EMERGING APPLICATIONS OF OR/MS: EMERGENCY RESPONSE PLANNING AND PRODUCTION PLANNING IN SEMICONDUCTOR AND PRINTING INDUSTRY

A Thesis
Presented to
The Academic Faculty

by

Ali Ekici

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
December 2009

EMERGING APPLICATIONS OF OR/MS: EMERGENCY RESPONSE PLANNING AND PRODUCTION PLANNING IN SEMICONDUCTOR AND PRINTING INDUSTRY

Approved by:

Professor Pınar Keskinocak, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Özlem Ergun
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor David Goldsman
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Nathaniel Hupert
Weill Medical College
Cornell University

Professor Julie L. Swann
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: 11 August 2009

To My Parents,

Fatma and Musa Ekici.

ACKNOWLEDGEMENTS

My sincere thanks belong to my advisor, Dr. Pınar Keskinocak, for her continuous guidance, encouragement and financial support during this study. I would like to express my special thanks to Dr. Özlem Ergun and Dr. Julie L. Swann for their guidance and valuable insights in my studies. I thank Dr. David Goldsman and Dr. Nathaniel Hupert for their time and effort in serving on my Ph.D. committee.

I thank my officemate Örsan Özener for his patience, insightful comments, and valuable friendship. I also thank my friends, Serhan Duran, Murat Eröz, Burak Oğuz, Burak Karacık, Aykağan Ak, Emrah Uyar, Ozan Gözbaşı, and Fatma Kılınç Karzan for their friendship and support during my study. Finally, I would like to thank my parents, my sister and my fiancee for their unconditional love and support during my hard times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Operations Research and Management Science (OR/MS) techniques have been widely used by decision-makers to address a wide range of problems in public, private and nonprofit sectors. Although the usage of operations research techniques has started with military applications during World War II, currently these techniques are used in several contexts such as production planning, logistics management and health care management.

In this thesis, we study three emerging applications of OR/MS, namely, (i) disease spread modeling, intervention strategies, and food supply chain management during an influenza pandemic, (ii) the practical applications of production planning and scheduling in the commercial lithographic printing industry, and (iii) packing/placement problems in chip design in the semiconductor industry.

In the first part of the thesis, Chapter 2, we study an emergency response planning problem motivated by discussions with the American Red Cross, which has taken on a responsibility to feed people in case of an influenza pandemic. During an emergency such as an influenza pandemic or a bioterror attack, regular distribution channels of critical products and services including food and water may be disrupted, or some of the infected individuals may not be able to go to grocery stores. We analyze the geographical spread of the disease and develop solution approaches for designing the food distribution supply chain network in case of an influenza pandemic. In addition, we investigate the effect of voluntary quarantine on the disease spread and food distribution supply chain network. Finally, we analyze the effect of influenza pandemic on the workforce level.

In Chapter 3, we study a real life scheduling/packing problem motivated by the practices in the commercial lithographic printing industry which make up the largest segment of the printing industry. We analyze the problem structure and develop efficient algorithms to form cost effective production schedules. In addition, we propose a new integer programming formulation, strengthen it by adding cuts and propose several preprocessing steps to solve the problem optimally.

In Chapter 4, the last part of the thesis, motivated by the chip design problem in the semiconductor industry, we study a rectangle packing/placement problem. We discuss the hardness of the problem, explore the structural properties, and discuss a special case which is polynomially solvable. Then, we develop an integer programming formulation and propose efficient algorithms to find a "good" placement.

CHAPTER I

INTRODUCTION

The operations research techniques are first used by British researchers during World War II on military problems including submarine warfare, strategic bombing and radar deployment. After the war, these techniques were also applied to financial, industrial, administrative and government problems. Currently, these quantitative techniques help decision-makers in finding better solutions to problems in almost every sector including industry, health, education and agriculture. In this thesis, we apply simulation and integer programming techniques on three different industrial and government problems.

The first application is an emergency response planning problem motivated by discussions with the American Red Cross, which is charged with food distribution in case of an influenza pandemic. Based on the recent incidents of avian flu (H5N1), swine flu (H1N1), and influenza pandemic cases in history (1918, 1957 and 1968) experts believe that a future influenza pandemic is inevitable and likely imminent. During an emergency such as an influenza pandemic or a bioterror attack, regular distribution channels of critical products and services including food and water may be disrupted, or some of the infected individuals may not be able to go to grocery stores. Evidence suggests that an efficient and rapid response will be crucial for mitigating morbidity, mortality, and costs to society. Hence, preparing for a potential influenza pandemic has received high priority from governments at all levels (local, state, federal), non-governmental organizations (NGOs), and companies. We develop a disease spread model to estimate the spread pattern of the disease geographically and over time. In testing the disease spread model, we use detailed census-tract level data from Georgia (the model can be adapted easily to other states or an entire country). We employ the model to estimate the food need under various scenarios of which households will be fed (e.g., based on income level, or the number of infected people in an household). Then, we integrate the disease spread model with a facility location and resource allocation network model for food distribution. Since only the small instances of the embedded facility location problem can be handled by commercial optimization software, we develop heuristics to find near-optimal solutions for large instances. We run our integrated disease spread and facility location model for the state of Georgia and present the estimated number of infections and the number of meals needed in each census tract for a one year period along with a design of the supply chain network. Furthermore, we analyze the impact of two intervention strategies, namely, school closure and voluntary quarantine; our results indicate that voluntary quarantine may be a better alternative due to being more effective and less disruptive. Moreover, we investigate the impact of voluntary quarantine on the food requirement and the food distribution network, and show that its effect on the food distribution supply chain can be significant. Our results can help decision makers prepare including how to allocate limited resources or respond dynamically. Finally, we analyze the impact of influenza pandemic on the workforce level. Estimating the workforce loss during an influenza pandemic is crucial since it may affect the supply chains of goods and services and the response plans of the governments and NGOs.

The second application is motivated by the production planning and scheduling challenges faced in the printing industry, which is one of the largest manufacturing industries in the United States with products ranging from newspapers, books, business order forms, maps and packaging. We study a practical scheduling/packing problem, called *Job Splitting Problem*(JSP), which is prevalent in the commercial lithographic printing industry which makes up the largest segment of the printing industry. In JSP, there are n types of items to be produced on an m-slot machine. A particular assignment of the types to the slots is called a "run" configuration and requires a setup cost. Once a run begins, the production continues according to that configuration and the "length" of the run represents the quantity produced in each slot during that run. For each unit of production in excess of demand, there is a waste cost. The goal is to construct a production plan, i.e., a set of runs, such that the total setup and waste cost is minimized. We show that the problem is strongly NP-hard and discuss some special cases that can be solved in polynomial time. Moreover, we develop efficient algorithms to prepare cost efficient production plans that balance the setup and waste cost while satisfying the demand. In the current practice, forming a "good" production plan takes hours, even days. Extensive tests on real-world and randomly generated instances show that the proposed algorithms are easy to implement and find near-optimal solutions within seconds. In addition to developing efficient algorithms for finding near-optimal solutions, we also propose a strong linear integer formulation that can utilize special branching rules and strengthen it by adding cuts. Finally, we propose several preprocessing procedures, which help in solving the problem optimally in a reasonable amount of time.

In the third application, we study a rectangle packing/placement problem, namely *1.5-Dimensional Rectangle Packing Problem* (RPP), which has applications in the semiconductor industry. Motivated by the chip design problem, there are n rectangles, and their positions along the x-axis are specified. Given their fixed horizontal positions, we need to decide about the vertical positions of the rectangles to minimize the total height of the resulting placement. We show that the problem is strongly NP-hard, propose a method for finding a lower bound to the optimal solution, discuss a special case which is polynomially solvable, and propose two heuristics. In addition, we develop an integer programming formulation and propose ways to strengthen it. Extensive numerical tests on randomly generated instances show that the heuristics are both efficient and effective, finding near-optimal solutions within a few seconds.

CHAPTER II

MODELING INFLUENZA PANDEMIC, INTERVENTION STRATEGIES, FOOD DISTRIBUTION AND WORKFORCE LOSS ESTIMATION

2.1 Introduction

Many experts believe that an influenza pandemic has been imminent given the flu cases that happened in the last few years (avian flu-H5N1 and swine flu-H1N1) and the history of influenza pandemic [84]. Epidemiologists warn that the next influenza pandemic could infect 33% of the population and kill millions [43]. The World Health Organization (WHO) estimates that 2-7.4 million people might die worldwide [109]. Furthermore, the next influenza pandemic might cause a $71.3-165.5 billion economic impact on the United States economy [82]. The U.S. Department of Health & Human Services and the U.S. Department of Commerce estimate that in the next pandemic 20% of working adults may become ill, and there may be a 40% workforce loss during the pandemic peak because of illness, fear of infection and the need to care for infected family members or school-aged children.

Increased travel volumes favor the spread of infectious diseases [47], which makes surveillance and planning activities more important. As a preparedness plan, WHO strengthened its influenza surveillance and control system, and it is operating 110 laboratories in 83 countries all over the world [16]. Different from other influenza pandemic cases in history, the recent cases of bird flu gave a clear warning and have been eliminated by the monitoring and intervention efforts of WHO [109]. Recently, the outbreak of H1N1 caused the WHO to increase the alert level to phase 6 (on its six-point scale) (http://www.who.int, last accessed on June 12, 2009). This indicates

Table 1: Influenza pandemic cases in history

Past Cases	**Mortality**	**Populations Affected**
1918-19 (Spanish flu) (A/H1N1)	40-50 million (2.2-2.8%)	Persons $<$ 65 years
1957-58 (Asian flu) (A/H2N2)	2 million (0.069%)	Infants, elderly
1968-69 (Hong Kong flu) (A/H3N2)	1 million (0.028%)	Infants, elderly

that a pandemic is declared, although there may still be time to plan responses for later waves.

The impact of influenza pandemic cases in history shows the extent of how governments need to prepare response plans. There have been three cases of worldwide influenza pandemic in the 20th century, namely, Spanish flu (1918), Asian flu (1957) and Hong Kong flu (1968). There are different estimates about the number of deaths during these three pandemic cases. The estimates due to Smith [97] are presented in Table 1. The most severe one, Spanish flu, mainly affected people under 65, whereas the other two pandemics affected infants and elderly persons.

According to WHO, fifty countries have developed pandemic preparedness plans and most industrialized countries are stockpiling antiviral drugs [14]. Most preparation has focused on developing cell-culture vaccine manufacturing, stockpiling antivirals and vaccines, and school-closing plans, but designing response supply chains is also very important for meeting the various needs of the public during an influenza pandemic. Some of the infected individuals may not be able to go to grocery store to buy food, e.g., if they follow voluntary quarantine recommendations and stay home. Logistics of delivering these basic supplies to infected or quarantined households is an important operations research question [111].

In addition to governments, many non-governmental organizations (NGOs) have worked on response plans for a potential influenza pandemic [57, 84]. For example, the American Red Cross Metropolitan Atlanta Chapter (ARC-MAC) has worked on determining ways to provide food to people who are infected and need to stay home [2]. Ohio Department of Health and Ohio Food Industry Foundation [87] prepared an influenza pandemic preparedness plan for groceries. Based on the lessons learned from H1N1 experience [103], "Response plans must be adaptable and science-driven". Indeed, it is difficult for supply chains to respond effectively without considering the disease spread geographically and over time.

In this chapter, we consider the problem of providing food to people who are not able to obtain it due to illness in their household during the influenza pandemic. First, we develop a disease spread model to estimate the spread of the disease geographically and over time and then construct a food distribution network based on these estimates. We consider a *capacitated multi-period hierarchical facility location problem* (CMPH-FLP) for food distribution. To the best of our knowledge, this is the first study in the literature integrating a disease spread and a network design model for planning purposes. We develop algorithms to create a supply chain network that is dynamic and responsive to the changing need in the population, yet still computationally-efficient so they can be used operationally. We also generate insights about the network (how many facilities and how long to open) under different scenarios. Our results show not only how to combine epidemiological dynamics with operations research but also how network design can be performed for responsive, multi-hierarchical supply chain design. The research not only offers advances in scientific methods but is also immediately relevant for decision makers given the anticipated continuation of the H1N1 epidemic. Furthermore, we compare the impacts of *school closure* and *voluntary quarantine*, i.e., the individuals in the households with an infected individual stay home voluntarily with some compliance rate limiting their peer group and community interactions, on the disease spread and workforce level under different start times and durations. Finally, we study the effect of voluntary quarantine on the food distribution supply chain.

The remainder of the chapter is organized as follows. In Section 2.2, we present the literature on the disease spread models and facility location problems. The disease spread model we developed and the simulation results follow in Section 2.3. We study the voluntary quarantine as an intervention policy and present the results in Section 2.4. Moreover, we compare its impact to that of school closure. The alternatives for food need estimation follow in Section 2.5. In Section 2.6, we provide the facility location model, propose a greedy heuristic to solve it and present the solution approaches proposed for food distribution. In Section 2.7, we provide the results of computational experiments and report on the performance of the heuristics proposed for food distribution logistics. We also provide insights about the impact of voluntary quarantine on food distribution. In Section 2.8, we investigate the effect of voluntary quarantine and school closure on the workforce level. Finally, we conclude with future research directions regarding planning for an influenza pandemic.

2.2 Literature Review

There are two main streams of literature related to our research: (i) disease spread models, and (ii) facility location and distribution models.

The disease spread models are developed to predict the spread patterns and the effect of intervention strategies in populations with complex social and spatial structures and have been thoroughly researched for different infectious diseases such as influenza, smallpox and SARS (see Ferguson et al. [34] for a review of spread models for smallpox and Lipsitch et al. [74] for SARS).

There are four common ways to model the spread of an infectious disease in a population: (i) using differential equations [12, 39], (ii) simulation (agent-based) modeling [33, 42, 111], (iii) random graphs [14], and (iv) difference equations [47, 92].

In differential equation models, every individual is assumed to be in one of the disease stages, e.g., susceptible (S), infected (I), or recovered (R) in an S-I-R model, and the cumulative number of people in each stage is used to define the instant changes. In simulation models, the entire population is identified by individuals and social contact networks, e.g., households and peer groups, and the spread of the disease is predicted by discrete event simulation modeling. A comprehensive comparison of agent-based and differential equation models is provided by Rahmandad and Sterman [89]. In random graph models, random graphs are used to construct the contact network, and the disease spread is predicted accordingly. In the models that use difference equations, the entire time horizon is identified by a sequence of time intervals, and the disease spread is predicted recursively. The spread in each time interval is defined as a function of the spread in previous intervals. Another feature that distinguishes disease spread models is the mixing assumption. In homogeneous mixing every individual has the same chance to get infected, while in heterogenous mixing the chance of getting infected for an individual depends on the number of contacts s/he makes during the day, which varies from person to person. Thus, disease spread models with a heterogenous mixing assumption capture more aspects of a real life setting, but as expected they are more complex when compared to homogeneous models. The literature about the modeling of influenza pandemic and annual influenza is summarized in Table 19 in Appendix A.1.

There are two performance measures commonly used to evaluate the effectiveness of intervention policies, namely, *peak infectivity* and *infection attack rate* (IAR) [89]. Peak infectivity is the percentage of the infected (symptomatic or hospitalized) population at a given time, and IAR is the cumulative percentage of people who have been infected (can be symptomatic or asymptomatic) during the course of the disease. School or workplace closure and other social distancing measures such as travel restrictions and quarantining can reduce the peak infectivity but may not significantly affect IAR [32, 42, 109]. For example, social distancing measures only delayed the spread in the 1918 and 1957 pandemics [109, 110] but had little impact on IAR. Nevertheless, delaying the spread and the peak is desirable for planning purposes, since it provides more time for preparedness efforts, and a flattened spread (with a smaller peak) decreases the probability of interruption of services and is less likely to result in capacity bottlenecks in response activities.

To determine the location of the food distribution facilities and their opening/closing decisions, we need to know the geographical spread. In addition, interventions can affect some groups differently than others. Thus, we develop a simulation-based disease spread model with *heterogeneous* mixing to predict the spread pattern of the disease *geographically* and *over time*, and test the effectiveness of voluntary quarantine as an intervention policy. Different from other papers in the literature, we consider the case when interventions are active for a limited period of time. An influenza pandemic may last up to one year [32, 78, 111], and the papers in the literature assume that voluntary quarantine remains active during the entire course of the disease. We consider a limited time voluntary quarantine since prolonged quarantine times may have a negative impact on the public morale or may be difficult to sustain for the full outbreak (e.g., one year). Social distancing measures such as voluntary quarantine isolate people or limit public gatherings, which are indicators of normal life and help maintain public morale [24]. According to a survey [53] and empirical research [15] based on the SARS experience in Toronto, people may develop emotional problems during and after the quarantine. Thus, it is in the public officials' interest to obtain maximum benefit out of voluntary quarantine by keeping the disruption in people's normal lives at a minimum. When voluntary quarantine is encouraged (or promoted via educational materials) for a limited time, the start time and the duration are important decision variables, which have not been studied in the literature.

Given an estimate of the disease spread over time and geographically, we need to determine the location of food distribution facilities and how to open and close them over time. This is a capacitated multi-period hierarchical facility location problem (CMPH-FLP) with two levels of facilities between supply and demand nodes; even a a single period, single level facility location problem (FLP) is NP-hard in the general case.

Capacitated and uncapacitated multi-period FLPs have been studied in the literature (e.g., Roy and Erlenkotter [91], Shulman [95]). Erlenkotter [29] provides a comparison of solution approaches for the multi-period FLP. A popular solution approach for dynamic FLP is to generate the alternative solutions for the single period problem and look for the best combination of these alternative solutions by dynamic programming [6, 100].

Hormozi and Khumawala [62] propose an exact algorithm to solve an uncapacitated dynamic FLP integrating mixed-integer and dynamic programming methods. They reduce the size of the state space for the algorithm proposed by Sweeney and Tatham [100]. Canel et al. [13] study a capacitated multi-commodity multi-period hierarchical (two-echelon) FLP and develop a 2-stage solution method. First, they use a branch and bound algorithm to generate a set of candidate solutions for each period. Then, they use dynamic programming to find the optimal sequence of candidate solutions over the multi-period horizon. Their algorithm is effective when facility opening and closing costs are high. Hinojosa et al. [55] also consider a capacitated multi-commodity multi-period hierarchical FLP, and propose a Lagrangean relaxation scheme with a heuristic algorithm for finding feasible solutions.

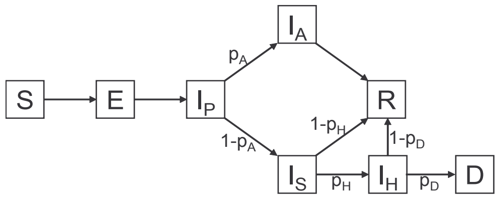
Our approach is to use a mixed integer linear program, which is different than previous attempts [13, 62]. Since real world examples, including ours, of CMPH-FLP are large (even single period instances can be hard to solve optimally), we develop a heuristic to solve CMPH-FLP efficiently and effectively. Moreover, we create an integrated solution approach that links the disease spread model and the facility location model and allows for dynamic disease spread updates as the real epidemic unfolds.

2.3 Disease Spread Model, Results, and Validation

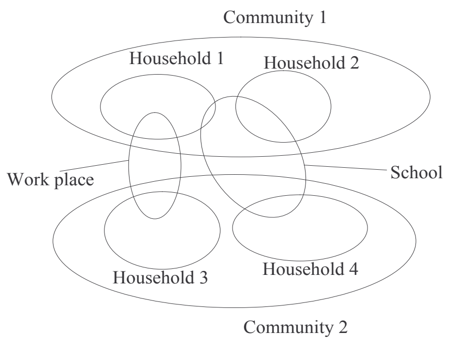
We construct an agent-based continuous time stochastic model for influenza transmission. In the base model, we do not apply any intervention strategy. We investigate the effect of voluntary quarantine (or social distancing) or school closure on the spread of the virus in Section 2.4. In addition to food distribution planning, this model may also be useful for other purposes such as estimating the region-based hospital capacity needs for local governments. The disease spread model is composed of two parts: (i) the progress of the disease within an infected individual, and (ii) the spread of the disease among the members of the population.

In our model, an infected individual goes through the stages of the disease according to the natural history for influenza pandemic in Wu et al. [111] (see Figure 1(a)). The progress of the disease depends on the age of the individual [108]. Hence, we divide the population into five age groups, namely, 0-5, 6-11, 12-18, 19-64, 65+. Each individual is assumed to be in one of the following stages at a given time: susceptible (S), exposed (E), presymptomatic (I_P), asymptomatic (I_A), symptomatic (I_S), hospitalized (I_H), recovered (R) or dead (D). p_A is the probability of not developing symptoms, p_H is the probability of hospitalization for a symptomatic individual, and p_D is the probability that a hospitalized individual dies. We summarize the values of the parameters for natural history of disease and the relevant references in Table 20 in Appendix A.2.

We model the spread of the disease among the members of the population via a contact network. Given the importance of age, density, and geography in predicting the disease spread [47], we construct a disease spread model that takes into account population heterogeneities. For example, children are considered to play a major role in the transmission of influenza [107] because they are assumed to be more susceptible due to lower immunity and to have more daily contacts through schools and play groups than adults have in workplaces.



(a) Natural history of disease for influenza



(b) Example of a contact network

Figure 1: Two main components of the disease spread model

First, the entire population is divided into communities that correspond to neighborhoods. The communities are linked to each other via peer groups, which account for the inter-community spread of the disease, and may be determined by age. In our model, we consider three levels of mixing: (i) community (day/night) (ii) peer group (day) and (iii) household (night) (see Figure 1(b)). All the individuals mix in the community during the day by visits to common areas such as grocery stores, churches, etc. The children in the first three age groups (0-5, 6-11, 12-18) mix with other children in kindergarten, elementary, and secondary schools. People in the age group 19-64 are considered as working adults, and they mix in workplaces with other adults. In the base model, we assume 50% of adults and 100% of children do not mix in their peer groups when symptomatic [111]. In the H1N1 case, Centers for Disease Control and Prevention (CDC) suggested that children with flu-like symptoms should stay home, which is also consistent with our assumption [90]. The elderly are assumed not to mix in peer groups. A susceptible individual may thus get an infection from the other individuals in his/her household, peer group, or in the community with different probabilities (contact rates) in each. We assume a constant import rate (1.5 infected individuals per day per 100,000 people) for each community, which represents the infected people coming from outside the contact network [111].

A comparison of the most relevant models in the influenza pandemic literature and our disease spread model is provided in Appendix A.3. To summarize, we develop a detailed SEIR (Susceptible-Exposed-Infected-Recovered) disease spread model with a spatial (geographical) component, age-based structure, heterogenous mixing, and night/day differentiation.

Our disease spread model is generic and can be applied to any geographical area. Given our collaborations with ARC-MAC, Georgia Department of Education, and Georgia Department of Human Resources, we take the state of Georgia as the test case and construct our model accordingly. There are 159 counties and 1615 census tracts in the state of Georgia, and the total population is 9,071,756. We consider each census tract as a single community and use census data [105] to form the households and peer groups and to identify the sizes of the age groups.

The details of the disease spread model are provided in Appendix A.4. An important parameter in the model is the basic reproductive number (R_0), which is the average number of secondary cases caused by an infectious individual [54] and determines the infectivity of the virus. For example, R_0 for the Spanish flu in 1918 is estimated to be around 1.7-2.0 [33] or 2-3 [83]. On the other hand, R_0 for the 1957 and 1968 pandemics are estimated to be around 1.5-1.7 [33] and 1.89 [92], respectively.

We ran the simulations for a range of R_0 values to account for low ($R_0 = 1.5$), medium ($R_0 = 1.8$) and high ($R_0 = 2.1$) infectivity. Figure 2 shows the spread of the disease over time among the population of Georgia for different R_0 values. Table 2 summarizes the simulation results as an average of 20 runs in the absence of intervention policies. Peak infectivity and IAR (infection attack rate) are defined above. *Peak day* is the time when the percentage of the infected population is at its maximum. CAR (*clinical attack rate*) is the cumulative percentage of the people who have been symptomatic, and *death ratio* is the percentage of the people who died because of influenza during the course of the disease.

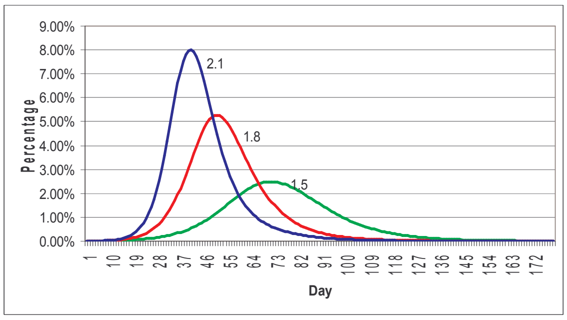


Figure 2: Percentage of symptomatic or hospitalized individuals under no intervention

Table 2: Results of the disease spread model with no intervention policy

R_0	**Peak Infectivity**	**Peak Day**	**CAR**	**IAR**	**Death Ratio**
1.5	2.48%	70	32.50%	49.65%	0.57%
1.8	5.27%	50	44.20%	67.49%	0.80%
2.1	8.01%	40	51.27%	78.27%	0.93%

In planning for food (or other resource) distribution, the spread of the disease over area and time is important since the time and the location of the food need determines the location and opening/closing time of the facilities. Based on the result of an unpaired t-test, there is not a significant difference in peak infectivity, peak day, and IAR for different number of initial infections (1 vs. 30) for $R_0 = 1.5$. However, for $R_0 = 1.8$ and 2.1, the disease peaks 1-2 days earlier if the number of initial infections is 30. Figure 3 shows the spread of the disease for two different disease initializations (seeds) over time and area for $R_0 = 1.8$. The shades are on a logarithmic scale, and darker shades represent higher number of infections in the relevant area. In the first case, the infection starts from Atkinson county, which is a rural area in the southeast part of Georgia. In the second case, the infection starts from Fulton county, which is in the Metropolitan Atlanta Area. As the maps in Figure 3 show, the spread in the Metropolitan Atlanta Area is not affected significantly by the location of the initial seed; these areas are always highly infected around day 50. However, the spread in the rural areas often depend on the location of the first infection.

As discussed before, our disease spread model parameters are in line with what have been used in the literature by several authors. To further validate our model, we calibrated the model parameters to obtain the age-specific clinical attack rates for the 1957 influenza pandemic reported by Chin et al. [17] (see Appendix A.5 for details). We obtained similar age-specific clinical attack rates for an R_0 value of 1.58, which is also consistent with the estimated R_0 value of 1.5-1.7 for 1957 pandemic [33]. Similar calibration procedures have been used by others [32, 78, 88].

2.4 Intervention Policy: Voluntary Quarantine vs. School Closure

In this section, we investigate the effect of a voluntary quarantine for several R_0 values. In addition, we compare voluntary quarantine and school closure in terms of their impacts on the disease spread. Since quarantined people interact with other individuals in their households, the risk of getting infection within the households is doubled for quarantined households, consistent with Ferguson et al. [33], Longini et al. [78] and Wu et al. [111]. We assume 50% compliance, that is, individuals in the quarantined households comply with the quarantine independently with probability 0.5. A household may be quarantined if an individual from that household develops symptoms or the individual is hospitalized. Once a household is quarantined, if no other individual in the quarantined household develops symptoms or is hospitalized within a week, the quarantine is released. Otherwise, the quarantine is extended for another week for that household. The quarantine is active for a limited period of time (2-12 weeks).

