	35.25	-91.98
Siloam	520	36.47	-90.92	Gray	427	35.27	-91.80
Spring River	274	36.24	-91.17	Gum Springs	628	35.22	-91.76
Union	478	36.45	-91.30	Guthrie	411	35.39	-91.67
Warm Springs	635	36.47	-91.05	Harrison	542	35.30	-91.65
Water Valley	75	36.32	-91.12	Hartsell	206	35.47	-91.68
West Roanoke	851	36.23	-91.09	Higginson	407	35.18	-91.72
Wiley	353	36.19	-90.91	Jackson	233	35.49	-91.75
Black Fish	279	34.94	-90.56	Jefferson	7917	35.19	-92.08
Franks	103	34.95	-90.73	Joy	7917	35.30	-91.96
Garland	225	34.95	-90.46	Kensett	2540	35.22	-91.67
Goodwin	280	34.97	-90.99	Kentucky	732	35.32	-92.06
Griggs	762	35.07	-90.61	Liberty	4306	35.41	-91.45
Heth	104	35.06	-90.46	McRae	430	35.10	-91.81
Johnson	131	35.11	-90.71	Marion	777	35.35	-91.86
L'Anguille	884	35.09	-90.96	Marshall	293	35.24	-92.06
Madison	2140	35.01	-90.80	Mount Pisgah	282	35.32	-91.85
Prairie	593	34.96	-90.90	Red River	351	35.19	-91.60
Telico	957	35.10	-90.82	Royal	1716	35.10	-92.03
Wheatley	762	34.95	-91.10	Russell	841	35.37	-91.48
Albion	1858	35.34	-91.80	Union	1368	35.07	-91.89
Antioch	772	35.14	-91.94	Velvet Ridge	956	35.41	-91.57
Bald Knob	8465	35.29	-91.56	Walker	285	35.11	-91.70
Big Creek	8465	35.42	-91.80	Augusta	656	35.28	-91.33
Cadron	1354	35.34	-91.95	Barnes	76	35.32	-91.16
Cane	2211	35.19	-91.86	Cache	384	35.16	-91.22
Chrisp	431	35.12	-91.88	Caney	503	35.04	-91.13
Clay	223	35.38	-91.75	Cotton Plant	448	35.04	-91.26
Cleveland	368	35.16	-92.04	Dent	5145	35.23	-91.09
Coffey	3383	35.19	-91.97	De View	656	35.22	-91.18
Coldwell	1087	35.41	-91.61	Franks	156	35.13	-91.09
Crosby	289	35.30	-91.85	Freeman	3018	34.97	-91.33
Cypert	953	35.26	-91.46	Garden	222	35.08	-91.40
Denmark	606	35.48	-91.62	Point	167	35.15	-91.36
Des Arc	480	35.27	-91.90	Pumpkin Bend	299	35.30	-91.09
Dogwood	68	35.10	-91.61	White River	1108	35.40	-91.31