Both the timing and the length of the quarantine are important in order to obtain the maximum benefit. We are particularly interested in the impact of voluntary quarantine on peak infectivity, which relates to the maximum capacity for the resources

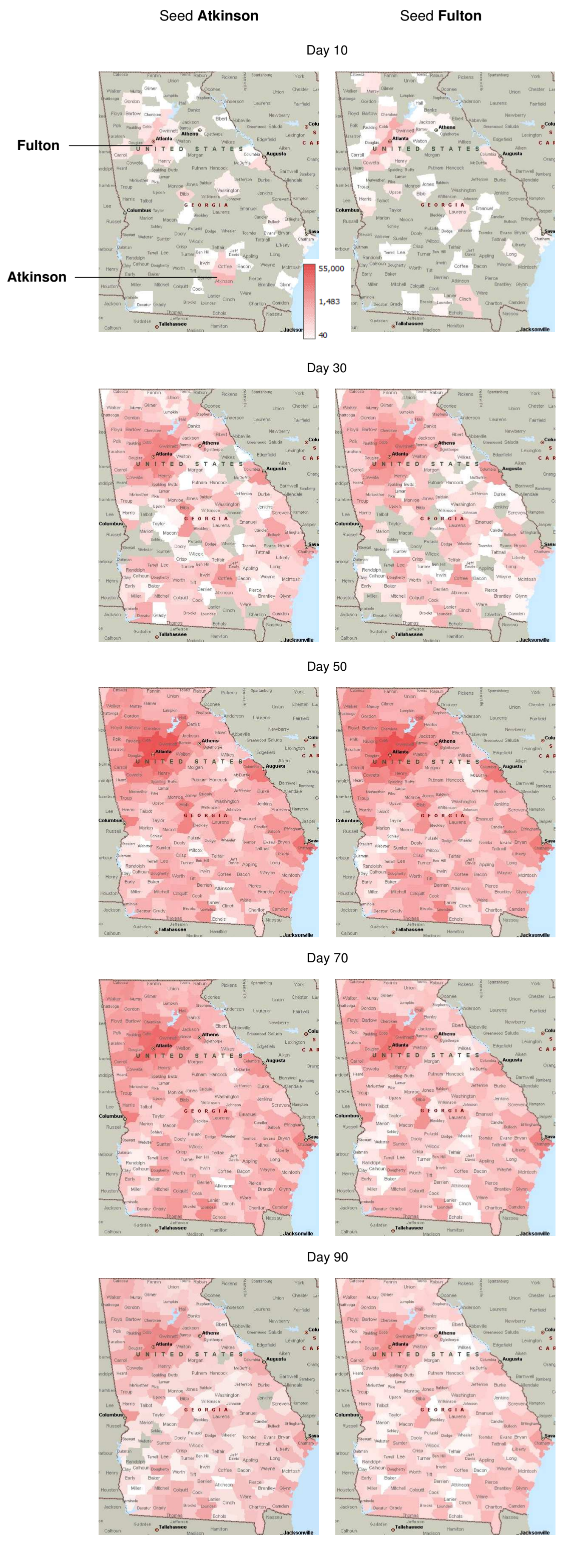
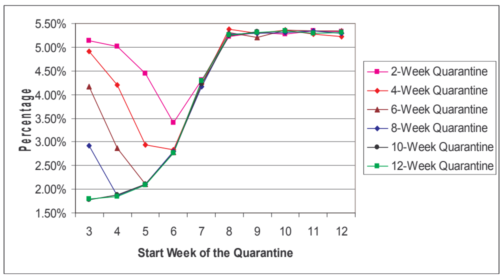


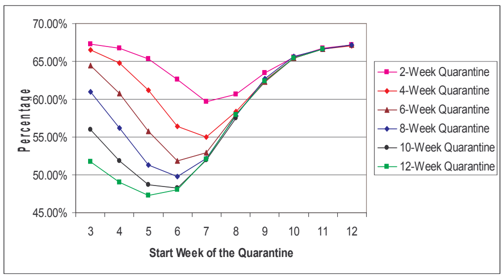
Figure 3: Comparison of the disease spread when the pandemic starts in a rural, less populated county (Atkinson) versus a central, densely populated county (Fulton). Darker shades indicate higher infection rates

that governments and NGOs may need to serve the needs of the public, and the IAR, which relates to the total amount of resources during the course of the disease [89]. Figure 4 shows the effect of the quarantine on the peak infectivity and IAR for $R_0 = 1.8$ as a function of the start time and length. From Figure 4(a), for a 2-week quarantine, the peak infectivity is lowest (3.40%) when the quarantine starts at the beginning of the sixth week. On the other hand, for a 12-week quarantine, the best start time in terms of peak infectivity is the beginning of the third week, and in this case the peak infectivity is 1.79%. Even a 12-week quarantine has no effect on the peak infectivity if the timing of the quarantine is not appropriate (e.g., week 7). From Figure 4(b), IAR is minimized at 59.71% if a 2-week quarantine is implemented at the beginning of week 7. For a 12-week quarantine, IAR is lowest (47.32%) if it is implemented at the beginning of week 5. *In general, as the length of the quarantine increases, the optimal start time (for minimizing the peak infectivity or IAR) decreases.* Note that the optimal start time of a quarantine is also related to the timing of the peak of the disease, which depends on R_0. As R_0 increases it is best to start a limited quarantine earlier to reduce peak infectivity.

During the course of the pandemic, estimating the disease spread parameters accurately, and thus, determining the exact duration and start time of the quarantine can be difficult. Therefore, voluntary quarantine can be announced a few weeks before the estimated optimal timing which can be compensated by extending the length of the quarantine. For example, for $R_0 = 1.8$, if a 6-week voluntary quarantine is started at the beginning of week 3 (instead of the beginning of week 5 to maximize the reduction in peak infectivity), similar peak infectivity (2.08%) can be achieved at the cost of a 3-week extension of the quarantine. However, delaying the start of the quarantine beyond the optimal timing may have severe consequences. For the previous example, if the quarantine is started 2 weeks late, the peak infectivity cannot be lower than 4.28%. Finally, as the length of the quarantine increases, starting the quarantine early is better than starting it late.



(a) Peak infectivity



(b) Infection attack rate

Figure 4: Effect of timing and length of quarantine on the peak infectivity and infection attack rate

Both the peak infectivity and IAR decrease as the length of the quarantine increases, but there is a diminishing rate of return. The peak infectivity (in Figure 4(a)) in an 8-week quarantine is almost equal to that of a 12-week quarantine.

For different R_0 values, Table 3 summarizes the results for an 8-week quarantine (with the objective of minimizing the peak infectivity), which is the break point where the diminishing rate of return is clearly observed for the peak infectivity. We observe that for moderate to high R_0 values, i.e, 1.8 and 2.1, an 8-week quarantine has about the same impact on peak infectivity compared to a quarantine that is in effect during the entire course of the disease. Imposing the quarantine during the entire course of the disease versus for only 8 weeks versus no quarantine results in IARs of 42%, 56%, and 67.5%, respectively, for $R_0 = 1.8$ and 58.5%, 63%, and 78%, respectively, for $R_0 = 2.1$. That is, for moderate to high R_0 values, an 8-week quarantine captures most of the benefits in reducing the peak infectivity and IAR. However, for $R_0 = 1.5$, there is a significant reduction in peak infectivity (from 0.80% to 0.30%) and IAR (from 40.5% to 17.4%) if the quarantine is imposed during the entire course of the disease instead of an 8-week time interval. Hence, for low R_0 values (where the epidemic curve is smoother with a smaller peak even under no intervention), a longer quarantine can be more beneficial.

Table 3: Summary of results under an 8-week voluntary quarantine with 50% compliance

R_0	**Quarantine Start Week**	**Peak Infectivity**	**Peak Day**	**CAR**	**IAR**	**Death Ratio**
1.5	7	0.80%	52	26.52%	40.46%	0.47%
1.8	4	1.86%	63	36.82%	56.14%	0.66%
2.1	3	3.97%	49	41.26%	62.87%	0.75%

In a voluntary quarantine, the reduction in the peak infectivity is high when compared to the reduction in other performance measures. For $R_0 = 1.8$, an 8-week quarantine with optimal timing reduces the peak infectivity by 64.71%, CAR by 16.70%, IAR by 16.82% and the death ratio by 17.22%. Keeping infected individuals at home decreases their interactions with the others outside, but in the long run susceptible individuals may have an interaction with other infected individuals in the community after the quarantine is released. The reductions in the peak infectivity, CAR, IAR and death ratio for different R_0 values for an 8-week quarantine with the optimal start time are provided in Appendix A.6. Depending on the start time and the duration of the quarantine, we sometimes observe two peaks in prevalence. This occurs because the disease spread slows down during the quarantine but speeds up again after the quarantine is released. However, even in these cases, the highest of the two peaks as well as the IAR are lower under quarantine versus no quarantine. In addition, delaying the peak offers tremendous opportunities for better preparedness and response (including the potential development of an appropriate flu vaccine).

Ferguson et al. [33], Longini et al. [78] and Wu et al. [111] also study voluntary quarantine as an intervention policy. The comparison of the results is provided in Appendix A.6. Different from our limited time assumption, all the other papers assume that the quarantine is active during the entire course of the disease, which may not be practical. Our results indicate that as R_0 increases, an 8-week quarantine (if started at the right time) is almost as effective as a quarantine that is imposed during the entire course of the disease.

We also ran experiments with other compliance rates (25%, 75%, 100%). We observe that as the compliance rate increases, peak infectivity and IAR decrease. However, for high compliance rates, e.g., 75% and 100%, we may observe two peaks depending on the R_0 value and the duration of the quarantine. After a quarantine with high compliance rate is released, if the virus is still active and no pharmaceutical intervention is applied, the number of infections increases again resulting in a second peak (the maximum of the two peaks is smaller than the peak in the no intervention case). Therefore, higher compliance rates provide a good opportunity to decrease peak infectivity and IAR significantly, and more importantly, they offer more time for preparedness by delaying the (second) peak.

In addition to voluntary quarantine, school closing is another intervention policy that has received much attention [14, 32, 33, 42]. Georgia Department of Education prepared a report to assist school administrators in preparing influenza pandemic plans [21] explaining local and state responsibilities, how to organize school activities, and how to communicate with staff, parents, and community. Jones [66] explains when and for how long to close the schools in case of an influenza pandemic. 4-week closures are recommended in case of a pandemic similar to the one in 1957 and up to 12-week closures are recommended for a pandemic such as the one in 1918. Recently, during the H1N1 epidemic, CDC considered school closures and suggested that the schools with a confirmed case should be closed for 2 weeks [81], and 726 schools were closed at the peak [90].

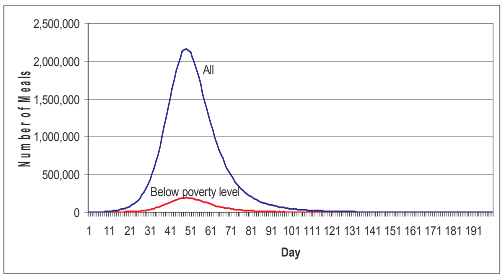
Based on the suggestions in Jones [66], we consider a 6-week school closure for $R_0 = 1.5$ and a 12-week school closure for $R_0 = 1.8$ and 2.1. Our simulation results indicate that school closure may not be as effective as voluntary quarantine. For example, for $R_0 = 1.8$, even a 2-week quarantine with optimal timing is as good as a 12-week school closure with optimal timing in terms of reducing both peak infectivity and IAR (see Appendix A.7 for the effect of the start time of school closure on peak infectivity and IAR). The main reason is that while quarantine targets all infected people, school closures do not affect adults. Thus, considering the additional disruptions in services that could be caused by school closures [103] (if some parents have to leave the workforce to stay home to take care of their children), voluntary quarantine may be a better alternative as a social distancing measure.

2.5 Estimating the Food Need

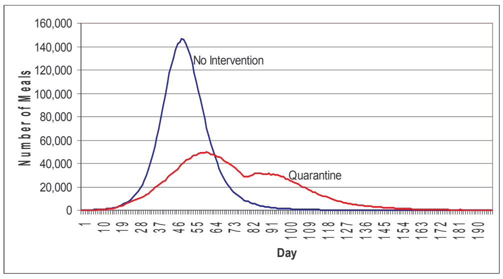
In this section, we propose several alternatives to calculate the food need in case of an influenza pandemic. In the state of Georgia, the ARC-MAC has taken on a responsibility to feed people in case of an influenza pandemic, and our research has been motivated by their planning. While the ARC-MAC focuses primarily on Metropolitan Atlanta Area (see Figure 23 in Appendix A.8 for a Metropolitan Atlanta Area map showing population densities), it is also leading the discussions with other organizations on planning for food distribution at the state level.

In the remainder of the chapter, we consider $R_0 = 1.8$ and design a food distribution supply chain network for an influenza pandemic. We estimate the food need using the disease spread model assuming that an individual needs 3 meals a day. There are several alternatives for calculating the food need depending on who to feed: (i) serve the households with at least one infected (symptomatic or hospitalized) individual (Figure 5(a)), (ii) serve the households that are below poverty level with at least one infected individual (Figure 5(a)), (iii) serve the households with all adults infected (Figure 5(b), under no intervention and 8-week quarantine starting at the fourth week), (iv) serve the households that are below poverty level with all adults infected, and (v) serve the quarantined households.

In Figure 5(a), the number of meals needed is high when compared to other alternatives since all the households with an infected individual are served in this case. In Figure 5(b), we observe two peaks for an 8-week quarantine. After the



(a) The households with an infected individual are served



(b) The households with all adults infected are served under no intervention or an 8-week quarantine policy

Figure 5: Number of meals needed daily for different alternatives

quarantine is released, there occurs another (lower) peak due to the increased number of interactions in the community.

2.6 Facility Location Model and Solution Approaches for Food Distribution

In this section, we explain CMPH-FLP and propose a mixed integer linear formulation. Since CMPH-FLP is NP-hard, and even medium size instances of CMPH-FLP cannot be handled using commercial solvers such as CPLEX 9.0, we propose an *Add-Drop Heuristic* (ADH), which is a modification of add [71] and drop [20] heuristics proposed for FLPs. An example of an add-drop heuristic can be found in Narula and Ogbu [85] in the context of determining the location of health care centers and hospitals. We develop new aspects to capture the hierarchical structure of the problem and the changes in the demand in a multi-period setting. Furthermore, we propose two solution approaches for designing and managing the food distribution network during the influenza pandemic where the approaches implement the solution of the CMPH-FLP heuristics in a static and dynamic fashion.

We model three tiers in the distribution network: supply points, major facilities (where the food is processed and/or packed for easy pick-up by individuals), and points-of-delivery (PODs) (e.g., schools, churches, community centers, businesses, etc.). Individuals/households who are in need will get their food from the PODs (which we assume can be run with minimum personal contact, e.g., individuals could drive through and someone puts the food into the trunk). Other ways of distribution can be determined for those without transportation. In our food distribution network, each census tract is considered as a demand node, and the amount of demand is determined by the number of individuals/households in need. The major facilities and PODs can be opened or closed over time based on the estimates of demand across area and time. In our formulation, we consider demand updates and closing/opening decisions on a weekly basis.

The questions that need to be answered are: (i) Where to locate major facilities and PODs? (ii) How to open/close these facilities over time, e.g., on a weekly basis as the (anticipated) food need changes? (iii) How to allocate the food among the major facilities and the PODs with the goal of minimizing the total cost of serving the target population?

2.6.1 Mixed Integer Linear Model

We develop a mixed integer linear formulation for CMPH-FLP (Table 4 summarizes the notation) assuming that the demand, i.e., the number of meals needed for each week/period in the time horizon for each census tract, is known; we obtain demand estimates by running the disease spread model. To the best of our knowledge, this type of formulation has not been proposed for CMPH-FLP.

The objective in our model is to minimize the total cost (including the travel time of individuals) while satisfying the demand. The definitions of the variables used in

Table 4: Notation used in the formulation

T	:	number of weeks (time horizon)
N_1, N_2, N_3, N_4	:	sets of supply points, major facilities, PODs, and demand nodes
S_i	:	amount of meals that can be supplied by supply point i for $i \in N_1$
F_j	:	fixed cost incurred if facility j is open during a week for $j \in N_2 \cup N_3$
f_j, g_j	:	cost of opening and closing facility j for $j \in N_2 \cup N_3$
c_o^1, c_o^2	:	unit material (meal) handling cost at a major facility and POD, respectively
C_j	:	capacity of the facility j for $j \in N_2 \cup N_3$
D_{kt}	:	demand of demand node k in week t for $k \in N_4$, $t \in T$
d_{ij}	:	distance (in miles) between node i and node j for $i \in N_k$, $j \in N_{k+1}$, $k \in \{1,2,3\}$
c_u^1	:	unit transportation cost from a supply point to a major facility per mile
c_u^2	:	unit transportation cost from a major facility to a POD per mile
$c_{individual}$	:	unit transportation cost from a POD to a demand node per mile

the formulation are as follows:

$$x_{ijt} = \text{amount of food sent from node } i \text{ to node } j \text{ in week } t \qquad i \in N_k, j \in N_{k+1}, k \in \{1,2,3\}, t \in \{1,\dots,T\},$$

$$y_{jt} = \begin{cases} 1, & \text{if facility } j \text{ is open during week } t \\ 0, & \text{otherwise} \end{cases} \qquad j \in N_2 \cup N_3, t \in \{1,\dots,T\},$$

$$w_{jt} = \begin{cases} 1, & \text{if facility } j \text{ is opened at the beginning of week } t \\ 0, & \text{otherwise} \end{cases} \qquad j \in N_2 \cup N_3, t \in \{1,\dots,T\},$$

$$z_{jt} = \begin{cases} 1, & \text{if facility } j \text{ is closed at the end of week } t \\ 0, & \text{otherwise} \end{cases} \qquad j \in N_2 \cup N_3, t \in \{1,\dots,T\}.$$

Using these variables, the objective function can be written as:

$$\begin{aligned} \mathcal{OF}(x,y,w,z) = &\sum_{t=1}^{T} \sum_{i \in N_1} \sum_{j \in N_2} (d_{ij} c_u^1 x_{ijt} + c_o^1 x_{ijt}) + \sum_{t=1}^{T} \sum_{i \in N_2} \sum_{j \in N_3} (d_{ij} c_u^2 x_{ijt} + c_o^2 x_{ijt}) \\ &+ \sum_{t=1}^{T} \sum_{j \in N_3} \sum_{k \in N_4} d_{jk} c_{individual} x_{jkt} + \sum_{t=1}^{T} \sum_{j \in N_2} (F_j y_{jt} + f_j w_{jt} + g_j z_{jt}) \end{aligned}$$

The mathematical formulation of CMPH-FLP is as follows:

$$
\begin{array}{llll}
\text{Minimize} & \mathcal{OF}(x,y,w,z) & & (1) \\
\text{subject to} & \sum_{j \in N_2} x_{ijt} \leq S_i & i \in N_1, t \in \{1,\ldots,T\}, & (2) \\
& \sum_{j \in N_3} x_{jkt} \geq D_{kt} & k \in N_4, t \in \{1,\ldots,T\}, & (3) \\
& \sum_{i \in N_k} x_{ijt} \leq C_j y_{jt} & j \in N_{k+1}, k \in \{1,2\}, t \in \{1,\ldots,T\}, & (4) \\
& \sum_{i \in N_k} x_{ijt} = \sum_{l \in N_{k+2}} x_{jlt} & j \in N_{k+1}, k \in \{1,2\}, t \in \{1,\ldots,T\}, & (5) \\
& w_{jt} \geq y_{jt} - y_{jt-1} & j \in N_2 \cup N_3, t \in \{1,\ldots,T\}, & (6) \\
& z_{jt} \geq y_{jt} - y_{jt+1} & j \in N_2 \cup N_3, t \in \{1,\ldots,T\}, & (7) \\
& y_{j0} = 0 & j \in N_2 \cup N_3, & (8) \\
& y_{jT+1} = 0 & j \in N_2 \cup N_3, & (9) \\
& y_{jt} \in \{0,1\} & j \in N_2 \cup N_3, t \in \{1,\ldots,T\}, & (10) \\
& w_{jt} \in \{0,1\} & j \in N_2 \cup N_3, t \in \{1,\ldots,T\}, & (11) \\
& z_{jt} \in \{0,1\} & j \in N_2 \cup N_3, t \in \{1,\ldots,T\}, & (12) \\
& x_{ijt} \geq 0 & i \in N_k, j \in N_{k+1}, k \in \{1,2,3\}, t \in \{1,\ldots,T\}. & (13)
\end{array}
$$

In the above model, (1) is the objective function, which is the summation of total transportation cost, handling cost, facility operating cost and facility opening/closing cost. Constraints (2) and (3) are the supply constraints and demand constraints, respectively. (4) represents the capacity constraints for each facility (either a major facility or a POD). Constraints (5) are flow balance constraints. Constraints (6) and (7) restrict service to open facilities. Constraints (8) and (9) set the initial and final values. Finally, (10)-(13) are the integrality and sign restrictions.

Commercial optimization software such as CPLEX 9.0 can handle only small instances of CMPH-FLP. To find (near-)optimal solutions for large instances, we develop heuristic approaches.

2.6.2 Heuristics for CMPH-FLP

In this section, we explain ADH and a variant of it, namely, *Hybrid Heuristic* (HH). The pseudocode for ADH is provided in Appendix A.9. The idea behind ADH is as follows: In each period, to determine which PODs and major facilities to open, we solve two single period FLPs. That is, for period t, first we solve a single period version of CMPH-FLP assuming that the demand of node k is a weighted average of future demands (with weights decreasing over time) as follows:

$$\overline{D}_{k,t+1} = \frac{D_{kT}}{2^{T-t}} + \sum_{j=t+1}^{T} \frac{D_{kj}}{2^{j-t}} \tag{14}$$

The solution to this problem helps us predict the major facilities and PODs that will be open in the future. Then, considering the estimated future decisions and the decisions in the previous period, we solve a single period FLP with the demand of demand node k as D_{kt} to determine which major facilities and PODs to open in period t.

We give the general idea behind the subroutine for solving these two single period problems. First, starting with one of the demand nodes randomly, we assign each demand node to the nearest POD as long as there is enough unused capacity in the POD. If the POD is full, the demand point is assigned to the nearest POD that still has unused capacity. Then, for each POD, we calculate the savings achieved by closing it and assigning its demand to other nearby PODs. Then, we close the POD with the highest positive savings. Note that while calculating the savings achieved by closing a certain POD or major facility, the PODs and major facilities that were open in the previous period (period $t-1$) have an advantage since the opening and closing costs for these facilities are already incurred. We continue these steps until there are no other PODs with positive savings. Next, assuming that the remaining open PODs represent the demand nodes, we apply similar steps to determine which major facilities to open.

To summarize, in ADH, we solve a single period version of CMPH-FLP for the (weighted) average of the future demand estimates, and then, for the current period's demand using the subroutine explained above. In an alternative heuristic, namely, *Single Period Heuristic* (SPH), we solve both of these two single period problems optimally. SPH has larger run times but may find a solution with a lower cost. As a third alternative, we combine ADH and SPH such that in each period, we solve the first problem with the (weighted) average of future demand using the subroutine explained above and solve the second problem with the current period's demand optimally. We call this the *Hybrid Heuristic* (HH), which reduces the solution time required compared to SPH. In addition to ADH, SPH and HH, we consider a greedy algorithm, called *Myopic Heuristic* (MH), which involves solving the single period problem (using the subroutine in ADH) in each period without considering the future and past decisions. We test ADH, SPH, HH and MH for a set of instances and compare the solution times and the optimality gap of the solutions found by them in Section 2.7.

2.6.3 Solution Approaches for Constructing the Food Distribution Network

In this section, we discuss two solution approaches to the food distribution logistics problem during an influenza pandemic. The first one is called the *Deterministic Approach* (DET-A) where the food need is estimated using the disease spread model and input to the CMPH-FLP, which is solved by the heuristics proposed in Section 2.6.2. The demand estimate may be the average of multiple simulation runs. In this approach, the network is not updated over time with new disease spread estimates. The advantage of this approach is its simplicity of implementation (i.e., the decisions about when and where to open/close facilities are made at the beginning).

However, the decisions can be improved with more information about the spread of the disease over time, which motivates the second approach, called the *Dynamic Update Approach* (DYN-A). At the beginning of each week, we update our estimate on the amount of food need by using the information about the up-to-date spread of the disease. In this approach, we implement only the decisions for the current period and then rerun the simulation in the next week for the remaining time horizon by providing the status of the real-world spread as an input to the simulation. Decisions in the next periods are then again determined by solving CMPH-FLP for the remainder of the time horizon. This rolling horizon approach decreases the deviation of the estimates from the real-world situation. To the best of our knowledge, we are the first to propose and implement such a dynamic update algorithm in the context of determining the location of distribution facilities during a large-scale disease spread.

2.7 Computational Results

In this section, we provide the results of the computational experiments. First, we test the performance of the proposed heuristics to solve CMPH-FLP. Then, we compare the performances of DET-A and DYN-A. Finally, we present the analysis about the impact of voluntary quarantine on food distribution.

In the computational experiments, we consider the case of serving households with all adults infected, but the approach is valid for alternative ways of calculating the food need. We assume that we serve food to people when more than 0.5% of the population is infected at a given time, and this corresponds to an 8-week time interval (between weeks 5 and 12) for $R_0 = 1.8$. Although the exact percentage is hard to estimate, the assumption is reasonable because the NGOs and/or governments will not construct a large food distribution network if the number of infections is under some threshold value. Furthermore, in real life, the event of influenza pandemic may be recognized some time after the occurrence of the first infection. For example, Germann et al. [42] consider initiating the intervention strategies 7 (or 14) days after the pandemic alert, and they assume that a pandemic alert is triggered when the total number symptomatic individuals reaches 10,000.

2.7.1 Performance of the Heuristics

To evaluate the performance of the heuristics, we consider Gwinnett county, Fulton county, and the Metropolitan Atlanta Area as the test cases. There are 71, 167, and 603 census tracts in Gwinnett county, Fulton county, and the Metropolitan Atlanta Area, respectively. The number of periods is 8 (weeks) in the test instances, corresponding to the length of the time interval during which more than 0.5% of the population is infected at a given time.

Based on the estimations provided by ARC-MAC, the capacity of a typical POD is assumed to be around 10,000 meals per week. The total capacities of the major facilities and supply points are equal to the total capacity of the PODs. The supply points, major facilities and PODs are randomly assigned to the relevant areas. The opening/closing/fixed operating costs of the PODs and the major facilities are proportional to the square root of the capacity of the relevant facility since facility related costs are usually represented by a concave function of capacity due to economies of scale [31]. The opening, operating and closing costs of a major facility is 10 times that of a POD of the same size since most of the food processing/packing operations will be performed in major facilities. The opening cost is assumed to be two times the closing cost and four times the fixed operating cost. The shipments from supply points to major facilities and from major facilities to PODs will be in large amounts and done by trucks. However, for the shipments from PODs to households, either an individual from the household will drive to a POD or a small truck will distribute to households. Based on this observation, the unit shipment costs from supply points to major facilities (c_u^1) and from major facilities to PODs (c_u^2) are assumed to be equal to each other and 50% of the unit shipment cost from PODs to demand nodes

($c_{individual}$). Finally, we assume three different settings for the shipment costs, namely, low, medium and high. In the "low" setting, the facility related costs dominate the objective function, in the "medium" setting the shipment costs and facility related costs are comparable, and in the "high" setting, the shipment costs dominate.

We compared the average performance of the heuristics for each of the three shipment cost settings (low, medium, high). We used CPLEX 9.0 as the optimization engine, and all computational experiments are carried out on a system with a 2.4 GHz Xeon processor and 2 GB RAM. For finding the optimal solution, we set a time limit of 8 hours for the small instances (Gwinnett and Fulton county) and 12 hours for the Metropolitan Atlanta Area instances. We set a time limit of 0.5 hour and 1 hour for each single period problem in SPH and HH, respectively, since two optimization problems are solved in SPH in each period whereas only one optimization problem is solved in HH in each period. The detailed results (CPU times and optimality gaps with respect to the best lower bound) are presented in Appendix A.10. For each parameter setting, we generated 10 different instances, and the results presented are the average of these 10 instances. Since the optimality gaps are calculated with respect to the best lower bound found by CPLEX 9.0 within time limits, they are conservative estimates.

From the computational experiments, we see that CMPH-FLP becomes easier as the shipment costs dominate the objective function, since almost all of the facilities are open during the entire time horizon. One simply assigns each demand point to the closest facility. We conclude that it is the number of major facilities that makes CMPH-FLP harder (see third, fourth and fifth set of instances in Table 26, where we increase the number of major facilities, PODs and supply points compared to the second set of instances). This also shows that if the location of the major facilities can be determined or fixed independently, the corresponding CMPH-FLP can be solved more easily, and dynamically with the disease spread.

As the instances become larger, we observe that the best integer solution found at the end of the time limit is significantly worse than the solutions found by the heuristics. Even if we increase the time limit, CPLEX 9.0 cannot handle large instances due to a memory problem.

The solutions found by SPH and HH are very close to each other, and the average optimality gap is around 3%. However, the solution time of HH is around 30% that of SPH. The solution times required for ADH and MH are negligible compared to SPH and HH, but the average optimality gap is around 4.5% and 5.6%, respectively. Hence, we propose using HH and ADH to solve CMPH-FLP. Furthermore, for very large instances such as the entire state of Georgia, ADH is the best alternative since in this case even the single period problem is difficult to handle using commercial optimization software.

2.7.2 Deterministic Approach vs. Dynamic Update Approach

In this section, we compare the performances of DET-A and DYN-A as well as a benchmark case, "Perfect Solution", which is the solution obtained assuming that we know the real-world spread ahead of time, which is impossible to know but provides a comparison base for our solution approaches. Since large instances cannot be solved optimally using CPLEX 9.0, we consider Gwinnett county as the test case with 5 major facilities, 36 PODs, and 10 supply points. We consider using both of the heuristics (ADH and HH) as well as solving CMPH-FLP optimally to compare DET-A and DYN-A.

The same experimental setting explained in Section 2.7.1 is used in these experiments. The benchmark case is generated by a single simulation of our disease spread model. The demand estimates in DYN-A and DET-A are calculated by taking the average of 5 simulation runs. According to the result of the experiments, if we use HH and ADH, the solutions obtained by DYN-A are approximately 0.33% and 0.54%

better, respectively, when compared to DET-A in terms of total cost. In addition, the total cost of the solutions obtained by DYN-A are within 4% of the "Perfect Solution". Even if we solve CMPH-FLP optimally at each step of DYN-A and DET-A, the benefit of using DYN-A is only 0.37%. This indicates that the performance of DYN-A over DET-A is not affected by the algorithm used to solve CMPH-FLP.

The performance of DET-A is close to that of DYN-A since we assumed that we know the parameters of the virus at the beginning of the horizon. If the parameters can be estimated with reasonable accuracy, because of its simplicity to implement, DET-A is a good alternative for designing the food distribution network. In the case where the parameters cannot be estimated accurately, using DYN-A is a better choice.

2.7.3 Effect of Quarantine on Food Distribution Supply Chain

In this section, we investigate the effect of a voluntary quarantine on food demand and food distribution supply chain both in terms of total serving cost and the quality of service, which is defined as the percentage of demand served within 10 miles. Quality of service is important not only because of convenience to the public and potential shortages in gas supply, but also because if the infected individuals drive a long distance and make an additional stop, the infection could be introduced to other areas.

A quarantine with optimal timing and length decreases the likelihood of capacity bottlenecks and supply chain disruptions significantly. In the case of an 8-week quarantine, more than 0.5% of the population is infected between weeks 6 and 18, and the reduction in the total demand is 26.70% when compared to the no intervention case. Note that the reduction in IAR was 16.82% (see Section 2.4). On the other hand, the reduction in the average demand (over time) is even higher (55%) since the demand

(and number of infections) is more spread over time (see Figure 6). A similar observation is made by Rahmandad and Sterman [89] after analyzing the effect of social distancing on the disease spread, and they also mention the effect of peak infectivity on health services infrastructure.

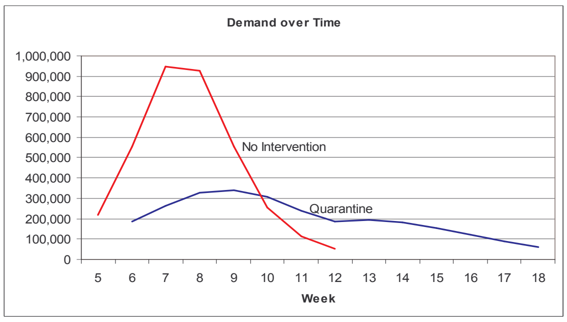


Figure 6: Effect of quarantine on the food requirement

Next, we study the effect of a quarantine on the supply chain network using the results of 10 different simulation runs with 200 PODs, 10 major facilities, and 20 supply points in the entire Metropolitan Atlanta Area (603 census tracts/demand nodes). The supply points and PODs are randomly assigned, but the major facilities are assigned to the most crowded census tracts of the 10 densely populated counties. Note that at most one major facility is assigned to each county.

The cumulative effect of a quarantine on the supply chain network is presented in Table 5. From Table 5, the reduction in total cost is higher when the shipment costs or facility related costs dominate since the reduction in total demand is almost fully reflected in the total cost in these settings. On the other hand, in the "medium" shipment cost setting, since both shipment costs and facility related costs comprise the total cost, the reduction in total demand is not fully reflected in the total cost.

Interestingly, from the third and fourth columns of Table 5, the quality of service decreases as the shipment costs decrease. This occurs because lower shipment costs increase the amount of demand served from a long distance. Additionally, we find that

Table 5: Comparison of 8-week quarantine with no intervention case

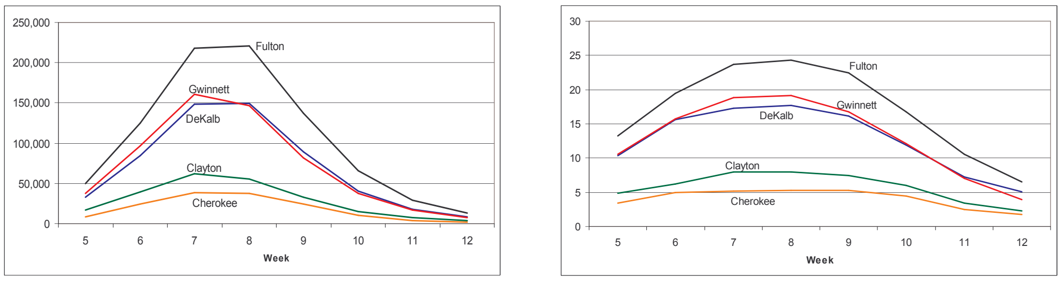
Shipment Cost	Cost Reduction in an 8-Week Quarantine Compared to No Intervention	Demand Served within 10 Miles	
		8-Week Quarantine	No Intervention
High	22.35%	99.79%	99.85%
Medium	18.74%	97.86%	99.66%
Low	25.97%	85.29%	94.99%

the quality of service is better under the no intervention case than under a quarantine (assuming all of the facilities can indeed operate at the estimated capacities). Since the average demand is lower under a quarantine, fewer facilities are operated in a given period, and this decreases the percentage of the demand served within 10 miles.

We provide a detailed analysis for the "medium" shipment cost case below. The demand and the number of open PODs over time under no intervention and the 8-week quarantine case can be seen for 5 counties in Figure 7 and 8. Under an 8-week quarantine policy, the number of open distribution centers is reduced by almost 50% which is consistent with the reduction in average demand. Both under no intervention and an 8-week quarantine, the number of open PODs start to increase earlier than demand does. Because the opening/closing costs are incurred only once, when the demand at a location is expected to increase, a nearby POD is opened earlier to utilize the benefit of shipping. This is mainly due to comparable shipment and facility operating costs. In the setting where the fixed operating cost dominates shipment cost, the number of open facilities closely follows the demand curve. Finally, proportional with the number of infections, most of the PODs are operated in densely populated counties.

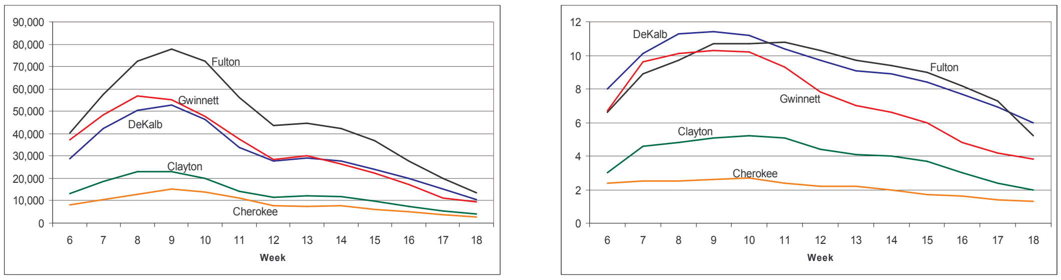
Figure 9 shows the average number of major facilities operated in each county. The major facilities in DeKalb and Cobb counties are the ones that are operated during almost the entire time horizon since they are among the most populated counties, and they have a better combined location than other alternatives.

Figures 10(a) and 10(b) show the minimum, average, and maximum open durations of the PODs for each county. The minimum open duration of the facilities is



(a) Demand over time (b) Number of open PODs over time

Figure 7: Demand and number of open PODs over time under the no intervention case



(a) Demand over time (b) Number of open PODs over time

Figure 8: Demand and number of open PODs over time under the 8-week quarantine

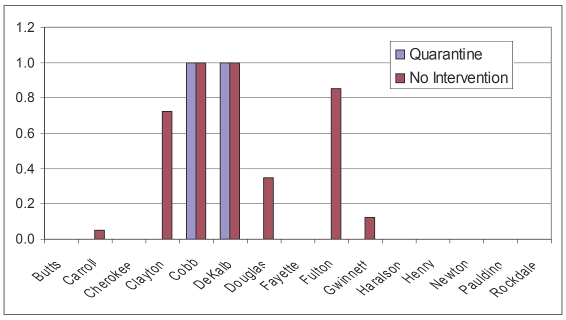
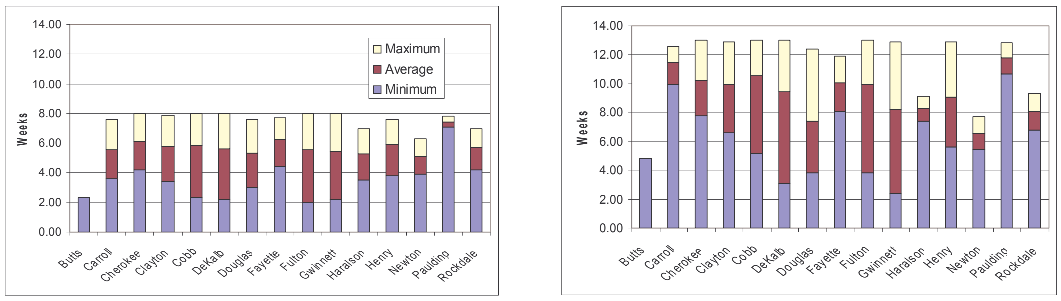


Figure 9: Average number of open major facilities

higher in the quarantine case because of a smoother demand curve that lasts for a longer time. In the no intervention case, some of the PODs are open when the demand peaks, and then they are closed when the demand decreases. The variability in the open duration of PODs is higher in the densely populated counties (Cobb, DeKalb, Fulton and Gwinnett) because the demand curve in these counties has a higher peak, increasing the number of opening/closing decisions.



(a) Duration of PODs under no intervention

(b) Duration of PODs under an 8-week quarantine

Figure 10: Minimum, average and maximum duration of open PODs

2.8 Effect of Influenza Pandemic on the Workforce Level

In this section, using the disease spread model, we analyze the impact of influenza pandemic and intervention strategies, namely, voluntary quarantine and school closure on the workforce.

The reduced active workforce level in critical services may result in secondary consequences causing greater impact than the influenza pandemic itself [67]. Influenza pandemic could result in as much as 33% workforce loss because of illness or the need to take care of infected family members [64]. “If the refineries lose 30 percent of their people, they have to shut down. Transport and delivery would be severely handicapped during a pandemic both because of gas shortages and the loss of workforce.” says Dr. Michael T. Osterholm, the director of the Center for Infectious

Disease Research and Policy at the University of Minnesota [56]. Gas shortages will also trigger interruptions in services. Food and water supplies may be interrupted, and individuals may also be unable to get to a grocery store.

In our analysis, we assume that work absenteeism occurs because of (i) illness, (ii) the need to care for infected family members, (iii) quarantine, or (iv) the need to take care of children in case of a school closure.

Under no intervention, work absenteeism may occur because of (i) and (ii). In the base case, we assume that 50% of symptomatic adults withdraw from work and stay home. First, we study the effect of different withdrawal rates from work on the disease spread. In Table 6, we see the effect of different withdrawal rates for working adults on the disease spread for $R_0 = 1.8$ assuming that one of the adults stay home if there is an infected children or elderly in the household. We observe that as the withdrawal rate increases the peak infectivity and IAR reduces and the peak time is delayed. Similar observation can be done for different R_0 values (see Tables 27 and 28 in Appendix A.11). This is reasonable because increased withdrawal rates decrease the number of interactions between the infected and susceptible individuals in the work places.

Table 6: Effect of different withdrawal rates from work on the disease spread for $R_0 = 1.8$

Withdrawal Rate	**Peak Infectivity**	**Peak Day**	**IAR**
25%	6.02%	48	70.79%
50%	5.00%	52	65.64%
75%	3.89%	57	58.77%
100%	2.80%	64	51.19%

Second, we analyze the impact of different withdrawal rates on the workforce loss. In Figure 11, we see how the work absenteeism changes over time for different withdrawal rates for $R_0 = 1.8$ (see Appendix A.12 for other R_0 values). From Figure 11, we observe that if the withdrawal rate is 50%, the peak work absenteeism is around 4.5%, and the peak occurs around day 50. As the withdrawal rate increases beyond 50%, the peak work absenteeism decreases. Higher withdrawal rates increase the percentage of symptomatic working adults that withdraw from work, but this also decreases the number of interactions in the work places.

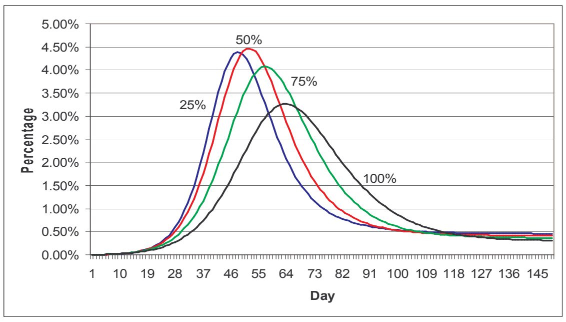


Figure 11: Effect of withdrawal rate on the work absenteeism for $R_0 = 1.8$

If the companies can manage their operations with conference calls instead of face-to-face meeting with the employees, this can reduce the number of infections in their workplaces and thus the work absenteeism. Even if telecommuting is not possible, instead of forcing the infected employees to come to work for business continuity, companies should consider sending them to home which may decrease the work absenteeism and also the overall IAR in the population. Furthermore, this may decrease the peak infectivity significantly, which is crucial for the load on the health services infrastructure.

Next, we analyze the effect of quarantine on the work absenteeism. In voluntary quarantine case, work absenteeism may occur because of (i), (ii) and (iii). Assuming that an 8-week voluntary quarantine is implemented, the effect of start time of the quarantine and the withdrawal rate on the peak work absenteeism for $R_0 = 1.8$ can be seen in Figure 12 (see Appendix A.13 for different length quarantines). Assuming that the withdrawal rate is 50%, the peak work absenteeism is minimized at around 3.5% if the voluntary quarantine is started at the beginning of week 3. Note that the peak work absenteeism is around 4.5% under no intervention case (see Figure

11). Similar to the observation in the no intervention case, if the withdrawal rate increases, peak work absenteeism decreases.

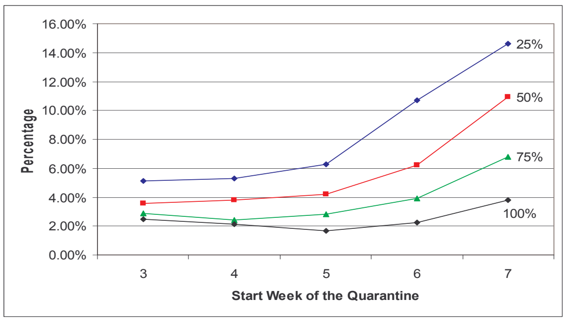


Figure 12: Effect of withdrawal rate and quarantine start time on the peak work absenteeism for an 8-week quarantine for $R_0 = 1.8$

From Figure 12, if the quarantine is started later than the optimal timing, peak work absenteeism can be higher compared to no intervention case, e.g., if an 8-week voluntary quarantine is started at the beginning of week 7, assuming that the withdrawal rate is 50%, the peak work absenteeism is around 11%. If the quarantine is started late, the number of infected people in the population is already high which results in quarantining a large number of people increasing the work absenteeism. This also supports the argument on starting the quarantine earlier than late (see Section 2.4).

Finally, we investigate the effect of school closures on the work absenteeism. Although work absenteeism may also occur because of (i) and (ii), the main reason for work absenteeism during school closures is (iv). In Figure 13, we see the effect of the start time of a 12-week school closure on the peak work absenteeism for $R_0 = 1.8$ assuming withdrawal rate for adults is 50%. We see that the peak work absenteeism is around 18%. Work absenteeism remains around 18% while the schools are closed. This is mainly because of the absenteeism of working adults who prefer staying home to take care of their children during school closures. Sadique et al. [93] also analyze the effect of school closures on the work absenteeism, and they make a similar observation. According to their results, around 16% work absenteeism occurs because of school closures.

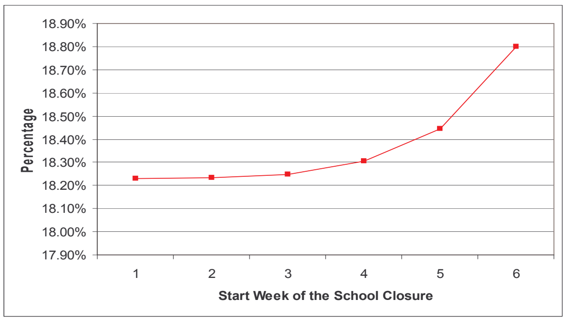


Figure 13: Effect of school closure start time on the peak work absenteeism for $R_0 = 1.8$

Our results on school closures also support the disruptiveness of school closures on the overall economy [90, 103]. This level of work absenteeism could result in secondary consequences that may cause a greater impact on the economy than the pandemic itself.

2.9 Conclusions and Future Directions

In this chapter, we construct a disease spread model with a spatial and an age-based structure for influenza pandemic that is helpful for developing intervention strategies and for preparedness planning. With the goal of designing a food distribution supply chain network during an influenza pandemic, we link the disease spread model to a facility location and resource allocation model and propose two solution approaches, namely, Deterministic Approach and Dynamic Update Approach. In the Deterministic Approach, the disease spread is estimated only at the beginning, and the food distribution network is constructed according to this estimate. In the Dynamic Update Approach, the estimates on the disease spread as well as the food distribution facility location and resource allocation decisions are updated over time.

Since the corresponding facility location problem (CMPH-FLP) is hard to solve for large instances, we design efficient algorithms to find near-optimal solutions. Our computational results indicate that the Hybrid Heuristic performs better when compared to the Add-Drop Heuristic, but for very large instances that cannot be handled by commercial optimization software, the Add-Drop Heuristic is the best alternative. We envision that a combined demand diffusion and resource allocation approach such as the one proposed in this paper could be useful in other applications, e.g., in the marketing operations using a Bass diffusion model [7] for demand and optimization to decide on the allocation of limited resources for distributing the products.

Models that predict the spread of the disease accurately help public health officials in developing efficient response plans ahead of time before the influenza pandemic hits. Estimating the disease specific parameters is one of the key issues in developing efficient response plans. While models such as the Deterministic Approach using estimates from earlier influenza pandemic cases can be used for advance planning purposes, dynamic approaches such as Dynamic Update Approach that generate updated estimates after the pandemic begins can be used to implement response plans.

We study voluntary quarantine as an intervention policy and find the best timing and length of the quarantine for different R_0 values. We conclude that an 8-week quarantine is equivalent to a 12-week quarantine (or a quarantine that is in effect during the entire course of the disease) in terms of reducing the peak infectivity for high R_0 values. For lower R_0 values, an 8-week quarantine may still be a good choice given the negative effect of prolonged quarantine times on public functions and morale, and the diminishing rate of benefits from long quarantines on the peak infectivity and infection attack rate (IAR). The optimal start time of a quarantine decreases both in the duration of the quarantine and the R_0 value. In addition to voluntary quarantine, we study school closure as an alternative social distancing measure and find that it may not be as effective as a voluntary quarantine. Since school closures can be more disruptive on the overall services and the economy [90, 103], e.g., if some parents have to leave the workforce to care for children who are out of school, we recommend limited-time voluntary quarantine as an effective intervention policy to public health officials.

Although the effect of a quarantine on IAR is limited, it can decrease the peak infectivity significantly, which is crucial for the continuity of the supply chains of goods and services. It can also reduce the probability of having capacity problems in various industries during an influenza pandemic. For example, in the food distribution supply network, the number of facilities operated decreases by almost half in the case of a quarantine when compared to the no intervention case. This significant reduction has several benefits including the reduced equipment and workforce requirements to operate the food distribution facilities where the workforce at each point in time is likely to be a scarce resource due to illness, fear of infection and the need to care for infected family members. In addition to decreasing the peak infectivity, quarantine delays the timing of the peak which is also important since a delayed peak offers more time for preparedness efforts. These benefits would also apply to health services [89]. The results of our research have been incorporated into the manual of ARC-MAC on food distribution planning during an influenza pandemic [3].

In this chapter, we analyzed the food distribution supply chain during an influenza pandemic. Designing medicine/vaccine distribution supply chains and analyzing the effect of influenza pandemic in terms of supply chain disruptions are two other important problems that need to be addressed to develop efficient response plans. In our disease spread model, we did not assume any seasonal effects or viral evolution, which may change the spread pattern of the virus and is the focus of parallel work [70]. Another future direction is optimizing the intervention policies such as distribution of vaccines and antivirals. This may decrease the number of infected people as well as the amount of food needed.

CHAPTER III

OPTIMAL JOB SPLITTING ON A MULTI-SLOT MACHINE WITH APPLICATIONS IN THE PRINTING INDUSTRY

3.1 Introduction

In this chapter, we consider the *Job Splitting Problem* (JSP) which has applications in the printing industry, one of the largest manufacturing industries in the United States with products ranging from newspapers, magazines, books, brochures, labels, newsletters, postcards, memo pads, business order forms, checks, maps, and packaging. JSP and the methods we develop for its solution are especially applicable in commercial lithographic printing establishments, which print a wide variety of products including newspaper inserts, catalogs, pamphlets, and advertisements and make up the largest segment of the printing industry, accounting for about 31% of employment and 39% of total establishments [11].

In JSP, there are n types of items (e.g., business cards, labels, brochures, retail advertisement coupons, etc.) with demands/orders $d_1, \ldots, d_n$ to be processed on a machine (press) which has m slots. In each "run", at most one type of item is assigned to each slot and the machine continues to process that type in that slot until the end of the run. A new run is initiated with a "setup" of cost c_s, also referred to as makeready, which determines the "pattern/configuration" for the type-slot assignments in that run. A production plan consists of a set of runs, each determined by three attributes: (i) type-slot assignment, (ii) quantity assigned to each slot, that is the quantity that will not go to "waste", and (iii) length, which is the largest quantity assigned to (equivalently, the quantity produced in) any one slot in that run. Our goal is to create a production plan to fulfill the demand so as to minimize the total cost of setups and "waste", which is the amount of production in excess of demand, with a unit waste cost of c_w. In JSP, waste occurs due to a mismatch in the order quantities. Waste reduction has several benefits, including less spending in raw material and disposal costs, better utilization of equipment and resources, and reduced environmental pollution. Note that minimizing waste is equivalent to minimizing the total length of all the runs in the production plan.

JSP is related to the well known one-dimensional *Cutting Stock Problem* (CSP) [28, 44, 50], which is motivated by paper manufacturing. A paper machine produces large reels (rolls) of paper. The process of cutting the reels into smaller rolls of paper based on customer specified widths is called trimming. The reels are cut into rolls according to patterns which are combinations of different roll widths (for a given diameter) [69]. For example, a reel of width 200 can be cut into 4 rolls of width 40 and one roll of width 35, resulting in 5 inches ($200 - 4\times 40 - 35 = 5$) of waste (trim loss). Given a demand vector for rolls of different widths, the goal in CSP is to generate a set of patterns for cutting the reels to satisfy the demand while minimizing the waste, or equivalently, while minimizing the number of reels used.

Using different patterns to cut the reels requires a setup on the winder (the machine used to cut the reels) since the position of the knives between consecutive patterns must be changed [69]. Winder setups may be time consuming and costly, especially if they must be done manually. CSP with the secondary objective of minimizing the number of patterns used, while still minimizing the waste, is referred to as the *Pattern Minimization Problem* (PMP) [80]. McDiarmid [80] shows that even a special case of PMP, where any two rolls fit into a reel but none of the three, is strongly NP-hard. Diegel et al. [25] identify necessary conditions in order to combine two or more patterns that reduces the number of patterns starting with a given solution while Haessler [49] develops a sequential heuristic procedure for PMP. Yanasse and Limeira [113] propose a hybrid procedure to reduce the number of patterns, and Foerster and Wascher [38] develop the KOMBI heuristic, which allows for different types of combinations to combine two or more patterns in order to reduce the number of patterns. Umetani et al. [104] solve PMP with an upper bound on the number of different patterns that can be used. Vanderbeck [106] develops an exact algorithm for PMP based on a column generation idea, which is strengthened by Alves and Valerio de Carvalho [1].

PMP and JSP seem related since both problems focus on reducing the waste and the number of setups. However, there are significant differences. First, while the orders have different widths in PMP, in JSP they all have essentially the same width from a modeling perspective since we can assign one order per slot. For example, consider a machine with 2 slots and two types of items with demands 50 and 75. If slot 1 is configured to produce type 1 and slot 2 is configured to produce type 2, then the waste occurs if the length of the run is more than 50. Hence, in JSP the waste occurs due to a mismatch in the order quantities. By contrast, in PMP the waste occurs due to potential mismatch in the widths of the ordered rolls. Second, while the goal in PMP is to minimize the waste with a secondary objective of minimizing the number of setups, in JSP the goal is to minimize the weighted sum of the waste and setup costs. Hence, the objective in PMP corresponds to a special case of JSP where the waste cost is significantly larger than the setup cost.

To the best of our knowledge, only a special case of JSP, where the number of slots on the machine is 4, has been studied in the literature by Teghem et al. [102] in the context of grouping and printing book covers while minimizing the total production cost. In Teghem et al. [102], the plate used in the printing process can accommodate up to four covers at a time. Hence, the number of slots (m) in JSP is set to 4. Teghem et al. [102] develop a heuristic algorithm based on simulated annealing. Unfortunately, they have not been able to obtain promising results (in terms of the solution quality and solution time) when the number of types exceeds 5.

In this chapter, we propose two efficient (fast) and effective heuristics that are capable of finding near-optimal solutions for large instances of JSP such as $m = 30$ slots and $n = 2086$ orders in less than a minute. In addition, we propose a new linear integer programming formulation which is proved to be more efficient than the one proposed by Teghem et al. [102]. We generate several cuts to strengthen the linear integer formulation and develop preprocessing steps that help in solving the problem optimally.

The remainder of the chapter is organized as follows. In Section 3.2, we give a formal definition of the problem, provide an illustrative example and present some observations about the structure of the problem. In Section 3.3, we prove that JSP is strongly NP-hard and discuss a special case. In Section 3.4, we present three different mathematical modeling approaches for JSP, namely, a nonlinear program and two linear integer programs. In Section 3.5, we propose two heuristics and prove that the first heuristic returns a 2-approximate solution. In Section 3.6, using the results of the proposed heuristics, we develop preprocessing steps to improve the solution time of the linear integer program. In Section 3.7, we present the results of the computational experiments conducted on real-world and randomly generated instances. Finally, we discuss extensions and future research directions.

3.2 Preliminaries

In this section, we provide an example to illustrate JSP and provide some basic observations about the structure of the problem based on special cases. Our notation is summarized in Table 7.

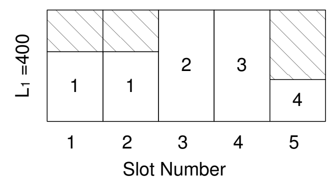
The following numerical example provides insights on the order-slot (or type-slot) assignment, waste and "quantity assigned to a slot" concepts.

Example 3.1. *Consider a machine with 5 slots and 4 types of items with demands*

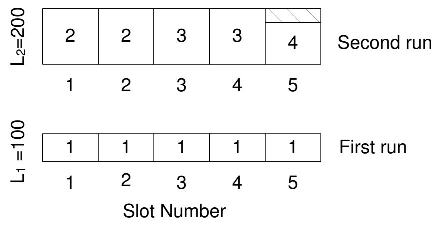
Table 7: Notation for JSP

$N = \{1, \ldots, n\}$	:	Set of types
m	:	Number of slots on the machine
d_i	:	Demand for type i, $i \in N$
L_j	:	Length of (the largest quantity assigned to any one of the slots in) run j
c_s	:	Setup cost for initiating a new run
c_w	:	Cost per unit of excess production

$d_1 = 500, d_2 = 400, d_3 = 400, d_4 = 150$. *We can initiate a run with type-slot assignment [1 1 2 3 4] and length 400. This indicates that slots 1 and 2 are configured to produce type 1, slot 3, slot 4 and slot 5 are configured to produce types 2, 3 and 4, respectively. This single run is a feasible production plan with excess production in slots 1, 2 and 5 (see Figure 14(a)) since the quantity assigned to these slots is less than the length of the run. Another feasible production plan is the one with two runs with type-slot assignments [1 1 1 1 1] and [2 2 3 3 4] and with lengths 100 and 200, respectively (see Figure 14(b)). In this solution, 2 runs are made with only 50 units of excess production. The objective in JSP is forming a production plan that takes into account the trade-off between waste and setup costs.*



(a)



(b)

Figure 14: Two different feasible production plans for Example 3.1

Assuming that there are r runs in a production plan, the cost function can be written as follows:

$$c_s r + c_w(m \sum_{j=1}^{r} L_j - \sum_{i \in N} d_i). \tag{15}$$

In (15), the first term is the total setup cost of the runs and the second term is the waste cost. We calculate the wasted quantity by subtracting the total demand from the total production. Since the total demand is constant, minimizing (15) is equal to minimizing

$$c_s r + c_w m \sum_{j=1}^{r} L_j. \tag{16}$$

Dividing (16) by $c_w m$, we obtain the following simpler cost function

$$cr + \sum_{j=1}^{r} L_j. \tag{17}$$

In (17), c is the scaled setup cost and it is defined as $c := \frac{c_s}{c_w m}$. From now on, we consider (17) as the objective function for JSP, which is the sum of the total setup cost and the total length of all the runs. In the rest of the chapter, we use setup cost to refer to the scaled setup cost.