Table A.5: POD locations in Arkansas case study

School	Lat.	Long.	School	Lat.	Long.
Dewitt Middle School	34.30	-91.34	Marmaduke High School	36.19	-90.39
Gillett High School	34.12	-91.38	Oak Grove Middle School	36.06	-90.50
Stuttgart Junior High School	34.48	-91.56	Paragould Junior High	36.06	-90.51
Dewitt High School	34.28	-91.35	Green Co. Tech Jr. High School	36.07	-90.56
Humphrey High School	34.42	-91.71	Paragould High School	36.06	-90.51
Meekins Middle School	34.50	-91.55	Batesville High School	35.76	-91.62
Stuttgart High School	34.48	-91.56	Batesville Middle School	35.78	-91.65
Piggott High School	36.38	-90.18	Southside High School	35.70	-91.63
Rector High School	36.26	-90.30	Batesville Junior High School	35.75	-91.62
Corning High School	36.42	-90.58	Cord-Charlotte High School	35.82	-91.44
Annie Camp Jr. High School	35.83	-90.73	Cushman High School	35.87	-91.76
Bay High School	35.75	-90.57	Midland High School	35.55	-91.62
Brookland High School	35.91	-90.58	Newark High School	35.72	-91.45
Nettleton High School	35.84	-90.70	Southside Middle School	35.70	-91.63
Nettleton Junior High School	35.84	-90.70	Sulphur Rock High School	35.75	-91.50
Riverside High School	35.82	-90.44	Newport High School	35.60	-91.27
Westside High School	35.84	-90.70	Mccrory High School	35.26	-91.20
Buffalo Is. Central HS	35.89	-90.34	Newport Junior High School	35.60	-91.27
Douglas Macarthur Jhs	35.82	-90.69	Swifton High School	35.82	-91.13
Jonesboro High School	35.82	-90.71	Tuckerman High School	35.74	-91.20
Riverside Jr. High School	35.76	-90.34	Lynn High School	36.01	-91.25
Westside Middle School	35.84	-90.70	River Valley High School	35.97	-91.32
Valley View High School	35.78	-90.74	Sloan-Hendrix High School	36.20	-91.19
East Junior High School	35.16	-90.17	Walnut Ridge High School	36.06	-90.95
Marion Middle School	35.20	-90.20	Black Rock High School	36.11	-91.10
Turrell High School	35.38	-90.26	Hoxie High School	36.05	-90.98
West Memphis High School	35.15	-90.20	Lee High School	34.77	-90.76
Wonder Junior High School	35.15	-90.18	Anna Strong Middle School	34.77	-90.76
Crawfordsville High School	35.23	-90.33	Academic Center Of Excellence	35.70	-89.97
Earle High School	35.28	-90.47	Armorel High School	35.92	-89.80
Marion High School	35.20	-90.20	Blytheville High School	35.93	-89.91
Marion Junior High School	35.20	-90.20	Blytheville Middle School	35.93	-89.91
West Junior High School	35.15	-90.19	Buffalo Is. Central Jhs	35.94	-90.26
Parkin High School	35.26	-90.55	Osceola Junior High School	35.70	-89.98
Wynne High School	35.23	-90.78	Gosnell High School	35.96	-89.97
Wynne Junior High School	35.23	-90.78	Manila High School	35.88	-90.16
Cross County High School	35.40	-90.81	Osceola High School	35.71	-90.01
Wynne Intermediate School	35.23	-90.78	Rivercrest High School	35.57	-90.04
Delaplaine High School	36.23	-90.73	Brinkley High School	34.89	-91.19
Greene Co. Tech High School	36.06	-90.52	Clarendon High School	34.70	-91.31

Table A.6: POD locations in Arkansas case study(cont.)

School	Lat.	Long.	School	Lat.	Long.
Holly Grove High School	34.60	-91.20	Hughes High School	34.95	-90.47
Central High School	34.54	-90.63	Palestine-Wheatley Junior High	34.92	-91.11
Marvell High School	34.56	-90.91	Palestine-Wheatley Senior High	34.97	-90.90
Miller Junior High School	34.56	-90.66	Forrest City Jr. High	35.02	-90.79
Barton High School	34.60	-90.75	Palestine-Wheatley Middle Sch.	34.91	-91.11
C.v. White High School	34.42	-90.80	Lincoln Middle School	35.01	-90.79
Elaine High School	34.31	-90.85	Ahlf Junior High School	35.25	-91.74
Kipp:delta College Prep School	34.52	-90.59	Bald Knob High School	35.31	-91.57
East Poinsett Co. High School	35.61	-90.34	Beebe Middle School	35.07	-91.90
Harrisburg High School	35.56	-90.73	Judsonia Middle School	35.28	-91.64
Marked Tree High School	35.53	-90.42	Kensett Middle School	35.23	-91.67
Weiner High Schoo	35.62	-90.91	Riverview High School	35.25	-91.69
Harrisburg Middle School	35.56	-90.72	Rose Bud High School	35.33	-92.07
Trumann High School	35.67	-90.52	White Co. Central High School	35.40	-91.68
Des Arc High School	34.98	-91.52	Bald Knob Middle School	35.31	-91.57
Devalls Bluff High School	34.78	-91.46	Beebe High School	35.07	-91.90
Hazen High School	34.78	-91.58	Beebe Junior High School	35.07	-91.90
Biggers-Reyno High School	36.34	-90.80	Bradford High School	35.43	-91.46
Maynard High School	36.42	-90.90	Mcrae High School	35.12	-91.82
Oak Ridge Central High School	36.37	-91.21	Pangburn High School	35.43	-91.84
Pocahontas Junior High School	36.28	-90.98	Searcy High School	35.25	-91.76
Pocahontas High School	36.28	-90.98	Southwest Middle School	35.25	-91.75
Forrest City High School	35.02	-90.79	Augusta High School	35.28	-91.37
			Cotton Plant High School	35.01	-91.25

Figure A.1: Evaluation of solutions obtained via *Consider All*. Grid labels: instance, α and ω.