The next two observations present upper and lower bounds on the number of runs in an optimal solution to JSP.

Observation 3.1. *The number of runs in a feasible solution is greater than or equal to* $\lceil \frac{n}{m} \rceil$.

Observation 3.1 follows since we can satisfy the demand of at most m types in each run. Teghem et al. [102] also make a similar observation for $m = 4$.

Note that when the setup cost is very high ($c \gg 0$), the setup cost dominates the objective function and transforms JSP to a problem where the objective is to find a solution with the smallest number of runs/setups. Hence, Observation 3.1 also indicates that there exists an optimal solution to JSP with $\lceil \frac{n}{m} \rceil$ runs when $c \gg 0$. Furthermore, when $c \gg 0$ and $n \leq m$, the optimal solution has a single run. In Section 3.3, we present a polynomial-time algorithm for finding the minimum waste solution with a single run.

Next, we show that there is an optimal solution with at most n runs. Teghem et al. [102] also mention a similar result. They claim that this is one of the extreme solutions (the other being the solution with the minimum number of runs), and they propose producing the same type in each slot of a run. In that case, the length of a run is equal to $\lceil \frac{d_i}{m} \rceil$ where i is the index of the type produced in that run. However, this solution is not extreme (or optimal for $c = 0$) since the waste is not minimized in this production plan. For example, consider a machine with 2 slots and 2 types of items with demands $d_1 = 7, d_2 = 9$. Producing only one type in a run results in two runs with type-slot assignments [1 1] and [2 2] and with lengths 4 and 5, respectively and results in a total waste of 2. However, the extreme solution, which has minimum (zero in this case) waste, is with two runs with type-slot assignments [1 2] and [2 2] and with lengths 7 and 1, respectively. We prove this result by a constructive proof which also illustrates a possible structure of the solution in this extreme case.

Observation 3.2. *A minimum waste solution can be obtained in at most n runs, i.e., there exists an optimal solution to JSP with at most n runs.*

Proof. We know that the total length of the runs in a feasible solution is at least $\lceil \frac{\sum_{i \in N} d_i}{m} \rceil$. Let $L := \lceil \frac{\sum_{i \in N} d_i}{m} \rceil$. We prove that the demand can be satisfied in at most n runs with total length L, which is a minimum waste solution since L is a lower bound on the total length of the runs.

We construct a solution by assigning the types to the slots from slot 1 with type 1 until either the quantity assigned to a slot reaches L or the demand of the current type is satisfied. If the quantity assigned to a slot reaches L, then we continue assigning the current type to the next slot. If the demand of a type is satisfied, we start assigning another type to the current slot without producing any waste. We continue the above process until all the types are assigned to m slots. The first run is determined by the first type assigned to each slot. After the first run is initiated, when the type assigned to a slot changes, a new run is initiated after changing the assigned type in this slot. The types assigned to other slots remain same. There are $n-1$ places where a new run is initiated. A new run is not initiated for the last type, type n, assigned to slots as described above since the remaining items are counted as waste. Including the first run, there are n runs. Thus, the total number of runs is n. More formally, the structure of the runs can be defined as follows. Let k_j^i be the index of the type assigned to slot j in the i^{th} order and d_j^i be the quantity of this type assigned to this slot. Then, the type-slot assignment for the first run is $[k_1^1 \; k_2^1 \; \ldots \; k_m^1]$ and the length of this run is equal to $\min_{j \in \{1,\ldots,m\}}\{d_j^1 \mid k_j^1 \neq n\}$. Let e_{jt}^i be the total quantity of type k_j^i produced in slot j in the first t runs. Assuming that the type-slot assignment for run t is $[k_1^{i_t^1} \; k_2^{i_t^2} \; \ldots \; k_m^{i_t^m}]$, the length of this run is $\min_{j \in \{1,\ldots,m\}}\{d_j^{i_t^j} - e_{j,t-1}^{i_t^j} \mid k_j^{i_t^j} \neq n\}$. The type-slot assignment for the next run, run $t+1$, is $[k_1^{i_{t+1}^1} \; k_2^{i_{t+1}^2} \; \ldots \; k_m^{i_{t+1}^m}]$ where $i_{t+1}^j = i_t^j + 1$ if $j \in \operatorname{argmin}_{j \in \{1,\ldots,m\}}\{d_j^{i_t^j} - e_{j,t-1}^{i_t^j} \mid k_j^{i_t^j} \neq n\}$ and $i_{t+1}^j = i_t^j$, otherwise. The length of this run is $\min_{j \in \{1,\ldots,m\}}\{d_j^{i_{t+1}^j} - e_{jt}^{i_{t+1}^j} \mid k_j^{i_{t+1}^j} \neq n\}$. □

When $c = 0$, JSP's objective turns into finding a solution with the smallest waste. Observation 3.2 shows that we can find a minimum waste solution with at most n runs. Hence, there exists an optimal solution to JSP with n runs when $c = 0$ and this solution can be found in polynomial time with the algorithm described in the proof of Observation 3.2. This result is summarized in the following corollary.

Corollary 3.1. *When $c = 0$, there exists an optimal solution for JSP with n runs and this solution can be found in polynomial time.*

In the light of Observations 3.1 and 3.2, we consider models and solutions with at least $\lceil \frac{n}{m} \rceil$ and at most n runs in the remainder of the paper.

3.3 Complexity and a Special Case

In this section, we show that JSP is strongly NP-hard and discuss a special case (in addition to those discussed in the previous section) that is solvable in polynomial time.

Theorem 3.1. *JSP is strongly NP-hard.*

Proof. McDiarmid [80] shows that a special case of PMP, where any two types fit into a reel but none of the three do, is strongly NP-hard. McDiarmid [80] considers this case since it is the simplest case of PMP that is not trivial. In this setting, the minimum number of reels is $\lceil \frac{\sum_{i \in N} d'_i}{2} \rceil$ where d'_i is the demand of type i. However, finding a solution with the minimum number of distinct patterns among the minimum number of reel solutions is NP-hard. This special case of PMP corresponds to a special case of JSP where the number of slots is 2, $0 < c \ll \frac{1}{m}$, and $d_i = d'_i$. □

Next, we study a special case of JSP that provides the motivation behind the *Multi-Run Heuristic* in Section 3.5.1. In this special case, $c \gg 0$ and the number of types is not greater than the number of slots, that is, $n \leq m$. Since the setup cost is very high, the optimal solution has a single run. We call the special case of finding the minimum waste solution with a single run (when $n \leq m$) the *Single Run Problem.* Although JSP is strongly NP-hard, we prove that the *Single Run Problem* can be solved in polynomial time by the *Single Run Algorithm* in Algorithm 1. Let y_i denote the number of slots assigned to type i.

Algorithm 1 *Single Run Algorithm.*

```
1: Set $y_i = 1$ for all $i \in N$.
2: Check whether $\sum_{i \in N} y_i < m$. If yes, go to Step 3. Otherwise, go to Step 4.
3: Calculate $\lceil \frac{d_i}{y_i} \rceil$ for all $i$. Let $j \in \text{argmax}_{i \in N}\{\lceil \frac{d_i}{y_i} \rceil\}$. Increase $y_j$ by 1. Go to Step
   2.
4: Assign the types to the slots based on $y_i$'s. For type $i$, assign a quantity of $\lceil \frac{d_i}{y_i} \rceil - 1$
   to $y_i \lceil \frac{d_i}{y_i} \rceil - d_i$ slots and assign a quantity of $\lceil \frac{d_i}{y_i} \rceil$ to the remaining $y_i - y_i \lceil \frac{d_i}{y_i} \rceil + d_i$
   slots.
```

Theorem 3.2. *The Single Run Algorithm solves the Single Run Problem in polynomial time ($O(m \log n)$), where the solution has a single run with length $L = \max_{i \in N}\{\lceil \frac{d_i}{y_i} \rceil\}$.*

Proof. First, we show the run time complexity of the algorithm. In each iteration, the number of slots assigned increases by one. Thus, the algorithm terminates in $O(m)$ iterations. The first iteration takes $O(n \log n)$ time due to the sorting of the n items. In the remaining iterations, sorting can be updated by moving the item with a newly assigned slot down the list, which can be done in $O(\log n)$ time using a binary search tree. Thus, the overall complexity of the algorithm is $O(m \log n)$.

Next, we show that the algorithm terminates with the minimum waste solution. By contradiction, assume that the solution found by the *Single Run Algorithm* does not have minimum waste. Then, there exists an assignment of slots $(y'_1, ..., y'_n)$ to types such that the length of the run, say L', is less than L. Since $n \leq m$, $\sum_{i \in N} y'_i = \sum_{i \in N} y_i = m$. Let $I := \{i : \lceil \frac{d_i}{y_i} \rceil \leq L'\}$ and $J := \{j : \lceil \frac{d_j}{y_j} \rceil > L'\}$. J is nonempty because of the assumption that the solution found by the *Single Run Algorithm* does not have minimum waste. Moreover, I is nonempty since $\sum_{i \in N} y'_i = \sum_{i \in N} y_i = m$. Obviously, $\sum_{i \in I} y'_i < \sum_{i \in I} y_i$, otherwise, there exists $j \in J$ such that $y_j \geq y'_j$ which means $\lceil \frac{d_j}{y_j} \rceil \leq L'$. However, this contradicts with the definition of J. $\sum_{i \in I} y'_i < \sum_{i \in I} y_i$ implies that there exists $k \in I$ such that $y_k > y'_k$. This means $L' \geq \lceil \frac{d_k}{y'_k} \rceil \geq \lceil \frac{d_k}{y_k} \rceil$. In addition, $L' \geq \lceil \frac{d_k}{y_k - 1} \rceil$ since $y_k - 1 \geq y'_k$.

Let $h \in J$. Then, $\lceil \frac{d_h}{y_h} \rceil > L' \geq \lceil \frac{d_k}{y_k - 1} \rceil$. In this case, at the iteration of the algorithm where the number of slots assigned to type k is increased from $y_k - 1$ to y_k, $y_k - 1$ slots are assigned to type i and $\hat{y}_h$ slots are assigned to type h for some $\hat{y}_h$ where $\hat{y}_h \leq y_h$. At this iteration, $\lceil \frac{d_h}{\hat{y}_h} \rceil > \lceil \frac{d_k}{y_k - 1} \rceil$ which contradicts with the second step of the algorithm. This extra slot should not be assigned to type k. $\square$

3.4 Mathematical Models

In this section, we present mathematical models for JSP, namely, a nonlinear integer formulation (Section 3.4.1), which is compact and intuitive but difficult to solve for large instances, and two linear integer formulations (Section 3.4.2), which are computationally more efficient.

3.4.1 A Nonlinear Integer Formulation

JSP can be easily formulated as a nonlinear integer program [102] with the following decision variables:

$$r_j = \begin{cases} 1, & \text{if run } j \text{ is initiated} \\ 0, & \text{otherwise} \end{cases} \qquad j \in \{1, \dots, n\},$$

$$z_{ij} = \text{number of slots assigned to type } i \text{ in run } j \qquad i \in N,\ j \in \{1, \dots, n\},$$

$$L_j = \text{length of run } j \qquad j \in \{1, \dots, n\}.$$

The nonlinear formulation is as follows:

$$\begin{aligned} \text{NIP: Minimize} \quad & c\sum_{j=1}^{n} r_j + \sum_{j=1}^{n} L_j && && (18)\\ \text{subject to} \quad & \sum_{i \in N} z_{ij} \le m r_j && j \in \{1, \dots, n\}, && (19)\\ & \sum_{j=1}^{n} L_j z_{ij} \ge d_i && i \in N, && (20)\\ & z_{ij} \ge 0 \ \text{ and integer} && i \in N,\ j \in \{1, \dots, n\}, && (21)\\ & L_j \ge 0 \ \text{ and integer} && j \in \{1, \dots, n\}, && (22)\\ & r_j \in \{0, 1\} && j \in \{1, \dots, n\}. && (23) \end{aligned}$$

Constraints (19) ensure that at most m slots are assigned in a run. Constraints (20) ensure that the demand is satisfied. Constraints (21), (22) and (23) represent integrality restrictions.

Unfortunately, even the continuous relaxation of this formulation, where the binary and integer variables are replaced by continuous variables, is not convex and is very difficult to solve.

3.4.2 Linear Integer Formulations

In this section, we present two linear integer formulations for JSP. The first linear integer formulation, which is intuitive and also proposed by Teghem et al. [102], leads to a high number of nodes in a branch and bound scheme (due to symmetry) since the subproblems solved at each node of the branch and bound tree have multiple optimal solutions with the same objective function value. With the goal of eliminating such inefficiencies, we propose a second formulation, add simple cuts to strengthen the formulation and utilize special branching rules to improve the solution time. In addition to r_j and L_j as defined in Section 3.4.1, the variables z_{ijk} and p_{ijk} ,which are defined below, are used in the first formulation.

$$z_{ijk} = \begin{cases} 1, & \text{if slot } k \text{ of run } j \text{ is assigned to type } i \\ 0, & \text{otherwise} \end{cases} \qquad i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\},$$

$$p_{ijk} = \text{quantity of type } i \text{ assigned to slot } k \text{ of run } j \qquad i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}.$$

The first formulation is as follows:

$$\begin{aligned}
\text{IP1: Minimize} \quad & c\sum_{j=1}^{n} r_j + \sum_{j=1}^{n} L_j & & (24)\\
\text{subject to} \quad & \sum_{j=1}^{n}\sum_{k=1}^{m} p_{ijk} = d_i & i \in N, & \quad (25)\\
& \sum_{i\in N} z_{ijk} \le r_j & j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}, & \quad (26)\\
& p_{ijk} \le L_j & i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}, & \quad (27)\\
& p_{ijk} \le d_i z_{ijk} & i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}, & \quad (28)\\
& p_{ijk} \ge 0 \ \text{ and integer} & i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}, & \quad (29)\\
& L_j \ge 0 \ \text{ and integer} & j \in \{1,\dots,n\}, & \quad (30)\\
& z_{ijk} \in \{0,1\} & i \in N,\ j \in \{1,\dots,n\},\ k \in \{1,\dots,m\}, & \quad (31)\\
& r_j \in \{0,1\} & j \in \{1,\dots,n\}. & \quad (32)
\end{aligned}$$

Constraints (25) ensure that the demand of each type is satisfied. The restriction that a slot can be assigned to at most one type is represented by (26). The quantity assigned to a slot cannot exceed the length of the run, which is guaranteed by (27). Constraints (28) ensure that the quantity assigned to a slot is zero for a given type if that slot is not assigned to that type. Constraints (29)-(32) are the integrality restrictions.

In practice, while the number of slots assigned to a type in a given run is important, the exact location of these slots does not matter. However, in this formulation we can have multiple solutions with the same number but different positions of slots, all having the same objective function value. The existence of multiple optima at each LP relaxation causes unnecessary branching in the branch and bound tree. To illustrate, consider a simple example where $m = 3, n = 5$, and $(d_1, d_2, d_3, d_4, d_5) = (100, 100, 100, 600, 600)$. The optimal solution of the LP relaxation has two runs and zero waste with run lengths 400 and 100. This solution can be represented by many different assignments of variable values. For example:

$$z_{121} = z_{222} = z_{323} = z_{411} = z_{512} = 1,\ z_{413} = z_{513} = 0.5;$$
$$z_{121} = z_{222} = z_{323} = z_{412} = z_{513} = 1,\ z_{411} = z_{511} = 0.5;$$
$$z_{121} = z_{222} = z_{323} = z_{411} = z_{513} = 1,\ z_{412} = z_{512} = 0.5.$$

We propose a second formulation that does not include such symmetric solutions and allows us to utilize special branching strategies to improve the solution time. The definitions of the variables r_j and L_j remain unchanged from Section 3.4.1. The additional variables are:

$$x_{ijk} = \begin{cases} 1, & \text{if } k \text{ slots of run } j \text{ are assigned to type } i \\ 0, & \text{otherwise} \end{cases} \qquad i \in N,\ j \in \{1, \ldots, n\},\ k \in \{0, 1, \ldots, m\},$$

q_{ijk} = quantity of type i assigned to slots in run j if k slots of run j are assigned to type i

$i \in N,\ j \in \{1, \ldots, n\},\ k \in \{0, 1, \ldots, m\}.$

The second formulation is:

$$
\begin{aligned}
\text{IP2: Minimize} \quad & c\sum_{j=1}^{n} r_j + \sum_{j=1}^{n} L_j && && (33)\\
\text{subject to} \quad & \sum_{j=1}^{n}\sum_{k=0}^{m} q_{ijk} = d_i && i \in N, && (34)\\
& \sum_{i\in N}\sum_{k=0}^{m} k x_{ijk} \le m r_j && j \in \{1,\dots,n\}, && (35)\\
& q_{ijk} \le k L_j && i \in N,\ j \in \{1,\dots,n\},\ k \in \{0,1,\dots,m\}, && (36)\\
& q_{ijk} \le d_i x_{ijk} && i \in N,\ j \in \{1,\dots,n\},\ k \in \{0,1,\dots,m\}, && (37)\\
& \sum_{k=0}^{m} x_{ijk} = 1 && i \in N,\ j \in \{1,\dots,n\}, && (38)\\
& q_{ijk} \ge 0 \text{ and integer} && i \in N,\ j \in \{1,\dots,n\},\ k \in \{0,1,\dots,m\}, && (39)\\
& L_j \ge 0 \text{ and integer} && j \in \{1,\dots,n\}, && (40)\\
& x_{ijk} \in \{0,1\} && i \in N,\ j \in \{1,\dots,n\},\ k \in \{0,1,\dots,m\}, && (41)\\
& r_j \in \{0,1\} && j \in \{1,\dots,n\}. && (42)
\end{aligned}
$$

Constraints (34) guarantee that the demand of each type is satisfied. Constraints (35) ensure that at most m slots are assigned to the types in each run. If k slots are assigned to type i in run j, then the quantity of type i assigned to slots in this run cannot be greater than kL_j, which is guaranteed by constraints (36). If the number of slots assigned to a type is k in a run, then only the corresponding quantity variable (q_{ijk}) can be nonzero for that type, which is ensured by constraints (37). Constraints (38) determine the number of slots assigned to a type in a run. Similar to the first formulation, constraints (39)-(42) are the integrality restrictions.

Remark 3.1. *The integrality of the variables q_{ijk} and L_j affects the optimal objective value by at most $n-1$. That is, if we solve (IP2) after relaxing the integrality constraints of L_j and q_{ijk} and round the optimal objective value of this relaxation up, we obtain a lower bound on the optimal integer solution. We can obtain a feasible integer solution by rounding the values of the variables as follows. We round L_j's up. For the variables q_{ijk}, first we round all of them up and then decrease some of them by 1 so that constraints (34) are satisfied. That is, let q^*_{ijk}'s denote the optimal solution* *if we relax the integrality constraints of* q_{ijk} *and* L_j. *Then, if* $\sum_{j=1}^{n}\sum_{k=0}^{m}\lceil q^*_{ijk}\rceil = d_i+\epsilon$ *for some nonnegative integer* ϵ, *decrease* ϵ *of* $\lceil q^*_{ijk}\rceil$*'s by 1. After these rounding steps, we obtain a feasible integer solution. Rounding* L_j*'s up increases the optimal objective value found for the relaxation by less than* n. *Thus, the feasible integer solution we obtain is greater than the optimal integer solution by at most* $n-1$. *This is negligible when compared to the magnitude of the demand of the types in real-world instances. Thus, in the rest of the paper, we assume that* q_{ijk}*'s and* L_j*'s are continuous variables.*

Note that the symmetry problem that appears in (IP1) is eliminated by changing the definition of z_{ijk}'s which are replaced by x_{ijk}'s in (IP2). While solving (IP2) by a branch and bound algorithm, we can use special branching rules on x_{ijk}'s as proposed by Nemhauser and Wolsey [86]. Instead of single variable branching, the following branching scheme can be used. Suppose x^*_{ijk}'s are the optimal values of x_{ijk}'s at some node of the branch and bound tree. Let $\sum_{k=0}^{m} kx^*_{ijk} = x^*$ for type i in run j for some x^*. Let $k' = \lfloor x^*\rfloor$. Then, the new scheme is to branch into two nodes: one with $\sum_{k=0}^{k'} x_{ijk} = 1$ and the other with $\sum_{k=k'+1}^{m} x_{ijk} = 1$.

Next, we present two cuts that strengthen (IP2). In fact, they are valid for both (IP1) and (IP2). The first cut is:

$$L_j \geq L_{j+1} \qquad j \in \{1,\ldots,n-1\}. \tag{43}$$

In both formulations, runs can be in any order. That is, suppose that we find a feasible solution for the LP relaxation at some node of the branch and bound tree, and the lengths of the runs in this solution are 100, 50 and 20. Then, we have 3! equivalent solutions that give the same objective value: $L_1 = 100, L_2 = 50, L_3 = 20$; $L_1 = 100, L_2 = 20, L_3 = 50$; $L_1 = 50, L_2 = 100, L_3 = 20$; $L_1 = 50, L_2 = 20, L_3 = 100$; $L_1 = 20, L_2 = 50, L_3 = 100$; $L_1 = 20, L_2 = 100, L_3 = 50$. This causes unnecessary branching in the branch and bound tree. To avoid this symmetry problem, we add constraints (43) to sort the runs in the nonincreasing order of their lengths.

The second cut we propose is as follows:

$$\sum_{i \in N} \sum_{k=1}^{m} q_{ijk} \leq mL_j \qquad j \in \{1, \ldots, n\}. \tag{44}$$

Quantity assigned to all slots in run j can be at most mL_j. However, in the following example, we see that the optimal solution of the LP relaxation can violate this. Consider a simple instance where $m = 2, n = 3$, and $(d_1, d_2, d_3) = (100, 200, 200)$. The optimal solution of the LP relaxation has two runs and zero waste with the following nonzero variables: $L_1 = 100, L_2 = 50, x_{111} = 1, x_{211} = 0.5, x_{311} = 0.5, x_{210} = 0.5, x_{310} = 0.5, x_{222} = 0.5, x_{322} = 0.5, x_{220} = 0.5, x_{320} = 0.5, q_{111} = 100, q_{211} = 100, q_{311} = 100, q_{222} = 100, q_{322} = 100$. In this solution, the total quantity assigned to slots in the first run is 300 whereas it can be at most 200. Similarly, the total quantity assigned is 200 in the second run whereas it can be at most 100. Constraints (44) eliminate such solutions.

Thus, from now on we concentrate on (IP2) with the additional constraints (43) and (44). We denote this formulation by (IP2$'$).

Although we add some cutting planes to the basic formulation, we still cannot solve some of the smallest real-world instances where $m = 20$ and $n = 54$. However, the formulations are useful in providing us with lower bounds on the optimal solution and evaluating the performance of the heuristics. Hence, we focus our attention to two areas: (1) constructing "good" heuristics for JSP (Section 3.5), and (2) strengthening (IP2$'$) and improving the lower bound (Section 3.6).

3.5 Heuristics for JSP

In this section, we propose two heuristics for JSP, namely, the *Multi-Run Heuristic* (MRH) and the *Balanced Cost Heuristic* (BCH). MRH, which is based on the idea of the *Single Run Algorithm* discussed in Section 3.3, finds a 2-approximate solution to

JSP. BCH is inspired by the *Part-Period Balancing*, which is a lot-sizing algorithm proposed by DeMatteis and Mendoza [23].

Before running the heuristics, we first process all the instances by applying an initial heuristic step. We check whether there are m types with equal demand, say d: if yes, then we produce these m types in a single run of length d (with zero waste) and take them out of the demand set. We repeat this preprocessing step as many times as possible, and then we apply the two heuristics to the remaining set of types. Our experimental results indicate that both of the heuristics (and their variations) run in seconds and neither of the variations dominate in terms of solution quality. Hence, our approach is to apply both heuristics (including their variations) and take the best solution. Here, we present the basic versions of the heuristics, and we discuss their variations in Appendix B.

3.5.1 Multi-Run Heuristic (MRH)

In this section, we describe the basic algorithm for MRH. In general, we do not know how many runs there are in an optimal solution, but we have upper and lower bounds on the number of runs as provided in Section 3.2. In the *Multi-Run Heuristic*, the idea is to solve the *Single Run Problem* with mr slots for all $r \in \{\lceil \frac{n}{m} \rceil, \ldots, n\}$ and then choose the solution with the best objective function value. Given the number of runs, say r', $cr' + \lceil \frac{\sum_{i=1}^{n} d_i}{m} \rceil$ is a lower bound on the optimal objective function value. Thus, for $r = r'$, the objective function value of the solution found using MRH is at least $cr' + \lceil \frac{\sum_{i=1}^{n} d_i}{m} \rceil$. One way to terminate MRH early is to check if this value is greater than the cost of the best solution found so far. If this is the case, we know that for r values where $r \geq r'$, MRH will not be able to find a better solution. The pseudocode for the *Multi-Run Heuristic* (MRH) is presented in Algorithm 2. The solution obtained when MRH is terminated has *index* runs and *min* total cost.

Next, we prove that MRH finds a 2-approximate solution to JSP.

Algorithm 2 *Multi-Run Heuristic.*

```
1: Set $min = cn + \lceil \frac{\sum_{i=1}^{n} d_i}{m} \rceil$ and $index = 0$.
2: for $r = \lceil \frac{n}{m} \rceil$ to $n$ do
3:     Suppose that we have a Single Run Problem with $mr$ slots and $n(\leq mr)$
       types. Apply the Single Run Algorithm to this problem. Assign the quantity
       of types to slots as explained in Section 3.3.
4:     Sort the slots in a nondecreasing order according to the quantity assigned to
       them.
5:     for $j = 1$ to $r$ do
6:         Assign the slots $m(j-1)+1, \ldots, mj$ to the $j^{th}$ run.
7:     end for
8:     The length of run $j$ in this solution is the quantity assigned to slot $m(j-1)+1$.
       Calculate the total cost using (17) and let current be the total cost.
9:     if $min > current$ then
10:        $min = current$ and $index = r$.
11:    end if
12:    if $min \leq c(r+1) + \lceil \frac{\sum_{i=1}^{n} d_i}{m} \rceil$ then
13:        Exit the for loop.
14:    end if
15: end for
```

Theorem 3.3. *The Multi-Run Heuristic returns a 2-approximate solution to JSP.*

Proof. In MRH, we apply the *Single Run Algorithm* for all $r \in \{\lceil \frac{n}{m} \rceil, \ldots, n\}$. We claim that setting $r = \lceil \frac{2n}{m} \rceil$ returns a 2-approximate solution. The total cost has two components, namely, setup cost and total length of the runs. We prove that when $r = \lceil \frac{2n}{m} \rceil$ both the total setup cost and the total length of the runs are less than or equal to two times the total setup cost and the total length of of the runs in the optimal solution, respectively.

The minimum number of runs in an optimal solution is $\lceil \frac{n}{m} \rceil$. Since $\lceil \frac{2n}{m} \rceil \leq 2\lceil \frac{n}{m} \rceil$, the total setup cost of the solution found by MRH (for $r = \lceil \frac{2n}{m} \rceil$) is less than or equal to two times that of the optimal solution.

The total length of the runs, $\sum_{j=1}^{n} L_j$, is at least $\lceil \frac{\sum_{i \in N} d_i}{m} \rceil$. $\sum_{j=1}^{n} L_j \geq \lceil \frac{\sum_{i \in N} d_i}{m} \rceil$ implies that total quantity produced, $m \sum_{j=1}^{n} L_j$, is greater than or equal to $\sum_{i \in N} d_i$. We next prove that the total quantity produced in the solution found by MRH (for $r = \lceil \frac{2n}{m} \rceil$) is not greater than $2 \sum_{i \in N} d_i$. This completes the proof.

In MRH, after applying the *Single Run Algorithm*, we sort the slots in a nonincreasing order of quantity of items assigned to them. Let s_1 be the quantity assigned to the first slot. The total length of the runs will be less than or equal to rs_1 since the length of the first run is s_1 and the runs are ordered in a nonincreasing order of their lengths. The total quantity produced in this solution is less than or equal to mrs_1. We show that mrs_1 is less than or equal to $2\sum_{i=1}^{n} d_i$.

Let N_1 be the set of the types that are assigned only one slot after applying MRH and let N_2 be the set of remaining types. Let $|N_1| = n_1$ and $|N_2| = n_2$. Then, $n = n_1 + n_2$. Let y_i be the number of slots assigned to type i for $i \in N_2$. Then, $s_1 \geq \lceil \frac{d_i}{y_i} \rceil$ and $s_1 \leq \lceil \frac{d_i}{y_i - 1} \rceil$ for all $i \in N_2$. Suppose that we assign one less slot to the types in N_2. There are two cases that have to be considered:

1. $s_1 < \lceil \frac{d_i}{y_i - 1} \rceil$: In this case, the quantity assigned to all $y_i - 1$ slots is at least s_1.
2. $s_1 = \lceil \frac{d_i}{y_i - 1} \rceil$: Quantity assigned to some of these $y_i - 1$ slots (at most $y_i - 2$ of them) can be $s_1 - 1$.

As a result, we have n_2 slots with at least an assigned quantity of s_1. Consider the remaining $y_i - 2$ slots assigned to type i, which are possibly assigned $s_1 - 1$ quantity. There are $mr - 2n_2 - n_1$ such slots. Now, consider the types that are assigned only one slot and add the quantity assigned to these slots to n_1 of $mr - 2n_2 - n_1$ slots. Then, we have $n (= n_1 + n_2)$ slots with an assigned quantity of at least s_1 and $mr - 2n$ slots with an assigned quantity of at least $s_1 - 1$. We know that $(mr - 2n)(s_1 - 1) + ns_1 \leq \sum_{i \in N} d_i$ and $mr \geq 2n$. Then, $\sum_{i \in N} d_i \geq (mr - 2n)(s_1 - 1) + ns_1 \geq \frac{mrs_1}{2}$ for $s_1 \geq 2$. This concludes the proof except for the special case where $s_1 = 1$.

Suppose that $s_1 = 1$. There are two cases:

1. $\sum_{i \in N} d_i > m\lceil \frac{n}{m} \rceil$: In this case, MRH applied to $r = \lceil \frac{2n}{m} \rceil$ gives a 2-approximate solution since more than half of the slots are assigned only 1 unit. (There are

$m\lceil\frac{2n}{m}\rceil$ slots and the total demand is greater than $m\lceil\frac{n}{m}\rceil$ which means more than half of the slots are assigned 1 unit.)

2. $\sum_{i\in N} d_i \le m\lceil\frac{n}{m}\rceil$: The problem is trivial. Let $n = km + n'$ where $0 < n' \le m$. Then, $\sum_{i\in N} d_i \le (k+1)m$. This means we can solve the problem optimally in $k+1$ runs with run length 1 for each run. Moreover, applying MRH for $r = k+1$ also returns this solution.

□

While applying the two heuristics (MRH and BCH) to JSP, our approach is to use both of the heuristics and their variations and take the best solution among them. Thus, our approach also gives a 2-approximate solution to JSP.

Although we prove that MRH is a 2-approximate algorithm, the worst-case example we found so far has a 1.5-approximate solution.

Example 3.2. *Consider a machine with $m(\gg 0)$ slots and m types with demands $d_1 = \ldots = d_{m-2} = \epsilon$, $d_{m-1} = mM + \epsilon$ and $d_m = 2mM + \epsilon$ where $M \gg \epsilon$. Let $c = \frac{3M}{4}$. The optimal solution is with two runs and with lengths $3M$ and ϵ. In the first run, one third of the slots are assigned to type $m-1$ and the remaining slots are assigned to type m and its length is $3M$. For the second run, the type-slot assignment is [1 2 $\ldots$ m] and its length is ϵ. The total cost is equal to $\frac{9M}{2} + \epsilon$. However, the best solution found by MRH has 3 runs with lengths $\frac{6mM+3\epsilon}{4m+3}$, $\frac{6mM+3\epsilon}{4m+3}$ and $\frac{3mM+3\epsilon}{2m+3}$. The type-slot assignments for these 3 runs are [$m \ldots m$], [$m \ldots m$ $m-1 \ldots m-1$] and [1 2 $\ldots$ $m-1$ $m-1$]. In the second run, $\frac{m}{3}+1$ of the slots are assigned to type m and the remaining are assigned to type $m-1$. Total cost of this solution is equal to $\frac{9M}{4} + 2(\frac{6mM+3\epsilon}{4m+3}) + \frac{3mM+3\epsilon}{2m+3} \approx \frac{27M}{4}$ which is a 1.5-approximation to the optimal solution for big m values.*

In the above example, the variability in the demand is high which forces MRH to do more runs in order to create balanced type-slot assignments that decrease the waste. On the other hand, the high setup cost prevents MRH from doing more runs. The variability in the demand and the high setup cost negatively impact the performance of MRH.

3.5.2 Balanced Cost Heuristic (BCH)

This heuristic is inspired by a lot-sizing heuristic, namely the *Part-Period Balancing* proposed by DeMatteis and Mendoza [23], which tries to balance the order cost and the inventory holding cost in a production setting. Using a similar idea, in the *Balanced Cost Heuristic* (BCH), we try to find a solution that balances the setup cost and the waste cost.

First, we give an overall idea about BCH and then present the pseudocode. In BCH, we construct only one run at each iteration of the algorithm. After constructing the run, we update the remaining demand and continue to the next iteration until all the demand is satisfied. Suppose that there are more than m types in JSP. We sort them in a nonincreasing order according to their demands. Without loss of generality, we assume that $d_1 \geq \ldots \geq d_n$. We assign a single slot to the types 1, 2, $\ldots, m-1$ and m. We consider the demand of these m types as a potential length for the current run considered in BCH. Starting from $i = m$, we calculate the waste cost as $\frac{1}{m}\sum_{j=i}^{m}(d_i - d_m)$ if the length of the run is set to d_i for all $i \in \{1, \ldots, m\}$. For each $i \in \{1, \ldots, m\}$, we compare this cost to the setup cost c to decide whether to set the run length to this value or not. The pseudocode for BCH is presented in Algorithm 3.

After the termination of the algorithm, the *totalcost* is the objective function value of the solution. This is the basic version of the *Balanced Cost Heuristic* (BCH). Since the running time of the algorithm for a single c value is less than a second, we propose the following variants which may lead to better solutions:

(BCH-v1): Run the algorithm for different c values and choose the one which gives the best solution in terms of the original c value.

(BCH-random): When comparing the waste cost and the setup cost, we generate a random number $z \in [0, 1]$ using the uniform distribution. If z is less than the ratio of the waste cost to the total cost (waste cost + setup cost), we fix this run. Otherwise, we continue the for loop until the generated z value is less than the ratio of the waste cost to the total cost. Apply the algorithm several times (with a new random seed, leading to different z values) for the original c value or for a different c value and choose the solution which has the minimum total cost in terms of the original c value.

Similar to MRH, we also propose variations to BCH, which are presented in Appendix B. The variations of BCH are also valid for BCH-v1 and BCH-random. In the computational experiments, we apply all these different strategies and take the best solution among the ones found.

3.6 Preprocessing Ideas for Strengthening the Integer Formulation

In this section, using the solutions found by the heuristics and some other observations, we further strengthen the integer formulation (IP2′). Particularly, we try to (i) improve the upper bound on the number of runs, (ii) tighten the upper and lower bounds on the length of the runs, and (iii) limit the number of slots that can be assigned to a certain type in a given run.

In (IP2′), the number of types, the number of runs and the number of slots that can be assigned to a type determine the number of variables and constraints. Thus, if we reduce the upper bound on the number of runs (currently n) and the number of slots that can be assigned to a type in a run (currently m for all the types), the size of the linear integer program reduces. Hence, we focus on obtaining better upper bounds on the number of runs and the number of slots that can be assigned to a type. The resulting stronger formulation allows us to solve several of the real-world

Algorithm 3 *Balanced Cost Heuristic.*

```
1: Set $totalcost = 0$.
2: if $n \geq m$ then
3:     Sort the types according to their demand in a nonincreasing order, $d_1 \geq d_2 \geq \ldots \geq d_n$.
4:     Choose the first $m$ types $(d_1, \ldots, d_m)$ and assign one slot to each without any splitting. That is, the quantity assigned to slot $i$ is equal to $d_i$.
5: else
6:     Apply the Single Run Algorithm.
7:     Determine the quantities assigned to each slot and sort the slots in a nonincreasing order according to the quantities assigned to them.
8: end if
9: Let $s_k$ denote quantity assigned to slot $k$ for $k \in \{1, \ldots, m\}$.
10: for $i = m$ to 1 do
11:     Calculate waste cost as $\frac{1}{m}\sum_{j=1}^{m}(s_i - \min(s_j, s_i))$.
12:     if $\frac{1}{m}\sum_{j=1}^{m}(s_i - \min(s_j, s_i)) > c$ then
13:         Exit the for loop and set the length of this run to $s_{i+1}$. The assignment of the slots to types is same as it is done at the beginning.
14:     end if
15: end for
16: Fix this run and update the demand for each type by subtracting the quantity that is produced in this run.
17: Update the totalcost by adding $c + s_{i+1}$. If all the demand is satisfied, terminate the algorithm. Otherwise, go to Step 2.
```

instances optimally.

3.6.1 A Better Upper Bound on the Number of Runs

In the formulations provided in Section 3.4.2, the number of runs is a variable, which we vary between 1 and n. Recall from Section 3.2 that n is an upper bound and $\lceil \frac{n}{m} \rceil$ is a lower bound on the optimal number of runs. Unfortunately, in practice n turns out to be a very loose upper bound. Luckily, the best solution found by the heuristics provides us with a much tighter upper bound on the optimal number of runs, even an exact value in some cases.

Observation 3.3. *Let $BEST$ denote the best solution found by the heuristics. Then,*

$$\left\lfloor \frac{BEST - \lceil \frac{\sum_{i \in N} d_i}{m} \rceil}{c} \right\rfloor \tag{45}$$

is an upper bound on the number of runs in the optimal solution.

We know that the total length of the runs in a feasible solution is at least $\lceil \frac{\sum_{i \in N} d_i}{m} \rceil$. Thus, if the total setup cost of a feasible solution is greater than $BEST - \lceil \frac{\sum_{i \in N} d_i}{m} \rceil$, the total cost of this solution is greater than $BEST$. Furthermore, since the unit setup cost is c, (45) gives an upper bound on the number of runs.

Using a tighter upper bound on the number of runs can significantly reduce the size of the formulation. For example, in one of the real-world instances where $n = 54$, $m = 20$ and $c = 1300$, after applying the heuristics and using Observation 3.3, we find that the upper bound on the number of runs is only 3 (instead of 54). In this instance, it turns out that the lower bound is also 3, hence, we conclude that the optimal solution has three runs.

Let $LB (= \lceil \frac{n}{m} \rceil)$ be the lower bound and UB be the upper bound on the number of runs. By solving the problem for a given number of runs, say r, for all $r \in \{LB, \ldots, UB\}$, we can find the optimal solution. For the rest of the discussion, we assume that we have fixed the number of runs to some $r \in \{LB, \dots, UB\}$ and the runs are ordered in a nonincreasing order of their lengths, $L_1 \geq \dots \geq L_r$. Then, by solving (IP2$'$) with a fixed number of r runs we try to find a better solution than the one found by the heuristics in r runs. In order to do this, we only have to check if the optimal objective value of this problem, where the number of runs is set to r, is smaller than $BEST$. This problem can be infeasible for some r values where $r \in \{LB, \dots, UB\}$, which implies that for those r values there is no better solution.

3.6.2 Setting Upper and Lower Bounds on the Lengths of the Runs

In this section, we try to improve the lower and upper bounds on the lengths of the runs which are 0 and $d_{max}(:= \max_{i \in N}\{d_i\})$ currently.

We start with some notation. Let L_j^{LB} and L_j^{UB} be the lower bound and upper bound on the length of run j. Initially, each run has a minimum length of 1, that is, $L_j^{LB} = 1$ for all $j \leq r$. Without loss of generality, we assume that $d_1 \geq \dots \geq d_n$. We first state some observations which form the building blocks of the preprocessing steps.

Observation 3.4. *If there is a better solution than the one obtained by the heuristics, the total length of the runs in this solution is less than or equal to* $\lfloor BEST - cr \rfloor$.

Let L_{total}^{UB} be the upper bound on the total length of the runs. Then, $L_{total}^{UB} = \lfloor BEST - cr \rfloor$.

Observation 3.5. $L_j^{UB} = \lfloor \frac{L_{total}^{UB}}{j} \rfloor$ *for all* $j \leq r$.

Since the runs are ordered in a nonincreasing order of their lengths, $L_1 \geq \dots \geq L_j$ for any $j \leq r$. This implies that $L_{total}^{UB} \geq \sum_{i=1}^{r} L_i \geq \sum_{i=1}^{j} L_i \geq \sum_{i=1}^{j} L_j$, which proves the observation.

Observation 3.6. *For* $j \leq \lceil \frac{n}{m} \rceil$, L_j *should satisfy the following inequality:*

$$\sum_{i=n-m(j-1)+1}^{n} \lceil \frac{d_i}{L_j} \rceil \leq m(r-j+1). \qquad (46)$$

If a type, say type i, is not produced in any of the first $j-1$ runs, the number of slots that are assigned to that type is at least $\lceil \frac{d_i}{L_j} \rceil$ since the lengths of the runs after the j^{th} run are not greater than that of run j. Until the j^{th} run, demand of at most $m(j-1)$ types can be satisfied. There are still at least $n-m(j-1)$ types that are not produced in any of the first $j-1$ runs. The total number of slots that have to be assigned to these types is at least $\sum_{i=n-m(j-1)+1}^{n} \lceil \frac{d_i}{L_j} \rceil$ since the demand of the types are in a nonincreasing order, $d_1 \geq \ldots \geq d_n$. Before we initiate the j^{th} run, we have $m(r-j+1)$ slots left in total. Then, Observation 3.6 follows. The minimum of such L_j values gives a lower bound on the length of j^{th} run, and it can be found by a line search algorithm on line segment $[1, L_j^{UB}]$.

Since L_j^{LB}'s are the lower bounds on the length of the runs, $ExtraLength$, defined as $ExtraLength := L_{total}^{UB} - \sum_{j=1}^{r} L_j^{LB}$, is the flexible portion of the total length that can be assigned to any run.

Observation 3.7. *The upper bounds on the run lengths are updated using the following subroutine in Algorithm 4.*

Algorithm 4 Subroutine for updating the upper bounds on the run lengths.

```
 1: $L_1^{UB} = \min\{L_1^{LB} + ExtraLength, L_1^{UB}\}$.
 2: for $j = 2$ to $r$ do
 3:     $sum = ExtraLength$.
 4:     for $h = j$ to 1 do
 5:         $sum = sum + L_h^{LB}$.
 6:         if $\lfloor \frac{sum}{j-h+1} \rfloor \leq L_{h-1}^{LB}$ then
 7:             $L_j^{UB} = \min\{\lfloor \frac{sum}{j-h+1} \rfloor, L_j^{UB}\}$.
 8:             Exit the inner for loop.
 9:         end if
10:     end for
11: end for
```

The following discussion provides an overall idea about the subroutine. The maximum length of run j is $L_j^{LB} + ExtraLength$. However, we can obtain a tighter upper bound on the length of run j since the runs are in a nonincreasing order of their lengths. We know that the length of run k for $k < j$ has to be greater than or equal to that of run j. Thus, if $L_{j-1}^{LB} < L_j^{LB} + ExtraLength$, then we can calculate a tighter upper bound for run j as $\lfloor \frac{L_{j-1}^{LB}+L_j^{LB}+ExtraLength}{2} \rfloor$.

3.6.3 Setting Upper Bounds on the Number of Slots Assigned to a Type

In the current formulation, the upper bound on the number of slots that can be assigned to a type in a run is m and we define $m + 1$ slot variables (x_{ijk}) for each type i for each run j. In this section, we improve this upper bound for each type. Improvement of this upper bound will decrease the number of variables and thus size of the problem.

Let max_{ij} be the maximum number of slots that can be assigned to type i in run j. Similar to $ExtraLength$ defined previously, let $ExtraSlots$ be the number of slots that we are flexible in assigning to the types and $Slots_i$ be the minimum number of slots that have to be assigned to type i. The following observation summarizes the preprocessing idea.

Observation 3.8. *$Slots_i = \lceil \frac{d_i}{L_1^{UB}} \rceil$ and $ExtraSlots = mr - \sum_{i \in N} Slots_i$. Then,*

$$max_{ij} = \min\{ExtraSlots + Slots_i, \lceil \frac{d_i}{L_j^{LB}} \rceil, m\}. \tag{47}$$

Since the runs are in a nonincreasing order of their lengths, the minimum number of slots that have to be assigned to type i is $\lceil \frac{d_i}{L_1^{UB}} \rceil$. Then, $mr - \sum_{i \in N} Slots_i$ gives the number of slots that we are flexible in assigning to the types. Observation 3.8 follows since it is not reasonable to assign more than $\lceil \frac{d_i}{L_j^{LB}} \rceil$ slots to type i in run j.

To summarize, lower bounds on the run lengths are updated using Observation 3.6. Upper bounds are calculated using Observations 3.5 and 3.7. The maximum number of slots that can be assigned to a type in a certain run is calculated using Observation 3.8.

Using the above information, we strengthen the formulation (IP2′) by adding valid cuts on the run lengths and possibly reducing the number of variables. The strengthened formulation after these preprocessing steps is:

$$
\begin{array}{llll}
\text{IP3: Minimize} & cr + \sum_{j=1}^{r} L_j & & (48)\\
\text{subject to} & \sum_{j=1}^{r}\sum_{k=0}^{max_{ij}} q_{ijk} = d_i & i \in N, & (49)\\
& \sum_{i\in N}\sum_{k=0}^{max_{ij}} kx_{ijk} \le m & j \in \{1,\dots,r\}, & (50)\\
& q_{ijk} \le kL_j & i \in N,\ j \in \{1,\dots,r\},\ k \in \{0,1,\dots,max_{ij}\}, & (51)\\
& q_{ijk} \le d_i x_{ijk} & i \in N,\ j \in \{1,\dots,r\},\ k \in \{0,1,\dots,max_{ij}\}, & (52)\\
& \sum_{k=0}^{max_{ij}} x_{ijk} = 1 & i \in N,\ j \in \{1,\dots,r\}, & (53)\\
& L_j \ge L_{j+1} & j \in \{1,\dots,r-1\}, & (54)\\
& \sum_{i\in N}\sum_{k=1}^{max_{ij}} q_{ijk} \le mL_j & j \in \{1,\dots,r\}, & (55)\\
& \sum_{j=1}^{r} L_j \le L^{UB}_{total} & & (56)\\
& L_j \ge L^{LB}_j & j \in \{1,\dots,r\}, & (57)\\
& L_j \le L^{UB}_j & j \in \{1,\dots,r\}, & (58)\\
& x_{ijk} \in \{0,1\} & i \in N,\ j \in \{1,\dots,r\},\ k \in \{0,1,\dots,max_{ij}\}, & (59)\\
& q_{ijk} \ge 0 & i \in N,\ j \in \{1,\dots,r\},\ k \in \{0,1,\dots,max_{ij}\}, & (60)\\
& L_j \ge 0 & j \in \{1,\dots,r\}. & (61)
\end{array}
$$

Constraints (56)-(58) are the new constraints that are added after the preprocessing steps. In addition, the set where the slot index k is chosen from is updated according to the new max_{ij} values.

3.7 Computational Results

In this section, we test the performance of MRH and BCH on three sets of problem instances. We also conduct experiments to assess the impact of the preprocessing steps presented in Section 3.6.

3.7.1 Evaluating the Performance of the Heuristics

We test the performance of our proposed heuristics in the following three sets of problem instances:

1. **T**: The instances solved by Teghem et al. [102], where $m = 4$ (i.e., there are 4 slots), $c_s = 18,676$ and $c_w = 3.36$. The demand structure in these 4 instances is as follows:

   - T1 : $n = 3$, $d_1 = 16,000$, $d_2 = 9,000$, $d_3 = 4,500$,
   - T2 : $n = 4$, $d_1 = 20,000$, $d_2 = 18,000$, $d_3 = 15,000$, $d_4 = 8,500$,
   - T3 : $n = 5$, $d_1 = 13,500$, $d_2 = 114,500$, $d_3 = 103,500$, $d_4 = 94,500$, $d_5 = 84,500$,
   - T4 : $n = 8$, $d_1 = 15,000$, $d_2 = 12,000$, $d_3 = 10,000$, $d_4 = 8,000$, $d_5 = 5,000$, $d_6 = d_7 = d_8 = 3,000$.

2. **RW**: Real-world instances provided to us by one of the 75 largest printing companies in North America.

3. **Random**: Randomly generated instances.

In the following discussion and tables, "Best" represents the best result obtained by MRH and BCH, "Optimal" represents the optimal objective value found by solving the integer programming formulation.

In Table 8, we compare the results on the set of instances in Teghem et al. [102], where the best solution found by the authors is under the column "Teghem et al. [102]". Columns "Best/Teghem et al. [102]" and "Best/Optimal" present the ratio of best result found by the heuristics to the result found by Teghem et al. [102] and to the optimal objective value, respectively. It is seen in Table 8 that the combination of MRH and BCH performed 7.8% better on average when compared to the method proposed by Teghem et al. [102]. In addition, the solutions found by our approach are optimal in 3 of 4 instances. Teghem et al. [102] do not present detailed results on the solution times but state that they are between 10 and 100 minutes. However, our heuristics find "good" solutions in less than a second. Furthermore, since these instances are small, we find the optimal solutions in less than 5 seconds using the strengthened linear integer formulation.

Table 8: Comparison of the results with the ones in Teghem et al. [102]

Instance	Best	Teghem et al. [102]	Optimal	Best/Teghem et al. [102]	Best/Optimal
T1	138,152	136,472	136,472	1.0123	1.0123
T2	247,916	247,916	247,916	1.0000	1.0000
T3	1,447,068	1,855,872	1,447,068	0.7797	1.0000
T4	264,348	294,980	264,348	0.8962	1.0000
				0.9220	**1.0031**

Next, we test the performance of our heuristics on real-world instances. In Table 9, column "(m, n)" represents the structure of the instance, where m is the number of slots on the machine and n is the number of types (orders). Columns "MRH" and "BCH" are the best results obtained by applying MRH and BCH (including their variations), respectively. Column "LB" is the best lower bound obtained for that instance. Column "Best/LB" represents the ratio of the best result found to the lower bound.

For some of the instances, LB is the optimal solution, which we find either by showing that the optimal number of runs is 1 and applying the *Single Run Algorithm* or by applying the preprocessing ideas and then solving the linear integer formulation. 14 out of 32 instances are solved optimally. On these instances, Observation 3 helped in finding the optimal number of runs because of the relatively high setup cost. Finding the optimal number of runs reduced the problem size significantly, and we were able to solve these instances by using the linear integer formulation (IP3). For the other instances, LB is found by solving the relaxed version of the linear integer

Table 9: Experiments on the real-world instances

Instance	**(m,n)**	**MRH**	**BCH**	**Best**	**LB**	**Best/LB**
RW1	(72,85)	18,600	17,400	17,400	17,137	1.0153
RW2	(20,54)	10,500	10,600	10,500	10,500(Optimal)	1.0000
RW3	(20,54)	10,800	10,800	10,800	10,420	1.0365
RW4	(32,353)	51,900	51,200	51,200	48,161	1.0631
RW5	(56,29)	6,700	6,700	6,700	6,700(Optimal)	1.0000
RW6	(36,26)	3,200	3,200	3,200	3,200(Optimal)	1.0000
RW7	(48,45)	5,382	5,400	5,382	5,367(Optimal)	1.0028
RW8	(30,34)	6,029	6,029	6,029	6,029(Optimal)	1.0000
RW9	(42,218)	22,467	22,467	22,467	21,039	1.0679
RW10	(54,58)	4,350	4,600	4,350	4,258	1.0216
RW11	(25,100)	3,100	3,100	3,100	3,100(Optimal)	1.0000
RW12	(65,62)	20,700	20,100	20,100	18,570	1.0824
RW13	(50,19)	3,050	3,050	3,050	3,050(Optimal)	1.0000
RW14	(64,52)	3,300	3,300	3,300	3,300(Optimal)	1.0000
RW15	(72,46)	5,000	5,000	5,000	4,970	1.0060
RW16	(72,199)	41,892	40,000	40,000	37,232	1.0743
RW17	(90,203)	12,760	12,700	12,700	11,888	1.0683
RW18	(54,22)	2,300	2,300	2,300	2,300(Optimal)	1.0000
RW19	(20,51)	7,900	7,900	7,900	7,900(Optimal)	1.0000
RW20	(54,39)	4,625	4,600	4,600	4,600(Optimal)	1.0000
RW21	(48,186)	12,200	12,200	12,200	12,200(Optimal)	1.0000
RW22	(80,10)	3,800	3,800	3,800	3,800(Optimal)	1.0000
RW23	(24,730)	109,900	108,600	108,600	106,634	1.0184
RW24	(54,223)	20,500	20,800	20,500	19,093	1.0737
RW25	(72,198)	31,700	31,500	31,500	28,871	1.0911
RW26	(52,447)	26,267	26,600	26,267	24,524	1.0711
RW27	(25,48)	5,100	5,100	5,100	5,100(Optimal)	1.0000
RW28	(54,678)	77,667	76,200	76,200	72,919	1.0450
RW29	(30,994)	149,500	148,500	148,500	144,467	1.0279
RW30	(43,390)	130,900	127,900	127,900	123,070	1.0392
RW31	(109,25)	10,500	10,500	10,500	10,440	1.0057
RW32	(30,2086)	315,600	314,400	314,400	307,867	1.0212
						1.0261

formulation. In some of these instances, we were able to tighten the lower bound by narrowing the range for the number of runs in the optimal solution. For example, in an instance where the lower and upper bounds on the optimal number of runs are 3 and 6, respectively, we show that the best solution found in 3 runs is worse than the one found by the heuristics.

Table 9 shows that the average optimality gap of our results is only 2.6%. Note that since we compare the heuristic solution to a lower bound on the optimal solution, the results represent a conservative estimate of the heuristics' performance. In 13 out of the 14 instances for which we were able to determine the optimal solution, the heuristics also found the optimal solution and in the instance we were not able to find the optimal solution, the optimality gap was 0.28%.

In general, BCH performed slightly better than MRH. As we explain in Section 3.5, we have three different application strategies for BCH. The first one is the deterministic one where we apply the algorithm for different c values and take the best that gives the lowest cost in terms of the original c value. The other two are randomized strategies. All of these increase the number of trials we can do for BCH. In the computational tests, we have run BCH for 1,000 different c values for the deterministic strategy. In the first randomized strategy, we run the algorithm for 1,000 different random number streams using the original c value. For the second randomized strategy, we run the heuristic for 10,000 different random number streams with a different c value each time. Our intuition is that BCH performs slightly better than MRH since the number of trials increases the probability of finding a better solution.

In addition to the close-to-optimal performance of our heuristics, one of our main contributions is the reduced solution times. In the current practice, constructing a production plan takes hours, even days for large instances such as $(m, n) = (30, 2086)$. The company which provided us the real-world instances uses an enumeration type of algorithm which finds a "good" solution in hours. Since the number of alternatives considered in an enumeration type algorithm is exponential, the solution times are long. The heuristics we propose are easy to implement and generate high quality production plans in less than a minute even for large instances.

Next, we test the heuristics on randomly generated instances to analyze their performance on different problem settings. In JSP, the number of slots, the number of types, the demand structure and the (scaled) setup cost are the parameters that determine the setting of the instance. We generate random instances and conduct a factorial experiment to see the effect of these parameters on the performance of the heuristics.

Considering the real-world instances, we propose two different settings for the number of slots. In the "Low" setting, the number of slots is generated uniformly between 20 and 60 and for the "High" setting, it is generated uniformly between 80 and 120. In JSP, the ratio of number of types to the number of slots may affect the solution. The instances with same ratio of number of types to the number of slots with other parameters being same have similar solutions. Thus, the effect of the number of types on the performance of the heuristics is tested taking into account the $\frac{n}{m}$ ratio. We generate the number of types uniformly between 0 and $6m$ (where m is the number of slots generated) to represent the "Low" $\frac{n}{m}$ ratio and between $8m$ and $12m$ to represent the "High" $\frac{n}{m}$ ratio.