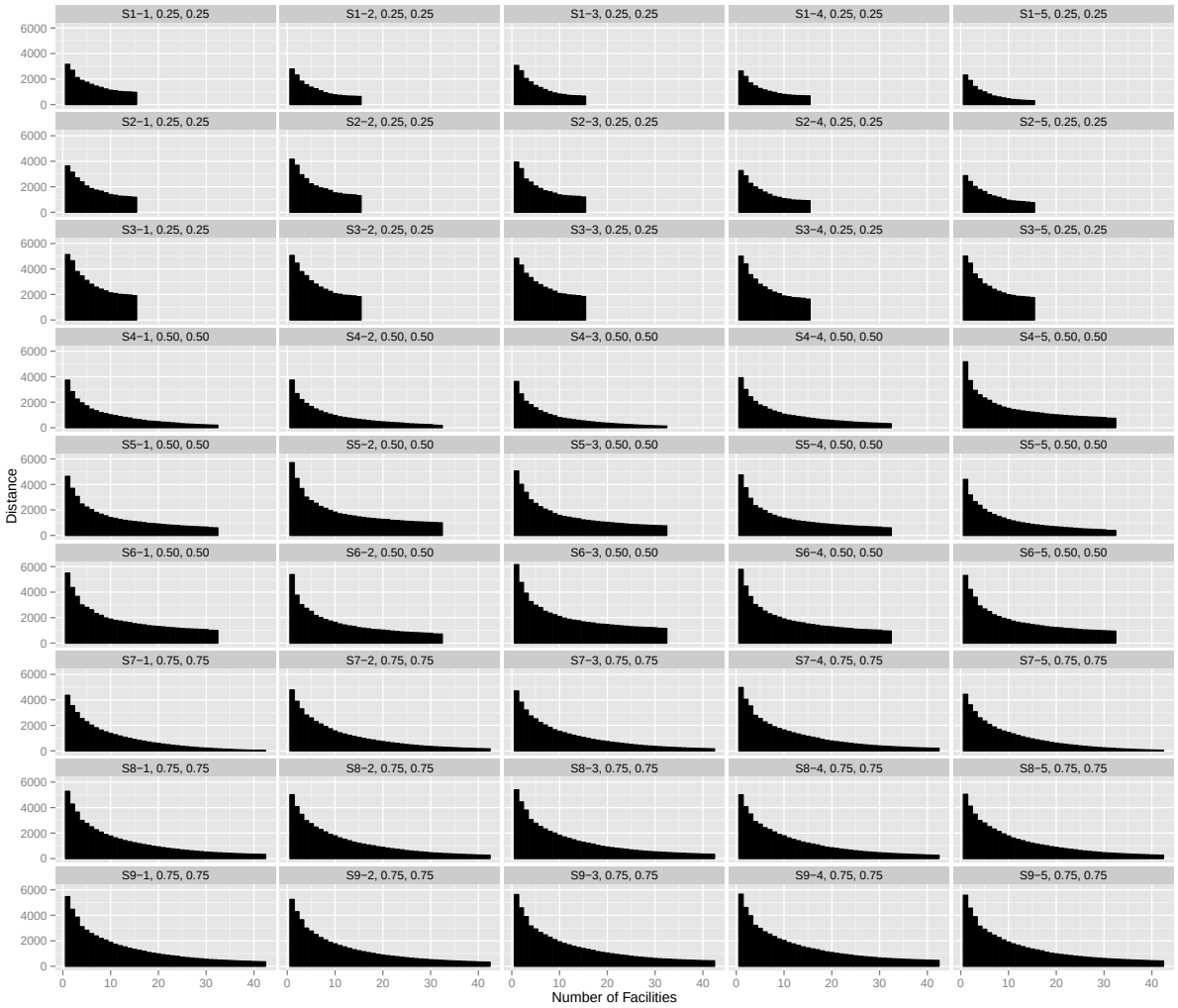


Figure A.2: Evaluation of solutions obtained via *Consider Only Verified*. Grid labels: instance, α and ω.