In addition to the number of types, the variability in the demand may affect the performance of the heuristics. To evaluate the effect of the demand variability, we propose two different settings while generating the random instances. We generate the demand for n types uniformly between 100 and 10,000 (as multiples of 100) for the "Varied" setting and between 5,000 and 6,000 (also as multiples of 100) for the "Less Varied" setting.

Finally, the last parameter that changes the structure of the solution is the (scaled) setup cost. To see how the heuristics behave under different setup costs, we consider three different settings for c: (i) $c = 100$ (Low), (ii) $c = 1,000$ (Comparable), and (iii) $c = 10,000$ (High). In $c = 10,000$ case, the setup cost is high, thus minimizing the number of setups becomes the first goal. When $c = 1,000$, the setup cost is still dominating but minimizing waste is also crucial. In the last case ($c = 100$), waste cost and setup cost are comparable but the minimizing waste dominates the setup cost considerations. For smaller values of c (such as $c = 10$ or 1), the focus is on minimizing the waste. In those cases, the solution of the problem is obvious and the heuristics find solutions with less than 0.5% optimality gap. Hence, we do not include these settings in the experiments. As a result, we have 24 different settings. We generate 10 instances for each of these settings.

The results are presented in Table 10. The columns "Slots", "Types/Slots", "Demand" and "Setup Cost" represent the number of slots, number of types, demand and setup cost setting for the randomly generated instances as explained above. The row "Avg. Gap" represents the average optimality gap of the solutions (obtained by the heuristics) of all randomly generated instances for which the corresponding parameter is set to the value mentioned in the column title. Similarly, the row "Std. Dev." represents the standard deviation of the optimality gap of all these instances.

Table 10: Experiments on the randomly generated instances

	Slots		**Types/Slots**		**Demand**		**Setup Cost**		
	Low	**High**	**Low**	**High**	**Varied**	**Less Varied**	**Low**	**Comparable**	**High**
Avg. Gap	2.274%	2.330%	2.847%	1.758%	3.831%	0.774%	1.730%	3.821%	1.356%
Std. Dev.	2.361%	2.504%	2.712%	1.974%	2.485%	0.982%	1.063%	3.250%	1.605%

As seen in Table 10, the number of slots does not affect the solution quality. However, when $\frac{n}{m}$ ratio decreases, the quality of the solutions found deteriorate. This is because when $\frac{n}{m}$ ratio increases, forming solutions with less waste is more likely. In addition, the variability in the demand results in bad performance of the heuristics which is quite intuitive. As the setup cost and the waste cost become comparable, the heuristics seem to perform slightly worse. The average gap is higher when the setup cost is comparable to the waste cost. It seems that the instances where the setup cost and waste cost are comparable are challenging because of the difficulty in determining the trade-off between setup and waste costs. The heuristics perform worse not only because these are challenging instances but also we are not able to solve these instances to optimality. We can only improve the lower bound on the number of runs in some of these randomly generated instances.

As the result of the experiments suggest, the setting where $\frac{n}{m}$ ratio is low, the demand is varied and the setup cost is comparable to the waste cost is the most challenging one. The average optimality gap of the solutions found by the heuristics under this setting is 7.52%. In addition, we analyze the performance of the heuristics on randomly generated instances where the demand structure is a mixture of “Varied” and “Less Varied” setting. That is, a subset of the types have a “Less Varied” demand structure, and the remaining types have a “Varied” structure. The average optimality gap of the solutions found by the heuristics under this setting is lower (7.06%) when compared to the most challenging setting described above. This is also a conservative estimate since we use lower bound on the optimal solution to calculate the optimality gaps.

3.7.2 Evaluating the Performance of the Preprocessing Steps

In order to solve the problem optimally, we proposed several preprocessing ideas in Section 3.6. They are based on: (i) improving the upper bound on the number of runs, (ii) setting upper and lower bounds on the run lengths, and (iii) finding the maximum number of slots that can be assigned to a certain type in a certain run which, possibly, decreases the number of variables. To measure the effect of these preprocessing ideas, we conducted experiments on the real-world and randomly generated instances that we were able to solve optimally.

The preprocessing ideas are most effective when the setup cost dominates the waste cost. As it is seen in Table 11, there are 112 such instances out of 272. In the other instances, the preprocessing ideas help in improving the lower bound on the optimal objective value. Without improving the upper bound on the number of runs, we cannot even find a feasible solution to the LP relaxation in most of these 112 instances after several hours. Out of the 112 instances, we were able to solve 42 instances optimally. 28 out of these 42 instances are solved after improving the upper bound on the number of runs. In these 28 instances, the optimal solution is easily computed since either it is a *Single Run Problem* that can be solved in polynomial time using the *Single Run Algorithm* or each type can be assigned to only one slot, which makes the problem trivial. Thus, we have not included these in the following experiments. However, the above results show that improving the upper bound on the number of runs is effective.

Table 11: Effect of preprocessing ideas

Total number of instances (random + real-world)	272
Number of instances with dominating setup cost	112
Number of instances solved optimally after preprocessing step (i)	28
Number of instances solved optimally after preprocessing steps (i)-(ii)	40
Number of instances solved optimally after preprocessing steps (i)-(ii)-(iii)	42

As a result, there are 14 instances that we consider in testing the effect of the other two preprocessing ideas. After setting upper and lower bounds on the run lengths (using Observations 3.4 - 3.7), we are able to solve 12 of these instances. Utilizing Observation 3.8 in Section 3.6.3, we are able to further improve all the solution times and solve the remaining two instances as well. Adding the result of Observation 3.8 resulted in a factor of 38 improvement in solution times on average except one instance. Utilizing Observation 3.8 reduced the solution time of this instance by a factor of 1,000 which is an outlier. This shows that both preprocessing ideas (setting upper and lower bounds on the run lengths and restricting the number of slots that can be assigned to a certain type in a certain run) strengthen the linear integer formulation. The preprocessing ideas work for all r values but they performed better when the number of extra slots, $mr - n$, is small. This corresponds to the instances where minimizing the number of setups has a higher priority.

In addition, we proposed a different branching scheme other than the usual single variable branching, which is efficient for the set of constraints such as $\sum_{k=0}^{m} x_{ijk} = 1$. For the instances we manage to solve optimally, we test how this branching scheme improves the solution time. Using this branching scheme reduced the solution times by a factor of 6 on average. Furthermore, the proposed branching scheme enabled us to solve four more instances optimally that the usual single variable branching rule was not able to solve.

3.8 Contributions and Extensions

In this chapter, we introduced a scheduling problem motivated by the printing industry. In current practice, forming a "good" production plan takes hours even days. We developed two heuristics which are easy to implement and find comparably "good" solutions in seconds. Furthermore, although we show that the problem is strongly NP-hard, we were able to obtain good lower bounds by developing a strong linear integer formulation that can utilize special branching rules and strengthen it by adding cuts. Finally, we developed several preprocessing procedures based on the results of the heuristics, which helped in solving 14 of the 32 real world instances to optimality.

The research discussed in this chapter can be extended to problems with two or more machines with different number of slots. Under such a setting, one can assume different setup costs on each machine and try to find a solution with minimum total cost. Alternatively, assuming different setup times, one can minimize the maximum completion time, which will require identifying a balanced assignment of the types to the machines.

CHAPTER IV

1.5-DIMENSIONAL RECTANGLE PACKING PROBLEM AND APPLICATIONS IN THE SEMICONDUCTOR INDUSTRY

4.1 Introduction

Very Large Scale Integration (VLSI) layout has been a fertile ground for new and interesting problems [94, 98]. A challenging subproblem is *routing*: connecting terminals of each net in the circuit such that the electrical current can flow from the voltage/current sources to their destinations with as little path delay as possible. Another concern for routing is to reduce the resource consumed in each such *route* to reduce the chip *congestion* and the total area of the chip. Total area minimization has several benefits including less spending in (i) raw material and disposal costs and (ii) production costs. The material used to produce chips has production defects which may result in a defective product if used in the production. The minimization of the area used decreases the size of the material used, and thus, the probability of producing a defective chip.

In this chapter, we introduce and study the *1.5-Dimensional Rectangle Packing Problem* (RPP) which has been used widely as the basis for the channel routing problem, with its special case single-sided channel routing, which is a subproblem of the general VLSI routing problem [101]. In RPP, there are n rectangles with prespecified horizontal locations. Let l_j and r_j denote the positions of the left and right edges, respectively, and h_j denote the height of rectangle j, $j = 1, \ldots, n$. The width of rectangle j is denoted by $w_j = r_j - l_j$. Given their fixed horizontal positions, the objective is to find a non-overlapping placement (or packing) of the rectangles

Table 12: Notation for RPP

$R = \{1, \dots, n\}$	:	Set of rectangles
l_j	:	Horizontal position of the left edge of rectangle j, $j \in R$
r_j	:	Horizontal position of the right edge of rectangle j, $j \in R$
h_j	:	Height of rectangle j, $j \in R$
$w_j \ (= r_j - l_j)$	:	Width of rectangle j, $j \in R$
Y	:	Total height of the placement
$I \ (\subset R \times R)$	:	Set of pairs of rectangles that intersect horizontally $(I = \{(i, j) \mid i, j \in R,\ i < j,\ l_j < r_i \leq r_j \text{ or } l_i < r_j \leq r_i\})$

so that the total height is minimized. More formally, in a given placement of the rectangles, let y_j denote the position of the top edge of rectangle j along the y-axis. Our goal is to create a placement of the rectangles so that $\max_j y_j$ is minimized. Figure 15 shows an instance of RPP and a feasible placement for the rectangles. The notation that we use throughout the paper is summarized in Table 12.

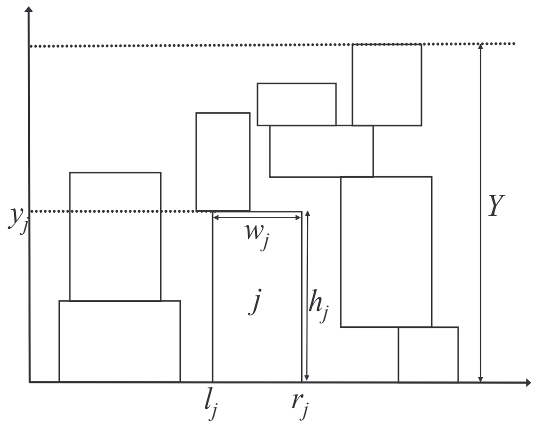


Figure 15: An example of RPP

The most common variant of the single-sided channel routing problem can be shown to be equivalent to a special case of RPP, where all the wires are of uniform width, i.e., rectangles of identical height. This problem has been treated extensively in both scientific literature and industrial use and is solved fairly successfully using the *Left Edge Algorithm* (LEA) [52]. However, the "uniformity" simplification turns into a severe limiting factor as circuit performance increases and chip complexity scales up with every new generation of processing technologies driven by the Moore's Law. For example, various wires need to be increased in width to reduce resistance and meet path timing constraints which determine how fast a chip's clock can switch. Further, in order to address the increasing complexity of the problem where n is typically very large in VLSI chips, bundling has been employed by microprocessor design companies like Intel, IBM and Motorola. Bundling refers to combining similar wires together into a *bus*. In a bus, many wires are placed together as a single but very tall rectangle. When 32-, 64-, or 128-bit buses are routed together, the complexity of the problem can be reduced by up to two orders of magnitude. This is not only a huge saving in runtime, but it also helps increase the level of abstraction and helps the chip designer make faster and better decisions to plan and then route the chip. On the other hand, the bundling of wires into buses causes the rectangles in RPP to be non-uniform which turns the problem to a hard one.

A problem related to RPP is the *Two-Dimensional Strip Packing Problem* (2SPP) introduced by Baker et al. [5]. In 2SPP, there is a strip of width W and a set $S = \{1, \ldots, n\}$ of rectangles with specified width (w_j) and height (h_j). The objective is to find a non-overlapping placement of the rectangles to the strip so that the height/length of the placement is minimized. In general, the rectangles can be placed arbitrarily to the strip, but in the literature the focus is on *orthogonal* placements such that the sides of the rectangles are parallel to the strip edges. Dowsland and Dowsland [27] and Lodi et al. [75] provide an extensive survey of various two-dimensional packing problems including 2SPP.

2SPP is motivated by the applications in several industries including wood, glass, textile, steel, plastic, paper and canvas [8, 10, 30, 58, 68, 75, 114]. In these applications, a set of rectangular items are cut from a single strip of material. Finding a minimum height placement is equal to minimizing the *trim loss* which reduces the waste or equivalently increases material utilization.

2SPP is NP-hard in the strong sense since it is a generalization of the well-known one-dimensional bin packing problem which is strongly NP-hard [40]. Several approximation algorithms are proposed for 2SPP [4, 5, 19, 96]. Most of the approximation algorithms place the rectangles into levels. Starting with the bottom of the strip, the subsequent levels are determined by the tallest rectangle placed in the current level. The approximation algorithm with the best absolute performance ratio is proposed by Steinberg [99] and has a worst-case ratio of 2. Recently, Kenyon and Remila [68] propose an asymptotic fully polynomial approximation scheme for 2SPP based on a new linear programming relaxation. In addition to approximation algorithms, several meta-heuristic algorithms are proposed for 2SPP. Hopper and Turton [61] provides a review of the meta-heuristic algorithms. An empirical investigation of meta-heuristic and heuristic algorithms is given by Hopper and Turton [60].

Martello et al. [79] propose a new relaxation which produces better lower bounds, and they use this in a branch-and-bound algorithm to solve 2SPP optimally. Recently, Bekrar et al. [9] develop three exact algorithms, namely, a branch-and-bound method, a dichotomous algorithm and a branch-and-price method.

RPP is a variant of 2SPP where the rectangles have fixed horizontal positions. Thus, the algorithms that consider moving the rectangles along the x-axis do not work for RPP. We develop algorithms that take into account the horizontal position restriction. The remainder of the paper is organized as follows. In Section 4.2, we prove that RPP is strongly NP-hard and discuss a special case which is polynomially solvable. In addition, we propose a method for finding a lower bound to the problem. In Section 4.3, we present an integer programming formulation for RPP. In Section 4.4, we discuss the applicability of the readily available heuristics and propose two new heuristics. In Section 4.5, we present the results of the computational experiments conducted on randomly generated instances. Finally, we conclude with contributions and extensions.

4.2 Complexity, Lower Bound, and a Special Case

In this section, we show that RPP is strongly NP-hard. Then, we propose a method to find a lower bound on the optimal solution. Finally, we discuss a special case which can be solved in polynomial time.

Theorem 4.1. *RPP is strongly NP-hard.*

Proof. The proof follows by a reduction from the *3-Partition* (3P) problem. An instance of 3P is defined as follows:

Instance: A set of integers $A = \{a_1, \ldots, a_{3m}\}$ and a bound $b \in Z^+$ such that $b/4 \leq a_i \leq b/2$ and $\sum_j a_j = mb$.

Question: Can A be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that for $1 \leq i \leq m$, $\sum_{a_j \in A_i} a_j = b$?

Answering this question is NP-complete in the strong sense [40]. We prove that the decision version of RPP is strongly NP-complete by creating an instance of RPP with $7m-2$ rectangles from an instance of 3P. Rectangle j ($\in \{1, \ldots, 2m-2\}$) has $l_j = (j-1)b$, $h_j = b$, $r_j = (2m-1)b$ for odd j and $r_j = 2mb$ for even j. Rectangle $2m-1$ has $l_{2m-1} = (2m-2)b$, $r_{2m-1} = (2m+1)b$ and $h_{2m-1} = b$. Rectangle $2m$ has $l_{2m} = 2mb$, $r_{2m} = (2m+1)b$ and $h_{2m} = (2m-1)b$. Rectangle j ($= 2m+i$ for $i \in \{1, \ldots, 2m-2\}$) has $l_j = (i-1)b$, $r_j = ib$ and $h_j = (2m-i)b$. Finally, the remaining $3m$ rectangles have $l_j = (2m-1)b$, $r_j = 2mb$ and $h_j = a_i$ for $j = 4m-2+i$ for $i \in \{1, \ldots, 3m\}$.

We want to show that there exists a solution to this instance of RPP with total height $\leq 2mb$ if and only if there exists a solution to 3P. First note that the total height of any placement is at least $2mb$ because rectangles 1 and $2m+1$ (similarly, rectangles $2m-1$ and $2m$) intersect horizontally. We make the following observations about a solution with total height $2mb$ for this instance of RPP.

1. Rectangles 1 and $2m+1$ (similarly, rectangles $2m-1$ and $2m$) must be adjacent vertically. Hence, rectangle 1 must be either at the top, or at the bottom of the

placement. Without loss of generality, let us assume that rectangle 1 is at the bottom. Then, rectangle $2m-1$ must be at the top (see Figure 16(a)) since it intersects with rectangle 1 horizontally. Hence, there is only one possible placement for rectangles 1, $2m-1$, $2m$ and $2m+1$ in a placement with height $\leq 2mb$ (assuming rectangle 1 is at the bottom).

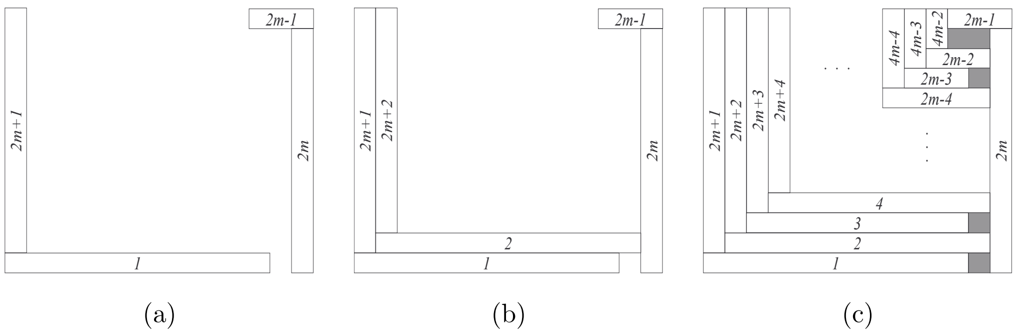


Figure 16: Reduction of RPP from 3P

2. Since rectangles 1, 2 and $2m+2$ intersect horizontally and have total height $2mb$, they must be placed adjacent to each other. Since rectangles 2 and $2m-1$ intersect, the only possible placement for rectangle 2 is just above rectangle 1, and rectangle $2m+2$ must be placed just above rectangle 2 (see Figure 16(b)).

3. By induction, let us assume that rectangles $1, \ldots, k-1$ and $2m+1, \ldots, 2m+k-1$, for some $k \leq 2m-2$ must be placed as in Figure 16(c) . Since rectangles $1, \ldots, k$ and $2m+k$ intersect horizontally and have total height $2mb$, they must be placed adjacent to each other. Since rectangles k and $2m-1$ intersect, the only possible placement for rectangle k is just above rectangle $k-1$, and rectangle $2m+k$ must be placed just above rectangle k. Hence, the placement shown in Figure 16(c) (or its vertical mirror image) is the only possible placement for rectangles $1, \ldots, 4m-2$ in a placement with total height $\leq 2mb$.

4. To keep the height of the placement at $2mb$, rectangles $4m-1, \ldots, 7m-2$ must

be placed into the m free areas (see shaded areas in Figure 16(c)) of height b each. But this is possible only if there is a solution to 3P.

From these observations, it follows that a solution with total height $\leq 2mb$ to this instance of RPP exists if and only if a solution to 3P exists. □

Next, we show that a lower bound can be calculated in polynomial time by solving the maximum weight clique problem on an interval graph. This lower bound is used to evaluate the performance of the heuristics (see Section 4.5). A graph $G = (V, E)$ is an interval graph if there exists a set $\{I_v \mid v \in V\}$ of real intervals such that $I_u \cap I_v \neq \emptyset$ if and only if $(u, v) \in E$ [26]. Interval graphs are *perfect* [26], i.e., $\chi(G) = \omega(G)$. The *chromatic number*, $\chi(G)$, is the minimum number of colors required to color the vertices of the graph such that any two adjacent vertices have different color. The *clique number*, $\omega(G)$, is the size of the largest clique in G. A clique C in G is a subgraph where all the vertices of C intersect pairwise.

From an instance of RPP, we create an interval graph $G^{RPP} = (V, E)$ as follows. For each rectangle j, we create a vertex $v_j (\in V)$ with weight h_j. If two rectangles, say i and j, intersect horizontally, we connect v_i and v_j by an edge $(v_i, v_j) \in E$. Any clique C in G^{RPP} corresponds to a set of rectangles which pairwise intersect horizontally. Hence, the total weight of the vertices in a (maximum weight) clique is a lower bound on the total height of a feasible (optimal) placement of the rectangles. Note that the maximum weight clique can be computed in polynomial time since G^{RPP} is an interval (and thus *perfect*) graph [48].

Proposition 4.1. *The weight of the maximum weight clique in $G^{RPP} = (V, E)$ is a lower bound to RPP.*

Next, we discuss a special case of RPP where all the rectangles have the same height, i.e., $h_j = h$ for all $j \in R$ for some $h \in \mathbb{R}^+$. Assume that we color the vertices of G^{RPP} using $\chi(G^{RPP})$ colors. Let V_i be the set of vertices colored with color i.

The vertices in V_i form an independent set, i.e., the rectangles corresponding to these vertices do not intersect horizontally. Hence, $\chi(G^{RPP})h$ is an upper bound on the optimal solution of the corresponding RPP. In Proposition 4.1, we showed that the weight of the maximum weight clique ($\chi(G^{RPP})h$) is a lower bound on the optimal solution. Thus, if all the rectangles in RPP have equal height (h), the total height of the optimal solution is $\chi(G^{RPP})h$. Since the coloring of the interval graphs can be done in polynomial time using the *Left Edge Algorithm* (LEA) [52], the optimal solution of RPP with equal heights can be found in polynomial time. The details of LEA is explained in Section 4.4.

Proposition 4.2. *If all the rectangles have equal height, say h, then RPP can be solved in polynomial time using LEA, and the height of the optimal solution is* $\chi(G^{RPP})h$.

4.3 Integer Programming Formulation

In this section, we present an integer programming formulation for RPP and propose ways to strengthen it. RPP can be formulated as an integer program using the following variables:

$$\begin{aligned}
&y_j = y - \text{coordinate of the top of rectangle } j && j \in R,\\
&Y = \text{total height of the placement,} &&\\
&x_{ij} = \begin{cases} 1, & \text{if rectangle } j \text{ is above rectangle } i \text{, i.e., } y_j > y_i \\ 0, & \text{otherwise} \end{cases} && (i,j) \in I.
\end{aligned}$$

The integer formulation is as follows:

$$\begin{array}{llll}
\text{IP: Minimize} & Y & & (62) \\
\text{subject to} & y_i + Mx_{ij} \geq y_j + h_i & (i,j) \in I & (63) \\
& y_j + M(1 - x_{ij}) \geq y_i + h_j & (i,j) \in I & (64) \\
& Y \geq y_j & j \in R & (65) \\
& y_j \geq h_j & j \in R & (66) \\
& x_{ij} \in \{0,1\} & (i,j) \in I & (67)
\end{array}$$

Constraints (63) and (64) compute the upper y-coordinates of the rectangles in the placement. The total height of the placement is calculated by constraints (65). Constraints (66) set the lower bounds on the y-coordinate of each rectangle. Finally, constraints (67) are the binary restrictions. In this formulation, M is a sufficiently large number, which can be chosen as $M = \sum_{j \in R} h_j$. Note that M can be improved by using the results of the heuristics.

For any feasible placement of the rectangles, there is an equivalent upside down placement. Figure 17 shows two placements of the rectangles, which are essentially the same.

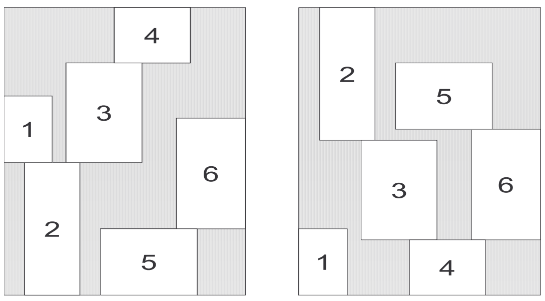


Figure 17: Equivalent placements

Integer programming formulation treats these two placements as different placements which results in unnecessary branching in the branch and bound tree. After the first branching in the branch and bound tree, there will be two branches which are essentially same (see Branch 1 and Branch 2 in Figure 18). To avoid this symmetry problem, we fix one of the x_{ij} (and the corresponding x_{ji}) variables to 1 (and 0). This can be done randomly, or after the LP relaxation at the root node is solved, the variable x_{ij} with $|x_{ij} - \frac{1}{2}|$ being the minimum can be selected.

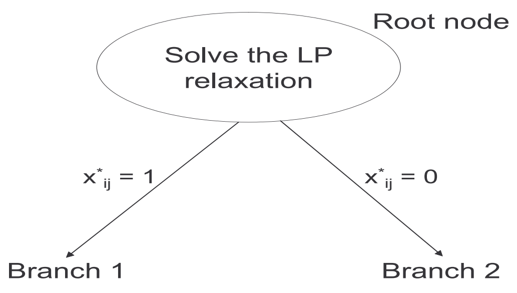


Figure 18: Branch and bound tree

Next, we propose a valid cut to strengthen (IP):

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \qquad \text{for any triple } (i,j,k) \text{ such that } (i,j),(j,k),(i,k) \in I. \quad (68)$$

Note that (depending on the value of M) $x_{ij} = x_{jk} = x_{ki} = \frac{3}{4}$ can be a feasible solution for (IP), but (68) eliminates this feasible solution since $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} \geq 2$.

4.4 Heuristics

In this section, we discuss LEA and the heuristics proposed for the 2SPP and their applicability to RPP. Finally, we propose two new heuristics for RPP.

4.4.1 Level Algorithms for 2SPP and Applicability to RPP

The most popular algorithms for 2SPP are the level algorithms where the rectangles are placed from left to right into rows forming levels [19, 76]. In each level, no rectangle is placed on top of another rectangle. Before placing the rectangles, they are sorted according to a nondecreasing order of their heights. The next rectangle in the list is either placed into an already created level or a new level is created for it. Assuming that P_l is the set of rectangles placed into level l, the height of the level, H_l, is determined by the height of the tallest rectangle placed into this level: $H_l = \max_{i \in P_l} h_i$. There can be several strategies used for placing the rectangles to the levels. The three most common ones are as follows:

- *Next-Fit Decreasing Height* (NFDH): Each rectangle is placed left justified into the current level if it fits. If the current level cannot accommodate the rectangle, a new level is created and the rectangle is placed left justified into this level.

- *First-Fit Decreasing Height* (FFDH): Each rectangle is placed left justified into the lowest level where it fits. If no level can accommodate the rectangle, a new level is created and the rectangle is placed left justified into this level.

- *Best-Fit Decreasing Height* (BFDH): Each rectangle is placed into the level where the unused horizontal space is minimum after the rectangle is placed. If no level can accommodate the rectangle, a new level is created and the rectangle is placed left justified into this level.

BFDH cannot be applied to RPP since the rectangles cannot move horizontally. However, in Section 4.4.3.1, we propose a new algorithm based on redefining unused space in the definition of BFDH. In RPP, when the rectangles are ordered according to their heights and placed according to NFDH or FFDH, FFDH performs better than NFDH since the placement of a rectangle in FFDH cannot be higher than NFDH which results in a better (if not the same) solution. Therefore, we consider FFDH algorithm and propose a variation of it, which we call the *Baseline Algorithm* (BA). The steps of BA are explained in Algorithm 5. BA terminates when all the rectangles are placed into a level. The height of the placement is the total height of the levels. Note that an important step in BA is selecting the next rectangle to be placed (Step 6 in Algorithm 5).

Algorithm 5 *Baseline Algorithm.*

```
1: Let $S$ be the set of unplaced rectangles. Set $S = R$.
2: Start with the first level: $l = 1$.
3: Let $P_l$ be the set of rectangles placed to level $l$. Set $P_l = \emptyset$.
4: $S_l = \{j \mid j \in S$ and $(j,i) \notin I$ and $(i,j) \notin I$ for all $i \in P_l\}$
5: if $S_l \neq \emptyset$ then
6:    Select a rectangle, say rectangle $j$, to be placed into level $l$ from $S_l$ and place it
      into level $l$. Update $P_l$, $P_l = P_l \cup \{j\}$, and $S_l$, $S_l = \{k \mid k \in S_l - \{j\}$ and $(j,k) \notin$
      $I$ and $(k,j) \notin I\}$.
7:    Go to Step 5.
8: else
9:    Calculate the height of level $l$: $H_l = \max_{i \in P_l} h_i$ and update $S$, $S = S - P_l$.
10:   if $S \neq \emptyset$ then
11:      Move to next level: $l = l + 1$ and go to Step 3.
12:   else
13:      Terminate the algorithm.
14:   end if
15: end if
```

In addition to level algorithms discussed above for 2SPP, a non-level algorithm called *Bottom-Left* (BL) is widely studied in the literature [59, 65]. In BL, after ordering the rectangles according to some rule, they are placed left justified to the lowest possible position. A very popular implementation is applying BL to the rectangles for 4 different ordering rules (height, width, perimeter, area) and returning the best solution [59]. Based on the implementation strategy of BL proposed by Hopper [59], we consider the following four selection rules at Step 6 of BA. Among the rectangles in S_l, select the rectangle with maximum (i) height (h_j), (ii) width (w_j), (iii) perimeter ($2(h_j + w_j)$), and (iv) area ($h_j w_j$).

As an implementation strategy, we run BA with these four selection rules and choose the best solution. We call this algorithm the *Deterministic Baseline Algorithm* (BA-Det). In addition to BA-Det, we also consider a randomized version of this algorithm (BA-Ran). In BA-Ran, after assigning a weight to all of the rectangles, among the rectangles in S_l, we make a random selection based on the weights; the larger the weight of the rectangle, the higher the chance of being selected. Similar to the four selection rules discussed above, we consider four different weight structures. Note that every time we implement this randomized version, we most likely obtain a different solution because of different random number streams. We implement the randomized version for a number of different random number streams for each of the four different weight rules and take the best solution.

4.4.2 Left Edge Algorithm (LEA)

Next, we discuss LEA which forms the basis for several channel routing algorithms [52] and and its applicability to RPP. Assuming that there is a set of horizontal segments (intervals) with left (l_j) and right (r_j) end points, LEA places these segments into levels so that no two intersecting segments are placed into the same level. The objective is to minimize the number of levels. LEA finds an optimal solution in polynomial time. LEA is summarized in Algorithm 6.

Algorithm 6 *Left Edge Algorithm.*

```
1: Sort the intervals according to a nondecreasing order of their left end points.
2: Place the intervals to the possible lowest level where there is not an interval
   intersecting with this interval.
```

We modify LEA for RPP by placing the next rectangle to the lowest possible position. Note that the resulting solution may not have a level structure since the rectangle is placed to the lowest position.

4.4.3 Two New Heuristics

The heuristics discussed in Sections 4.4.1 and 4.4.2 are the modifications of the heuristics proposed for similar problems in the literature. In this section, we propose two new heuristics for RPP, namely, *Baseline Waste Algorithm* (BWA) and *Knapsack Fill Algorithm* (KFA).

4.4.3.1 Baseline Waste Algorithm

We propose a modification of BA, called *Baseline Waste Algorithm* (BWA). Remember that BFDH for 2SPP considers the unused horizontal space at each level for the rectangle to be placed next and places this rectangle into the level in which the unused space is minimized. In BWA, we redefine the unused horizontal space and call it “waste”. In BWA, the next rectangle to be placed into the current level is selected based on this waste. The pseudocode for BWA is provided in Algorithm 7.

Algorithm 7 *Baseline Waste Algorithm.*

```
1: Let $S$ be the set of unplaced rectangles. Set $S = R$.
2: Start with the first level: $l = 1$.
3: Let $P_l$ be the set of rectangles placed to level $l$. Set $P_l = \emptyset$.
4: Select a rectangle from $S$, say $j$.
5: Place rectangle $j$ into level $l$, and update $P_l$, $P_l = P_l \cup \{j\}$.
6: Calculate the height of level $l$: $H_l = h_j$.
7: $S_l = \{k \mid k \in S, h_k \leq H_l$ and $(i,k) \notin I$ and $(k,i) \notin I$ for all $i \in P_l\}$
8: if $S_l \neq \emptyset$ then
9:    Select a rectangle, say rectangle $j$, to be placed into level $l$ from $S_l$ and place it
      into level $l$. Update $P_l$, $P_l = P_l \cup \{j\}$, and $S_l$, $S_l = \{k \mid k \in S_l - \{j\}$ and $(j,k) \notin$
      $I$ and $(k,j) \notin I\}$.
10:   Go to Step 8.
11: else
12:   Update $S$, $S = S - P_l$.
13:   if $S \neq \emptyset$ then
14:      Move to next level: $l = l + 1$ and go to Step 3.
15:   else
16:      Terminate the algorithm.
17:   end if
18: end if
```

In BWA, there are two important steps: Steps 4 and 9. In Step 4, we randomly select a rectangle from S, whereas in Step 9, we calculate the “waste” for each rectangle and place the one with minimum waste. We propose four alternatives for waste calculation which are illustrated in Figure 19, where A_i denotes the size of the relevant shaded area for $i = 1, 2, 3$, and H_l denotes the height of relevant level.

Assuming that we are considering rectangle i for placement we calculate the

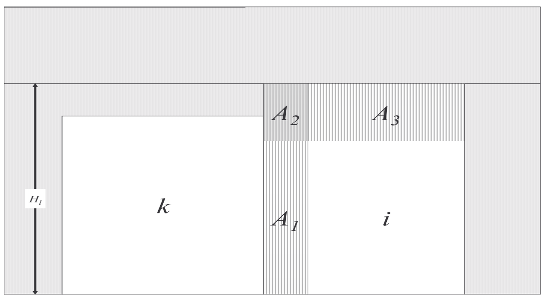


Figure 19: Illustration of the waste calculation rules

waste as follows: among the already placed rectangles into the current level, P_l, let rectangle k be the closest one (in terms of horizontal space) to rectangle i, i.e., $k \in \text{argmin}_{j \in P_l}(\min\{|l_j - r_i|, |l_i - r_j|\})$. Then, the four alternatives for waste calculation are: (i) A_1, (ii) $A_1 + A_2$, (iii) A_3, and (iv) $A_1 + A_2 + A_3$. In Step 9, we select the rectangle with the minimum waste. Note that in Step 4, we randomly select the initial rectangle. Similar to BA-Ran, we can run BWA several times for different random number streams to improve the best solution.

4.4.3.2 Knapsack Fill Algorithm

We develop an algorithm which uses the maximum weight independent set which can be found in polynomial time ($O(n\text{log}n)$) on interval graphs [63]. Finding the maximum weight independent set can be considered as solving a version of the knapsack problem with side constraints. Therefore, we call this algorithm *Knapsack Fill Algorithm* (KFA). The pseudocode for KFA is provided in Algorithm 8. In KFA, the main idea is to find the maximum weight independent set, say T, on G^{RPP} and place the rectangles in T to the current level, say l. After determining the height of the current level, $H_l = \max_{j \in T} h_j$, we find the maximum weight independent set, say T', among the unplaced rectangles such that the height of the current level, H_l, is not exceeded if the rectangles in T' are placed on top of the previously placed rectangles. After placing the rectangles in T', we repeat the same procedure until no more rectangles can be placed into the current level without exceeding the level height, H_l. Then, the next level is created.

Algorithm 8 *Knapsack Fill Algorithm.*

```
1: Let $S$ be the set of unplaced rectangles. Set $S = R$.
2: Start with the first level: $l = 1$.
3: Let $P_l$ be the set of rectangles placed to level $l$. Set $P_l = \emptyset$.
4: Find the maximum weight independent set, $MWIS(l)$, in $S$ and place the rect-
   angles in $MWIS(l)$ into level $l$. Update $P_l$, $P_l = P_l \cup MWIS(l)$ and update $S$,
   $S = S - MWIS(l)$.
5: Calculate the height of level $l$ as follows: $H_l = \max_{i \in P_l} h_i$.
6: Let $S_l \in S$ be the set of rectangles, that can be placed on top of the already
   placed rectangles in $P_l$ without exceeding the height of level $l$.
7: while $S_l \neq \emptyset$ do
8:    Find the maximum weight independent set, $MWIS(l)$, in $S_l$ and place the
      rectangles in $MWIS(l)$ into the current level.
9:    Update $P_l$, $P_l = P_l \cup MWIS(l)$ and update $S$, $S = S - MWIS(l)$.
10:   Update $S_l$.
11: end while
12: if $S \neq \emptyset$ then
13:    Set $l = l + 1$ and go to Step 3.
14: else
15:    Terminate the algorithm.
16: end if
```

The total height of the levels gives the height of the solution found by the algorithm. This is the basic version of the algorithm which we call *Deterministic Knapsack Fill Algorithm* (KFA-Det). Similar to the idea in BWA, we can randomly choose an initial rectangle to place to the current level. This is called the *Randomized Knapsack Fill Algorithm* (KFA-Ran) where Step 4 of Algorithm 8 is replaced by the following step:

4 : Select a rectangle, say j, from S and place it into level l and update P_l, $P_l = P_l \cup \{j\}$ and S, $S = S - \{j\}$.

In this randomized version, we can try different random number streams to possibly improve the solution. An important part of KFA-Det and KFA-Ran is defining the weight of the intervals (rectangles). Similar to BA, we consider four alternatives for defining the weight (W_i) of a rectangle: (i) height ($W_i = h_i$), (ii) width ($W_i = w_i$), (iii) perimeter ($W_i = h_i + w_i$), and (iv) area ($W_i = h_i w_i$).

4.4.4 Improving a Feasible Solution

In this section, we propose an improvement step for any feasible solution, including the solutions obtained by the above algorithms. After all the rectangles are placed, we consider "pushing down" the rectangles which may decrease the height of the placement. We illustrate this improvement step by an example. In Figure 20, on the left, we see the placement of the rectangles after applying the BA with selection rule height. However, if we consider pushing down the rectangles until they touch to a previously placed rectangle, the height of the placement can be improved.

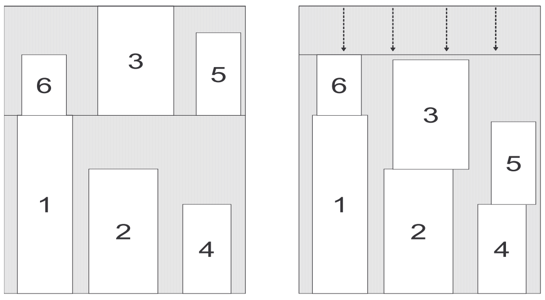


Figure 20: Improvement after the placement

4.5 Computational Results

In this section, we present the computational results based on three set of instances.

1. **2SPP**: These instances are generated from the instances of 2SPP that are widely tested in the literature [77].
2. **Random**: Randomly generated instances.
3. **RW**: Real-world instances generated based on the structure of the real world problems in the semiconductor industry.

The integer programming formulation proposed in Section 4.3 can solve only small instances with 20 rectangles. Since the lower bound proposed in Section 4.2 is better than the lower bound found by the integer program, in this section we compare the solution found by the heuristics to the lower bound proposed in Section 4.2.

The 2SPP instances discussed in Lodi et al. [77] can be categorized into 10 classes. In the first 4 classes, the width of the strip (W) is 100. Assuming that $H = 100$, the width (w_j) and the height (h_j) of the rectangles are generated as integers based on the types in Table 13. We use $U[a, b]$ to denote the uniform distribution with support $[a, b]$.

Table 13: Type of the rectangles generated in 2SPP instances

	Type 1	**Type 2**	**Type 3**	**Type 4**
w_j	$U[\frac{2}{3}W, W]$	$U[1, \frac{1}{2}W]$	$U[\frac{1}{2}W, W]$	$U[1, \frac{1}{2}W]$
h_j	$U[1, \frac{1}{2}H]$	$U[\frac{2}{3}H, H]$	$U[\frac{1}{2}H, H]$	$U[1, \frac{1}{2}H]$

The rectangles in Class k ($k \in \{1, 2, 3, 4\}$) are generated from Type k with 0.7 probability and from one of the remaining types with probability 0.1. In the last 6 classes, the strip width (W) and the rectangle sizes are generated as explained in Table 14.

Table 14: The setting in the last 6 classes of 2SPP instances

	Class 5	**Class 6**	**Class 7**	**Class 8**	**Class 9**	**Class 10**
W	10	30	40	100	100	300
w_j, h_j	$U[1, 10]$	$U[1, 10]$	$U[1, 35]$	$U[1, 35]$	$U[1, 100]$	$U[1, 100]$

For each class, 5 different settings ($\{100, 200, 300, 400, 500\}$) are considered for the number of rectangles. After generating the 2SPP instances, the horizontal locations of the rectangles are uniformly generated on the strip. For each class and number of rectangle setting, we generated 10 instances. In testing the performance of the heuristics, the randomized heuristics are run for 1000 times. The results are summarized in Table 15. The numbers in each column of Table 15 are the ratio of the best solution obtained by the corresponding heuristic to the lower bound (found by the maximum weight clique on G^{RPP}), and the numbers are the average of the 10 instances.

In Table 15, we see that proposed heuristics (BWA, KFA) perform better than the modifications of the existing heuristics in the literature. The average optimality gaps of the solutions found by BWA, KFA-Det and KFA-Ran are 2.24%, 1.93% and 1.72%, respectively. On the other hand, the average optimality gaps of the solutions found by LEA, BA-Det and BA-Ran are 5.33%, 3.29% and 6.50%, respectively. Among the modifications of the existing heuristics in the literature, BA-Det performs better.

As the number of rectangles increase, BA-Det and KFA-Det perform better than their randomized versions (BA-Ran and KFA-Ran). This is mainly due to large number of different placement combinations as the number of the rectangles increase. BA-Det and KFA-Det find a "good" placement of the rectangles by matching them based on their sizes. However, BA-Ran and KFA-Ran tries different combinations to find a better placement, and as the problem size (number of rectangles) increases, the number of different combinations increase which makes finding a "good" placement by random trials harder. From this computational study, we understand that despite the slightly worse performance of KFA-Det on small 2SPP instances, it can be a better alternative due to better performance on the large instances. Furthermore, the performance of the heuristics is relatively worse on the instances from Classes 8 and 10.

In addition to these 2SPP instances, we generate random instances to analyze the effect of the variability in the height and width of the rectangles on the performance of the proposed heuristics. Assuming that the width (W) of the strip and H is 100, the random instances are generated as explained in Table 16.

For each class, five different settings are considered for the number of rectangles: $\{100, 200, 300, 400, 500\}$. The horizontal locations of the rectangles are uniformly generated on the strip. Similar to 2SPP instances, 10 instances are generated for each class and number of rectangles setting. The results of the computational experiments

Table 15: Performance of the algorithms on 2SPP instances

Number of Rectangles	Class	LEA	BA - Det	BA - Ran	BWA	KFA - Det	KFA - Ran
100	**1**	1.0249	1.0105	1.0038	1.0061	1.0075	1.0028
	2	1.1026	1.0382	1.0281	1.0170	1.0328	1.0226
	3	1.0169	1.0029	1.0010	1.0019	1.0040	1.0011
	4	1.0716	1.0430	1.0475	1.0281	1.0218	1.0154
	5	1.0169	1.0072	1.0084	1.0022	1.0058	1.0013
	6	1.0696	1.0594	1.0572	1.0236	1.0294	1.0119
	7	1.0509	1.0187	1.0180	1.0109	1.0080	1.0063
	8	1.1262	1.0860	1.0907	1.0492	1.0509	1.0315
	9	1.0544	1.0124	1.0085	1.0066	1.0160	1.0034
	10	1.1442	1.0748	1.0645	1.0406	1.0481	1.0202
200	**1**	1.0278	1.0060	1.0046	1.0041	1.0045	1.0024
	2	1.0758	1.0381	1.0489	1.0177	1.0286	1.0267
	3	1.0175	1.0016	1.0016	1.0011	1.0019	1.0006
	4	1.0809	1.0383	1.0752	1.0372	1.0207	1.0277
	5	1.0114	1.0028	1.0166	1.0013	1.0032	1.0004
	6	1.0449	1.0509	1.0937	1.0178	1.0218	1.0125
	7	1.0500	1.0244	1.0395	1.0158	1.0173	1.0140
	8	1.0998	1.0735	1.1153	1.0542	1.0484	1.0444
	9	1.0331	1.0102	1.0176	1.0071	1.0071	1.0046
	10	1.1178	1.0983	1.1370	1.0674	1.0560	1.0348
300	**1**	1.0234	1.0085	1.0109	1.0061	1.0062	1.0041
	2	1.0619	1.0266	1.0533	1.0192	1.0241	1.0242
	3	1.0225	1.0038	1.0053	1.0029	1.0041	1.0023
	4	1.0637	1.0355	1.0842	1.0347	1.0199	1.0240
	5	1.0021	1.0027	1.0179	1.0012	1.0028	1.0006
	6	1.0268	1.0449	1.1214	1.0229	1.0185	1.0159
	7	1.0379	1.0115	1.0369	1.0097	1.0067	1.0076
	8	1.0990	1.0821	1.1656	1.0624	1.0368	1.0552
	9	1.0363	1.0106	1.0285	1.0110	1.0086	1.0087
	10	1.1198	1.0966	1.1771	1.0716	1.0499	1.0416
400	**1**	1.0309	1.0085	1.0165	1.0080	1.0070	1.0055
	2	1.0530	1.0317	1.0684	1.0183	1.0240	1.0275
	3	1.0150	1.0031	1.0061	1.0027	1.0028	1.0024
	4	1.0746	1.0358	1.0989	1.0399	1.0211	1.0307
	5	1.0036	1.0028	1.0283	1.0014	1.0024	1.0006
	6	1.0338	1.0457	1.1432	1.0214	1.0164	1.0147
	7	1.0262	1.0125	1.0422	1.0092	1.0081	1.0092
	8	1.0939	1.0705	1.1862	1.0580	1.0385	1.0536
	9	1.0435	1.0147	1.0398	1.0140	1.0119	1.0137
	10	1.1097	1.0964	1.1908	1.0704	1.0530	1.0495
500	**1**	1.0231	1.0081	1.0185	1.0069	1.0052	1.0054
	2	1.0554	1.0307	1.0692	1.0171	1.0254	1.0266
	3	1.0135	1.0028	1.0057	1.0026	1.0035	1.0023
	4	1.0576	1.0220	1.0878	1.0279	1.0137	1.0217
	5	1.0023	1.0030	1.0295	1.0021	1.0012	1.0007
	6	1.0207	1.0430	1.1591	1.0224	1.0151	1.0157
	7	1.0313	1.0143	1.0523	1.0122	1.0079	1.0114
	8	1.0826	1.0723	1.1819	1.0511	1.0339	1.0471
	9	1.0422	1.0153	1.0447	1.0145	1.0092	1.0134
	10	1.1220	1.0903	1.2016	1.0701	1.0532	1.0404
Average		**1.0533**	**1.0329**	**1.0650**	**1.0224**	**1.0193**	**1.0172**

Table 16: The setting of the random instances

	Class 1	**Class 2**	**Class 3**	**Class 4**	**Class 5**	**Class 6**	**Class 7**	**Class 8**
w_j	$U[1, W]$	$U[1, \frac{1}{2}W]$	$U[1, \frac{1}{4}W]$	$U[1, \frac{1}{8}W]$	$U[1, W]$	$U[1, \frac{1}{2}W]$	$U[1, \frac{1}{4}W]$	$U[1, \frac{1}{8}W]$
h_j	$U[1, H]$	$U[1, H]$	$U[1, H]$	$U[1, H]$	$U[\frac{2}{5}H, \frac{3}{5}H]$	$U[\frac{2}{5}H, \frac{3}{5}H]$	$U[\frac{2}{5}H, \frac{3}{5}H]$	$U[\frac{2}{5}H, \frac{3}{5}H]$

are summarized in Table 17. Similar to Table 15, the numbers in each column are the ratio of the best solution obtained by the corresponding heuristic to the lower bound.

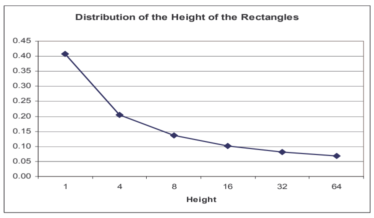
Table 17 shows the effect of number of trials (10, 100 and 1000) on the performance of BWA and KFA-Ran. When the number of trials decreases, the quality of the solutions found by BWA and KFA-Ran deteriorates. When the instances become larger, running the randomized algorithms can be prohibitive because of the run times. For example, for 500 rectangle instances running 1000 trials for BWA and KFA-Ran takes around 45 and 85 CPU seconds. However, KFA-Det runs in less than a second and the solution found by KFA-Det is slightly worse than the solutions found by BWA and KFA-Ran after 1000 trials. Therefore, if the problem size is very large or a near-optimal solution is required in seconds KFA-Det can be a better alternative. We see that when the variability in the height of the rectangles increases, the performance of the heuristics deteriorates. As the height of the rectangles become close to each other, it is easier to find a match between the rectangles so that the total height of the placement is minimized. This argument is also supported by LEA which solves the special case where the heights of the rectangles are same in polynomial time. Furthermore, we see that BWA performs better when the variability in the height of the rectangles decreases (see Classes R5-R8 in Table 17). Finally, if the variability in the width of the rectangles is very small (see Classes R4 and R8) or very large (see Classes R1 and R5), the heuristics perform better.

We propose an improvement step in Section 4.4. Based on the random instances, this step improves the solution of BA-Det, BA-Ran, BWA, KFA-Det and KFA-Ran by 1.16%, 7.13%, 3.39%, 2.37%, 2.29%.

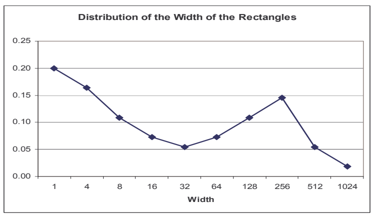
Table 17: Performance of the algorithms on random instances

Number of Rectangles	**Class**	**BWA**			**KFA - Det**	**KFA - Ran**		
		1000	**100**	**10**		**1000**	**100**	**10**
100	**R1**	1.0103	1.0149	1.0265	1.0155	1.0075	1.0130	1.0249
	R2	1.0424	1.0591	1.0862	1.0427	1.0319	1.0540	1.0723
	R3	1.0299	1.0442	1.0763	1.0484	1.0197	1.0381	1.0594
	R4	1.0048	1.0259	1.0583	1.0375	1.0077	1.0222	1.0530
	R5	1.0055	1.0095	1.0161	1.0187	1.0078	1.0131	1.0180
	R6	1.0158	1.0238	1.0372	1.0432	1.0295	1.0391	1.0581
	R7	1.0136	1.0270	1.0340	1.0485	1.0131	1.0304	1.0542
	R8	1.0002	1.0016	1.0207	1.0212	1.0001	1.0038	1.0225
200	**R1**	1.0090	1.0125	1.0167	1.0093	1.0064	1.0090	1.0144
	R2	1.0473	1.0547	1.0696	1.0371	1.0445	1.0580	1.0705
	R3	1.0685	1.0864	1.1109	1.0623	1.0481	1.0612	1.0897
	R4	1.0340	1.0510	1.0766	1.0539	1.0182	1.0341	1.0535
	R5	1.0051	1.0078	1.0133	1.0165	1.0102	1.0130	1.0183
	R6	1.0158	1.0210	1.0269	1.0330	1.0276	1.0364	1.0423
	R7	1.0244	1.0313	1.0457	1.0436	1.0277	1.0354	1.0611
	R8	1.0106	1.0172	1.0333	1.0407	1.0101	1.0230	1.0320
300	**R1**	1.0154	1.0175	1.0242	1.0123	1.0130	1.0160	1.0225
	R2	1.0465	1.0540	1.0669	1.0293	1.0480	1.0538	1.0612
	R3	1.0536	1.0682	1.0873	1.0473	1.0402	1.0492	1.0666
	R4	1.0557	1.0752	1.0914	1.0497	1.0341	1.0461	1.0691
	R5	1.0070	1.0087	1.0131	1.0168	1.0121	1.0142	1.0184
	R6	1.0139	1.0196	1.0274	1.0326	1.0278	1.0318	1.0431
	R7	1.0234	1.0310	1.0429	1.0367	1.0262	1.0327	1.0478
	R8	1.0163	1.0235	1.0461	1.0298	1.0141	1.0214	1.0446
400	**R1**	1.0121	1.0148	1.0169	1.0116	1.0117	1.0134	1.0188
	R2	1.0461	1.0495	1.0594	1.0276	1.0470	1.0530	1.0595
	R3	1.0774	1.0867	1.1015	1.0529	1.0582	1.0694	1.0828
	R4	1.0419	1.0575	1.0722	1.0365	1.0260	1.0369	1.0593
	R5	1.0059	1.0084	1.0109	1.0165	1.0119	1.0135	1.0178
	R6	1.0138	1.0174	1.0232	1.0278	1.0271	1.0323	1.0373
	R7	1.0172	1.0239	1.0319	1.0294	1.0201	1.0276	1.0339
	R8	1.0117	1.0182	1.0247	1.0369	1.0106	1.0155	1.0247
500	**R1**	1.0112	1.0141	1.0157	1.0095	1.0114	1.0128	1.0162
	R2	1.0381	1.0442	1.0486	1.0271	1.0431	1.0479	1.0558
	R3	1.0636	1.0760	1.0882	1.0475	1.0501	1.0592	1.0707
	R4	1.0526	1.0636	1.0812	1.0458	1.0299	1.0422	1.0564
	R5	1.0049	1.0060	1.0085	1.0140	1.0094	1.0100	1.0135
	R6	1.0163	1.0207	1.0256	1.0287	1.0337	1.0390	1.0442
	R7	1.0263	1.0341	1.0440	1.0380	1.0300	1.0378	1.0487
	R8	1.0124	1.0181	1.0299	1.0261	1.0123	1.0185	1.0261
Average		**1.0255**	**1.0335**	**1.0457**	**1.0326**	**1.0240**	**1.0320**	**1.0446**

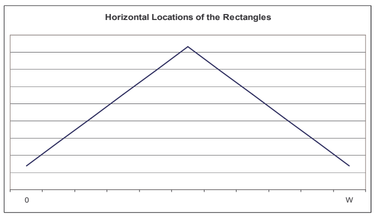
Finally, based on personal communication with the semiconductor companies, the following instances are generated to imitate the real-world instances. In these instances, the width of the strip is 1024. The height (h_j) of the rectangles can be 1, 4, 8, 16, 32, 64, and the width (w_j) of the rectangles can be 1, 4, 8, 16, 32, 64, 128, 256, 512 and 1024. The height and the width of the rectangles are generated based on the probability mass functions provided in Figures 21(a) and 21(b). In the real-world instances, the rectangles are densely placed in the middle of the strip. Therefore, the horizontal locations are generated using a probability function as illustrated in Figure 21(c). Three different settings are considered for the number of rectangles: 10, 100, 1000.



(a) Distribution of the height of the rectangles



(b) Distribution of the width of the rectangles



(c) Distribution of the horizontal locations of the rectangles

Figure 21: Structure of the real world instances

Based on 100 instances for each setting, the average ratios of the solutions found to the best lower bound are summarized in Table 18. The average optimality gap of the solutions obtained by KFA-Ran is around 1.0% on real-world instances. Similar to previous results, as the number of rectangles increases, the performances of the randomized heuristics (BA-Ran, BWA, KFA-Ran) deteriorate.

Table 18: Performance of the algorithms on real-world instances

Number of Rectangles	LEA	BA - Det	BA - Ran	BWA	KFA - Det	KFA - Ran
10	1.0302	1.0032	1.0000	1.0000	1.0071	1.0000
100	1.1115	1.0549	1.0185	1.0190	1.0565	1.0026
1000	1.0698	1.0403	1.1546	1.0354	1.0126	1.0127

For small instances, KFA-Ran is a better alternative because the random trials increase the probability of finding a better solution. However, as the size of the instance increases, KFA-Det performs better than KFA-Ran. The higher number of different placement combinations makes it hard to find a "good" solution by random trials. In addition, the increased run times of KFA-Ran and BWA make them less preferable over KFA-Det.

4.6 Conclusions and Contributions

In this chapter, we introduce a rectangle packing problem motivated by the practice in the semiconductor industry. We propose an integer programming formulation and discuss ways to improve it. We study the complexity of the problem, propose a lower bound and discuss a special case that can be solved in polynomial time. Since the problem is strongly NP-hard, we develop heuristics which are efficient and effective.

Although we generate valid cuts to improve the integer program, generating all the cuts at the beginning is not very efficient because of the number of cuts. A future direction is developing a branch-and-cut algorithm to effectively utilize these cuts.