Figure A.3: Evaluation of solutions obtained via *Consider Minimax Regret*. Grid labels: instance, α and ω.

Figure A.4: Comparison of solutions obtained via the three strategies. Grid labels: instance, α and ω.

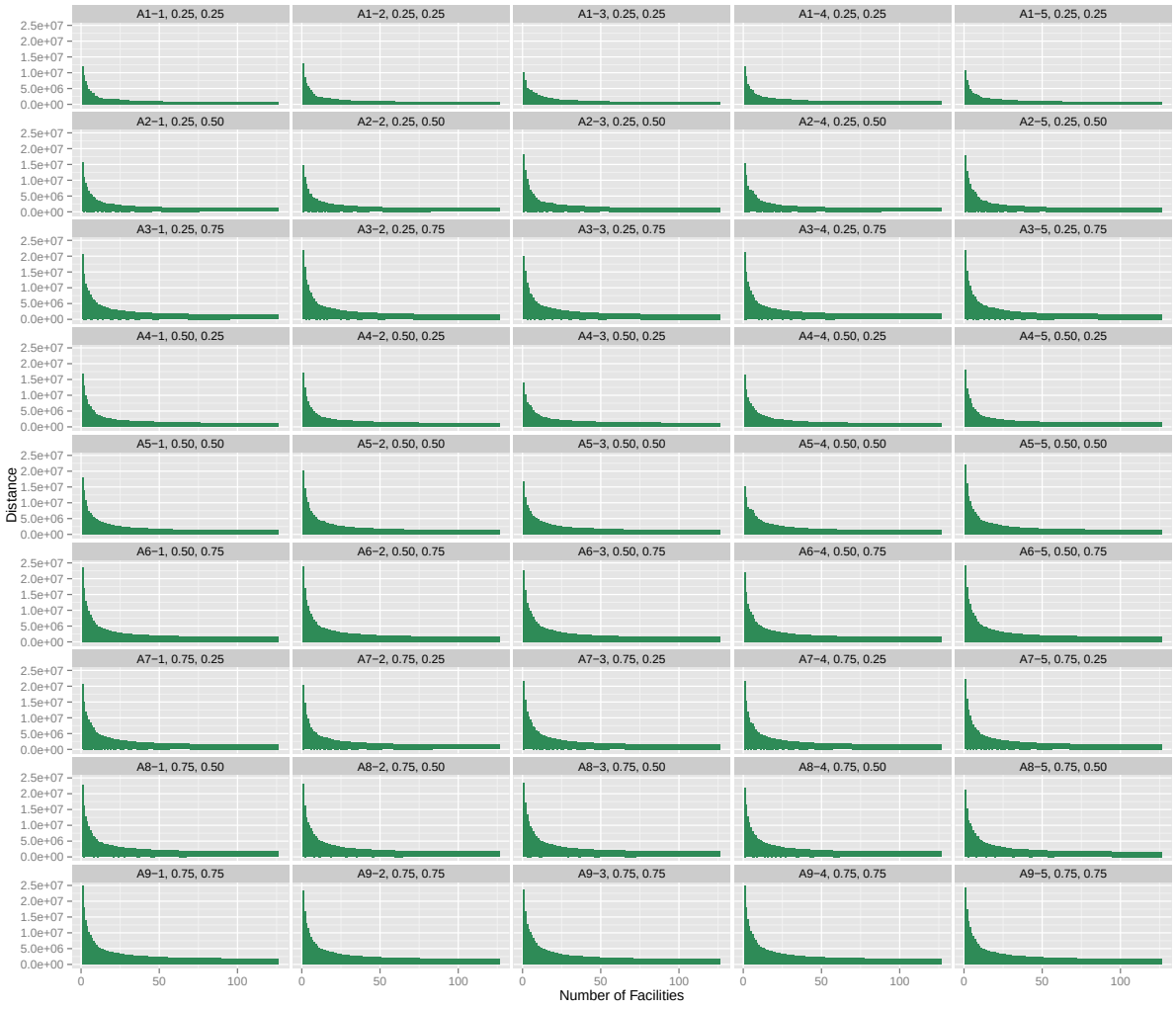


Figure A.5: Evaluation of solutions obtained via *Consider All*. Grid labels: instance, α and ω.

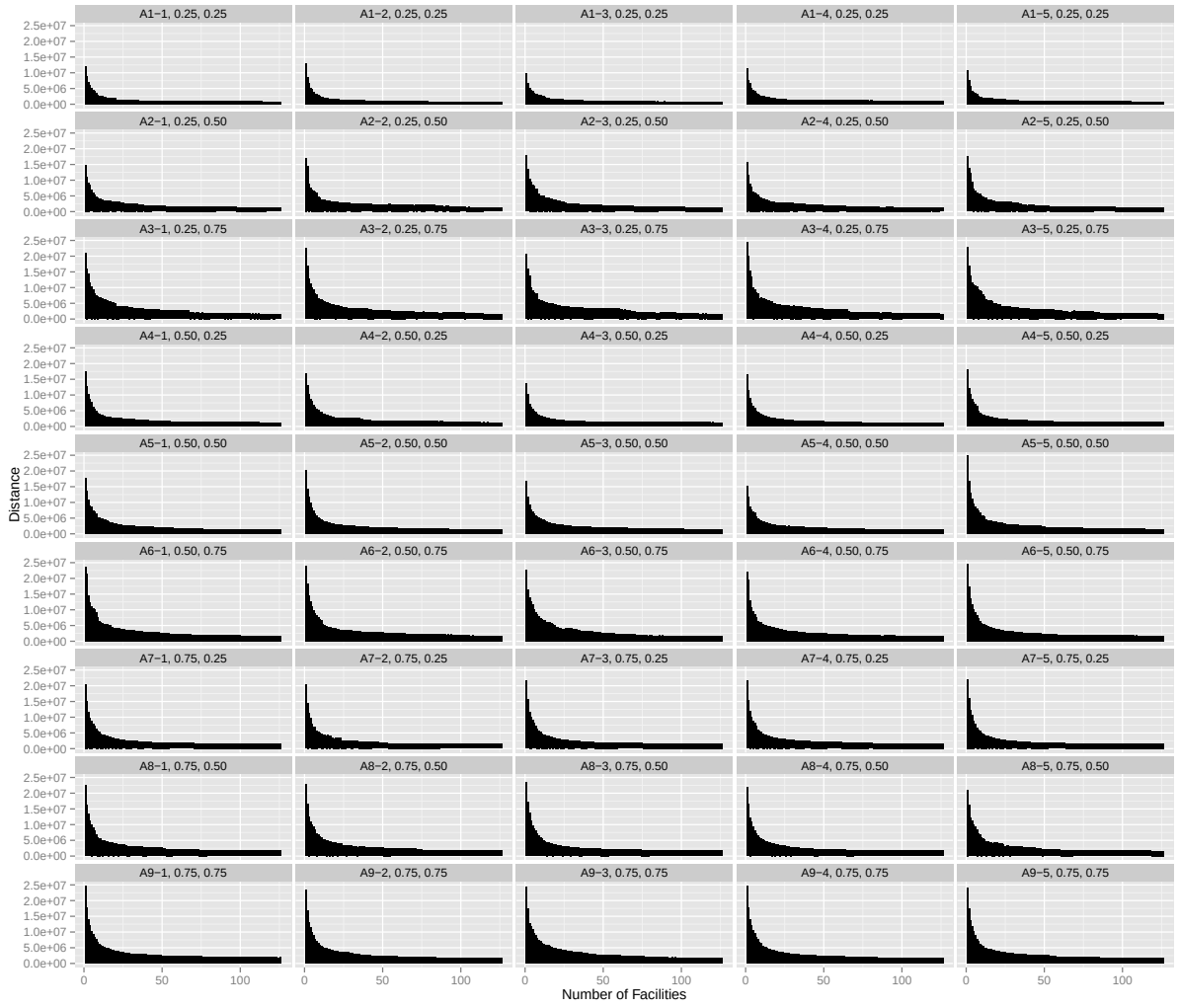


Figure A.6: Evaluation of solutions obtained via *Consider Only Verified*. Grid labels: instance, α and ω.

Figure A.7: Evaluation of solutions obtained via *Consider Minimax Regret*. Grid labels: instance, α and ω.

Figure A.8: Comparison of solutions obtained via the three strategies. Grid labels: instance, α and ω.