Another interesting problem is the one with vertical and/or horizontal interval constraints. In this problem, the rectangles can be placed into certain vertical and/or horizontal intervals. In this version of the problem, the feasibility of the problem is crucial. Depending on the instance, it may not be feasible. Therefore, finding a feasible solution is an important problem.

CHAPTER V

CONCLUSION

In this thesis, we studied three applications of OR/MS using integer programming and simulation tools. In the Chapter 2, we studied food distribution logistics during an influenza pandemic. We developed an agent-based disease spread model and integrated it with a facility location model for designing the food distribution network. Since the corresponding facility location problem is hard to solve for large instances, we designed efficient algorithms to find near-optimal solutions. Furthermore, we investigated the effect of timing and length of non-pharmaceutical intervention strategies, namely, voluntary quarantine and school closure, on the disease spread and food distribution supply chain. A voluntary quarantine with appropriate timing and length can reduce the infections at the peak as well as the likelihood of capacity bottlenecks and supply chain disruptions significantly and may have a greater positive impact when compared to school closure. In addition, we analyzed the changes on the work absenteeism during an influenza pandemic. Our results have implications for governments, non-governmental organizations and private industry preparing response plans for an influenza pandemic.

In Chapter 3, we introduced a scheduling problem motivated by the challenges in the commercial lithographic printing industry. We studied the complexity of the problem and discussed some special cases that are polynomially solvable. Motivated by these special cases and the heuristics in the lot-sizing literature, we developed efficient and effective heuristics that find near-optimal solutions within seconds. We also proved that one of the heuristics returns a 2-approximate solution. Furthermore, we proposed an integer programming formulation that can utilize special branching rules and strengthened it by preprocessing ideas and valid cuts.

In Chapter 4, we studied a variation of the two-dimensional strip packing problem which is motivated by the chip design problem in the semiconductor industry. We showed that the problem is NP-hard, and after exploiting the problem structure, we proposed a lower bound which can be used to evaluate the performance of the heuristics. We discussed the applicability of the heuristics that are proposed for similar problems in the literature, and proposed new heuristics using the maximum weight independent set on interval graphs. Computational experiments on randomly generated instances show that the new heuristics are more effective than the readily available heuristics for similar problems.

APPENDIX A

SUPPLEMENTARY MATERIALS FOR CHAPTER 2

A.1 Literature Review About Modeling Influenza Pandemic and Annual Influenza

Table 19: Literature review for influenza pandemic

Spread Model	**Reference**	**Main Goal**
Random Graphs (Heterogeneous mixing)	Glass et al. [45]	Study targeted social distancing as an intervention policy within a local community
	Carrat et al. [14]	Study intervention strategies such as vaccination, neuraminidase inhibitors, quarantine and closure of schools or workplaces
Difference Equations (Homogeneous mixing)	Rvachev and Longini [92]	Forecast the global spread of Hong Kong flu based on estimated parameters from Hong Kong
	Grais et al. [47]	Study the effect of current international airline traffic by simulating the 1968 pandemic with the current volume of airline traffic for 52 global cities
	Flahault et al. [37]	Study the impact of intervention strategies such as vaccination, isolation, antiviral usage and air traffic reduction on the global spread using the model developed by Rvachev and Longini [92]
	Larson [72]	Investigate social distancing as an intervention strategy
Differential Equations (Homogeneous mixing)	Flahault et al. [36]	Simulate the spread of influenza for 9 cities in Europe considering only the regular air traffic
	Ferguson et al. [35]	Investigate neuraminidase inhibitor therapy on drug-resistant annual influenza
	Chowell et al. [18]	Evaluate hypothetical public health measures during the 1918 influenza pandemic in Geneva, Switzerland
	Glasser et al. [46]	Study targeted vaccination strategies for annual influenza
	Wu et al. [112]	Study geography-based allocation of vaccines
Simulation (Heterogeneous mixing)	Longini et al. [78]	Study intervention strategies such as vaccination, antiviral usage and quarantine for rural Southeast Asia
	Patel et al. [88]	Find optimal vaccine usage to minimize the number of illnesses given limited quantities of vaccine
	Ferguson et al. [32]	Evaluate targeted antiviral usage for Thailand
	Wu et al. [111]	Investigate household-based interventions such as quarantine, contact tracing, antiviral usage and isolation for Hong Kong
	Germann et al. [42]	Study intervention policies such as travel restrictions, vaccine and antiviral usage for the US
	Ferguson et al. [33]	Study intervention strategies such as vaccine and antiviral usage, isolation, quarantine, school or workplace closure, travel restrictions for Great Britain and the US
	Das et al. [22]	Study the impact of an influenza pandemic from several perspectives such as infected, dead, healthcare cost and lost wages
	Lee et al. [73]	Investigate the spread pattern of the influenza pandemic in Norfolk, VA via a detailed social network model and city level details such as climate, geography and economy
	Halloran et al. [51]	Study intervention strategies such as antiviral treatment, quarantine, school closure, community and workplace social distancing for Chicago

A.2 Natural Disease History Parameters

Table 20: Parameters for natural history of disease

Parameter	Value	References
p_A	0.4 for working adults (19-64) and 0.25 for others	Ferguson et al. [35], Wu et al. [111], Germann et al. [42], Longini et al. [78]
p_H	0.18 for children between 0 and 5, 0.12 for elderly (65+) and 0.06 for others	Longini et al. [78], Wu et al. [111]
p_D	0.344 for elderly and children between 0 and 5 and 0.172 for others	Carrat et al. [14], Wu et al. [111]
Duration of $E + I_P$	Weibull distribution with mean 1.48 days (including an offset of 0.5 days) and standard deviation 0.47	Ferguson et al. [32], Wu et al. [111]
Duration of I_P	0.5 days	Ferguson et al. [32], Wu et al. [111]
Duration of I_S	Exponential distribution with mean 2.7313 days	Wu et al. [111]
Duration of I_A	Exponential distribution with mean 1.63878 days	Wu et al. [111]
Duration of I_H	Exponential distribution with mean 14 days	Ferguson et al. [32], Wu et al. [111]

A.3 Comparison of Our Disease Spread Model and the Models in the Literature

Table 21: Comparison of the proposed model with the ones in the literature

Reference	Natural History	Spatial Component	Age-Based	Night/Day Differentiation
Wu et al. [111]	Detailed SEIR	No	No	No
Ferguson et al. [33], Ferguson et al. [32], Patel et al. [88], Longini et al. [78]	SEIR	Yes	Yes	No
Germann et al. [42]	SEIR	Yes	Yes	Yes
Our Model	Detailed SEIR	Yes	Yes	Yes

A.4 Details of Disease Spread Model

In this section, we explain the details of the disease spread model. In the contact network, workplace sizes are generated using a Poisson distribution with mean 20, and the maximum workplace size is assumed to be 1000 (similar to Germann et al. [42]). The working adults mix with other working adults depending on the tract-to-tract worker flow data [105]. The average peer group sizes are 14, 20 and 30 for kindergarten, elementary schools and secondary schools, respectively [41].

At the beginning of the simulation, every individual is assumed to be susceptible. We introduce an initial number (30) of infected individuals randomly to the community to represent the entrance of the virus to the population. The length of the simulation is 365 days, which is sufficient to capture the disease spread.

Different disease settings are defined by the coefficient of transmission (β), the relative hazards of an infected individual at presmpytomatic and asymptomatic stages (h_P and h_A) to symptomatic stage and relative hazards in peer groups and community to households (h_G and h_C) [111]. As the base case, we take the hazard of an individual at the symptomatic stage, h_S, and the hazard in households, h_H, as 1. We make age-based adjustments to the calculation of these parameters. Let r_{XY} be the average number of people infected in Y by an individual who is at stage X where Y is the household (H), peer group (G) or the community (C) and X is the presymptomatic (P), asymptomatic (A) or symptomatic (S) stage. The r_{XY} values are calculated as follows.

$$r_{PH} = \sum_{n=1}^{7} p_n(n-1)(1-\phi_P(\tfrac{h_P\beta}{2n}))$$
$$r_{AH} = p_A \sum_{n=1}^{7} p_n(n-1)\phi_P(\tfrac{h_P\beta}{2n})(1-\phi_A(\tfrac{h_A\beta}{2n}))$$
$$r_{SH} = (1-p_A) \sum_{n=1}^{7} p_n(n-1)\phi_P(\tfrac{h_P\beta}{2n})(1-\phi_S(\tfrac{\beta}{2n}))$$
$$r_{PG} = (q_1n_1+q_2n_2+q_3n_3+q_4n_4+q_5n_5)(1-\phi_P(\tfrac{h_Ph_G\beta}{2}))$$
$$r_{AG} = p_A(q_1n_1+q_2n_2+q_3n_3+q_4n_4+q_5n_5)\phi_P(\tfrac{h_Ph_G\beta}{2})(1-\phi_A(\tfrac{h_Ah_G\beta}{2}))$$
$$r_{SG} = (1-p_A)((q_1n_1+q_2n_2+q_3n_3)\phi_P(\tfrac{h_Ph_G\beta}{2})(1-\phi_S(0))+(q_4n_4+q_5n_5)\phi_P(\tfrac{h_Ph_G\beta}{2})(1-\phi_S(\tfrac{h_G\beta}{4})))$$
$$r_{PC} = N(1-\phi_P(\tfrac{h_Ph_C\beta}{N}))$$
$$r_{AC} = p_AN\phi_P(\tfrac{h_Ph_C\beta}{N})(1-\phi_A(\tfrac{h_Ah_C\beta}{N}))$$
$$r_{SC} = (1-p_A)N\phi_P(\tfrac{h_Ph_C\beta}{N})(1-\phi_S(\tfrac{h_C\beta}{N}))$$

The proportion of the population in age group i for $i \in \{1,2,3,4,5\}$ is q_i, and p_A is the probability that a presymptomatic individual does not develop symptoms. Using the probability of developing symptoms in Table 20, p_A is calculated as follows:

$$p_A = 0.25q_1 + 0.25q_2 + 0.25q_3 + 0.40q_4 + 0.25q_5.$$

n_i is the average size of peer groups for age group i. In our model, the maximum household size is 7 [105]. p_n is the probability that an individual lives in a household size of n, and N is the total number of people in the considered area. $\phi_P(h)$ is the probability that an infection does not occur between two individuals during the presymptomatic phase for a constant hazard of infection h. $\phi_A(h)$ and $\phi_S(h)$ are the similar probabilities of no infection during the asymptomatic and symptomatic phases for constant hazard of infection h. In addition, $\phi_X(h)$ is defined as follows:

$$\phi_X(h) = E(e^{-hD_X}) = \int_0^\infty e^{-ht} f_X(t)dt,$$

where the duration (D_X) of disease stage X for $X \in \{P, A, S\}$ is defined by probability density function f_X for which values are given in Table 20. The r_{XY} values are used to calculate the following five disease parameters.

$$R_0 = \sum_{X\in\{P,A,S\}} \sum_{Y\in\{H,G,C\}} r_{XY}$$

$$\theta = \frac{\sum_{X\in\{P,A\}} \sum_{Y\in\{H,G,C\}} r_{XY}}{R_0}$$

$$\omega = \frac{\sum_{Y\in\{H,G,C\}} r_{AY} + p_A r_{PY}}{R_0}$$

$$\gamma = \frac{\sum_{X\in\{P,A,S\}} \sum_{Y\in\{G,C\}} r_{XY}}{R_0}$$

$$\delta = \frac{\sum_{X\in\{P,A,S\}} r_{XC}}{\sum_{X\in\{P,A,S\}} \sum_{Y\in\{G,C\}} r_{XY}}$$

R_0 is defined in the main text. θ is the proportion of transmission that occurs at either presymptomatic or asymptomatic stage. ω is the proportion of infections generated by individuals who are never symptomatic. γ is the proportion of transmission that occurs outside the households. δ is the proportion of transmission outside the home that occurs in the community.

After setting these five parameters, the coefficient of transmission (β) and the relative hazards (h_P, h_A, h_G, h_C) can be calculated. We assume that θ is 0.3 and ω is 0.15 [111], 70% of the infections occur outside the household, and half of these infections occur within the peer groups. These estimates are consistent with the estimates in Ferguson et al. [33].

In the disease spread model, we simulate the time of next infection and choose the individual that will be infected. The following infection time is generated by calculating the "instantaneous force of infection" for each individual [111] using the parameters ($\beta, h_P, h_A, h_G, h_C$), which are discussed above. We have adjusted the calculation of force of infection for our age-based model using the age-based parameters (see below for the calculation of the force of infection). After the infected individual is selected, the disease progresses according to the natural history with the assumed transition times and probabilities for the influenza.

In our model, we assume that the relative infectivity of the children (0-18) compared to adults (19+) is 1.5 and the relative susceptibility of the children compared to adults is 1.15 [14]. The relative infectivity values are adjusted from the corresponding probability of transmission values between children and adults proposed by Carrat et al. [14]. The susceptibility and infectivity parameters are normalized so that the expected susceptibility of an individual is 1.0, and the expected infectivity of an individual is 1.0, h_P and h_A for symptomatic, presymptomatic and asymptomatic cases, respectively.

Using these parameters, the force of infection experienced by the i^{th} individual during the day (λ_i^D) and during the night (λ_i^N) are calculated as follows:

$$\lambda_i^D = S_i \sum_{j=1}^{N} (\delta_{ij}^G m_j \epsilon_j h_G \beta + \delta_{ij}^C \frac{m_j h_C \beta}{N_i}),$$
$$\lambda_i^N = S_i \sum_{j=1}^{N} (\delta_{ij}^H \frac{m_j \beta}{n_i^{HA}} + \delta_{ij}^C \frac{m_j h_C \beta}{N_i}).$$

In the above equations, S_i is the relative susceptibility of the individual and

$$S_i = \begin{cases} S_C, & \text{if individual } i \text{ is a child} \\ S_A, & \text{otherwise.} \end{cases}$$

Let q_A be the proportion of adults in our population, then S_C and S_A can be calculated using the following equations.

$$(1 - q_A)S_C + q_A S_A = 1.0$$
$$S_C = 1.15 S_A$$

N_i is the number of individuals in the i^{th} individual's community, and n_i^{HA} is the active household size of this individual where dead and hospitalized individuals are not counted. δ_{ij}^H, δ_{ij}^G and δ_{ij}^C are the indicator functions defined for households, peer groups and the community, respectively.

$$\delta_{ij}^Y = \begin{cases} 1, & \text{if individuals } i \text{ and } j \text{ are in the same } Y \\ 0, & \text{otherwise} \end{cases} \qquad Y \in \{H, G, C\}.$$

ϵ_j is the indicator variable showing whether the j^{th} individual withdraws from work or school.

$$\epsilon_j = \begin{cases} 0, & \text{with probability 1.0 if individual } j \text{ is a symptomatic child} \\ 0, & \text{with probability 0.5 if individual } j \text{ is a symptomatic adult} \\ 1, & \text{otherwise} \end{cases}$$

Finally, m_j (the infectivity of individual j) is defined as follows:

$$m_j = \begin{cases} I_X^C, & \text{if } j \text{ is a child at stage } X \\ I_X^A, & \text{if } j \text{ is an adult at stage } X \\ 0, & \text{otherwise,} \end{cases}$$

where I_X^C and I_X^A are the infectivity of an infected child and an infected adult at stage X (for $X \in \{P, A, S\}$), respectively. The values of these infectivity parameters are calculated as follows by using the expected relative hazard of an individual.

$$(1 - q_A)I_X^C + q_A I_X^A = \begin{cases} h_P, & \text{if } X = P \\ h_A, & \text{if } X = A \\ 1.0, & \text{if } X = S \end{cases}$$

$$I_X^C = 1.5 I_X^A$$

A.5 Model Validation

Table 22: Calibrated parameters to achieve the age-specific clinical attack rates for the 1957 pandemic

Parameter		**Calibrated**	**Original**
Work place sizes		30	20
p_A for elderly		0.5	0.25
Proportion of community infections in total number of outside home infections (θ)		0.2	0.5
Susceptibility of children	0-5	1.8	1.15
	6-11	1.7	1.15
	12-18	1.6	1.15

Table 23: Comparison of age-specific clinical attack rates

Age Group	Our Model	1957 Pandemic
1 (0-5)	32.62%	32.17%
2 (6-11)	35.18%	35.02%
3 (12-18)	39.08%	38.44%
4 (19-64)	22.07%	22.24%
5 (65+)	10.45%	10.00%
Total	**24.72%**	**24.72%**

A.6 Voluntary Quarantine Results

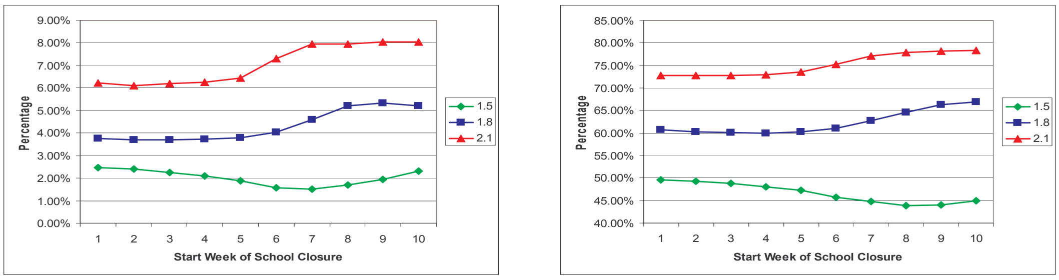
Table 24: Reduction in the performance measures for an 8-week quarantine

R_0	Peak Infectivity	CAR	IAR	Death Ratio
1.5	67.74%	18.40%	18.51%	18.22%
1.8	64.71%	16.70%	16.82%	17.22%
2.1	50.44%	19.52%	19.68%	19.68%

Table 25: Comparison of the quarantine results with the results in the literature

		Quarantine		Decrease in	Decrease in
Reference	R_0	Start - End	Compliance Rate	Peak Infectivity	IAR
Longini et al. [78]	1.4-1.7	Day 14 - Day 365	70%	Not reported	99%
Ferguson et al. [33]	1.7-2.0	Day 1 - Day 365	50%	25-26%	14.7-18.5%
Wu et al. [111]	1.8	Day 1 - Day 365	50%	70%	33%
Our model	1.8	Day 21 - Day 77	50%	64.71%	16.82%

A.7 School Closure Results



(a) Peak infectivity (b) Infection attack rate

Figure 22: Effect of timing of school closure on the peak infectivity and infection attack rate for different R_0 values

A.8 Map of Metropolitan Atlanta Area

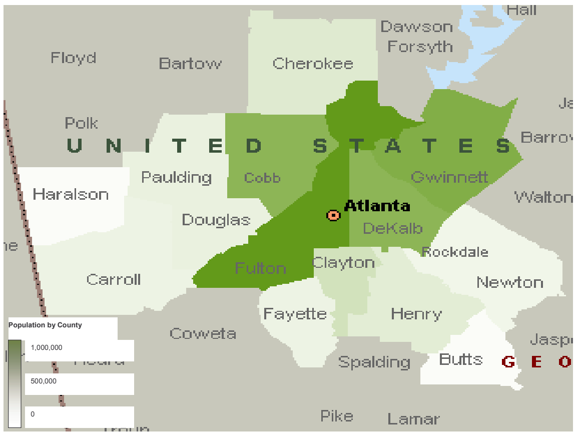


Figure 23: Counties in the Metropolitan Atlanta Area by population

A.9 Add-Drop Heuristic (ADH)

```
1: for $t = 1$ to $T$ do
2:   for $k = 1, 2$ do
3:     Let $PO^{POD}$ and $PO^{MF}$ be the set of open PODs and major facilities in
       period $t-1$, respectively. Let $FO^{POD}$ and $FO^{MF}$ be the set of PODs and
       major facilities that are planned to be open in the future.
4:     if k==1 then
5:       $D_{kt} = \frac{D_{kT}}{2^{T-t}} + \sum_{j=t+1}^{T} \frac{D_{kj}}{2^{j-t}}$
6:       $FO^{POD} = FO^{MF} = \emptyset$
7:     end if
8:     for $i \in N_2$ do
9:       Set the total serving cost to zero : $TotalServingCostMF_i = 0$.
10:      Calculate the cost of serving a unit of demand : $UnitServingCostMF_i =$
         $c_u^1 \min_{s \in N_1} d_{si}$.
11:    end for
12:    for $j \in N_3$ do
13:      Set the total serving cost to zero : $TotalServingCostPOD_j = 0$.
14:      Calculate the cost of serving a unit of demand : $UnitServingCostPOD_j =$
         $\min_{i \in PO^{MF}}(c_u^2 d_{ij} + UnitServingCostMF_i)$.
15:    end for
16:    Assign all the demand to the closest POD as the capacity permits.
17:    $CO^{POD}$ : Set of PODs that have been assigned a demand.
18:    $POD_{kj}$ : Demand amount of demand node $k$ assigned to POD $j$ for all
       $k \in N_4, j \in N_3$.
19:    Update the total serving cost of each POD that has a demand assigned to it :
       $TotalServingCostPOD_j = F_j + \sum_{k \in N_4} POD_{kj}(d_{jk}c_{individual} + UnitServingCostPOD_j)$
20:    repeat
```

```
21:   for $j \in CO^{POD}$ do
22:     if $j \in PO^{POD} \bigcup FO^{POD}$ then
23:       $CostSavingPOD_j = TotalServingCostPOD_j$
24:     else
25:       $CostSavingPOD_j = TotalServingCostPOD_j + f_j + g_j$
26:     end if
27:   end for
28:   for $j \in CO^{POD}$ do
29:     repeat
30:       Find the closest POD, say $j'$, with an assigned demand.
31:       Assign the demand of POD $j$ to POD $j'$ as the capacity permits. Let
          $POD_{kj}^{j'}$ be the demand of demand node $k$ that has been reassigned to
          POD $j'$ instead of POD $j$.
32:       Update the cost saving : $CostSavingPOD_j = CostSavingPOD_j -$
          $\sum_{k \in N_4} POD_{kj}^{j'}(d_{jk}c_{individual} + UnitServingCostPOD_{j'})$
33:     until All the demand of POD $j$ can be assigned to other open PODs
34:     if All the demand of POD $j$ cannot be assigned to other open PODs
        then
35:       Set $CostSavingPOD_j = 0$.
36:     end if
37:   end for
38:   Determine the POD to be closed :
      $PODToBeClosed = \text{argmax}_{j \in CO^{POD}} CostSavingPOD_j$
      $MaxCostSaving = \max_{j \in CO^{POD}} CostSavingPOD_j$.
39:   if $MaxCostSaving > 0$ then
40:     Close POD $PODToBeClosed$ and assign its demand to the PODs de-
        termined while calculating the cost saving.
```

```
41:         Update set of open PODs ($CO^{POD}$).
42:       end if
43:     until $MaxCostSaving \leq 0$
44:     if k==1 then
45:       $FO^{POD} = CO^{POD}$
46:     end if
47:     Let $D_j^{POD}$ denote the amount of demand assigned to POD $j$ for all $j \in CO^{POD}$.
48:     Assign all the demand of the PODs to the major facilities as the capacity permits.
49:     $CO^{MF}$ : Set of major facilities that have been assigned a demand.
50:     $MF_{ji}$ : Demand amount of POD $j$ assigned to major facility $i$ for all $i \in N_2, j \in CO^{POD}$.
51:     Update the total serving cost of each major facility that has a demand assigned to it :
        $TotalServingCostMF_i = F_i + \sum_{j \in CO^{POD}} MF_{ji}(d_{ij}c_u^2 + UnitServingCostMF_i)$
52:     repeat
53:       for $i \in CO^{MF}$ do
54:         if $i \in PO^{MF} \bigcup FO^{MF}$ then
55:           $CostSavingMF_i = TotalServingCostMF_i$
56:         else
57:           $CostSavingMF_i = TotalServingCostMF_i + f_i + g_i$
58:         end if
59:       end for
60:       for $i \in CO^{MF}$ do
61:         repeat
62:           Find the closest major facility, say $i'$, with an assigned demand.
```

```
63:               Assign the demand of major facility $i$ to POD $i'$ as the capacity permits. Let $MF_{ji}^{i'}$ be the demand of POD $j$ that has been reassigned to POD $i'$ instead of POD $i$.
64:               Update the cost saving :
                  $CostSavingMF_i = CostSavingMF_i - \sum_{j \in CO^{POD}} MF_{ji}^{i'}(d_{ij}c_u^2 + UnitServingCostMF_{i'})$
65:             until All the demand of major facility $i$ can be assigned to other open major facilities
66:             if All the demand of major facility $i$ cannot be assigned to other open major facilities then
67:               Set $CostSavingMF_i = 0$.
68:             end if
69:           end for
70:           Determine the major facility to be closed :
              $MFToBeClosed = \text{argmax}_{i \in CO^{MF}} CostSavingMF_i$
              $MaxCostSaving = \max_{i \in CO^{MF}} CostSavingMF_i$.
71:           if $MaxCostSaving > 0$ then
72:             Close major facility $MFToBeClosed$ and assign its demand to the major facilities determined while calculating the cost saving.
73:             Update set of open major facilities ($CO^{MF}$).
74:           end if
75:         until $MaxCostSaving \leq 0$
76:         if k==1 then
77:           $FO^{MF} = CO^{MF}$
78:         end if
79:       end for
80:       $CO^{MF}$ and $CO^{POD}$ are the set of major facilities and PODs that will be open
```

```
        in period $t$.
81: end for
```

A.10 Performance of the Heuristics

Table 26: Experiments to test the performance of the heuristics with CPU time in seconds and gap compared to the best lower bound

Instance	Algorithm	Shipment Cost					
		Low		Medium		High	
		CPU	GAP	CPU	GAP	CPU	GAP
71 Demand Nodes	MH	0.4	4.73%	0.4	3.61%	0.4	4.18%
36 Distribution Centers	ADH	0.5	3.84%	0.5	3.23%	0.5	3.20%
5 Major Facilities	SPH	26.1	1.50%	10.5	1.87%	6.5	1.43%
10 Supply Points	HH	6.6	1.83%	4.4	1.48%	3.1	1.58%
	Best Integer Solution	15301.9	0.00%	389.8	0.00%	79.8	0.00%
167 Demand Nodes	MH	0.9	5.17%	1.0	3.87%	1.0	2.38%
36 Distribution Centers	ADH	1.4	5.28%	1.4	1.80%	1.4	0.88%
5 Major Facilities	SPH	78.8	3.47%	27.8	1.16%	14.1	0.34%
10 Supply Points	HH	19.2	4.40%	9.8	1.20%	6.0	0.34%
	Best Integer Solution	24406.2	1.76%	429.5	0.00%	83.5	0.00%
167 Demand Nodes	MH	0.9	12.91%	0.9	4.85%	1.0	3.43%
36 Distribution Centers	ADH	1.4	11.58%	1.4	2.51%	1.4	2.76%
10 Major Facilities	SPH	140.7	8.98%	36.4	1.32%	23.9	1.03%
10 Supply Points	HH	27.1	8.55%	12.7	1.58%	8.2	1.06%
	Best Integer Solution	28290.4	7.15%	3807.3	0.00%	719.5	0.00%
167 Demand Nodes	MH	1.0	7.17%	1.0	3.05%	1.0	2.20%
41 Distribution Centers	ADH	1.5	7.45%	1.5	1.83%	1.5	1.30%
5 Major Facilities	SPH	113.7	7.45%	37.4	1.00%	17.3	0.52%
10 Supply Points	HH	23.6	7.92%	11.5	1.19%	7.2	0.47%
	Best Integer Solution	28767.0	3.97%	1240.8	0.00%	116.5	0.00%
167 Demand Nodes	MH	0.9	6.76%	1.0	3.42%	1.0	2.58%
36 Distribution Centers	ADH	1.4	6.50%	1.4	1.36%	1.4	1.34%
5 Major Facilities	SPH	85.4	5.24%	27.3	1.26%	13.6	0.50%
15 Supply Points	HH	18.6	5.40%	9.2	0.94%	6.7	0.50%
	Best Integer Solution	21893.1	3.14%	785.5	0.00%	119.0	0.00%
167 Demand Nodes	MH	1.6	14.77%	1.6	6.61%	1.7	3.46%
72 Distribution Centers	ADH	2.6	14.31%	2.4	4.26%	2.4	2.11%
10 Major Facilities	SPH	1116.7	9.22%	440.2	3.02%	116.8	1.31%
20 Supply Points	HH	77.4	10.97%	42.0	3.16%	23.8	1.23%
	Best Integer Solution	28853.4	10.70%	28852.9	2.25%	26014.5	0.20%
603 Demand Nodes	MH	17.0	14.12%	16.7	5.35%	18.3	3.41%
151 Distribution Centers	ADH	24.2	13.11%	24.8	3.54%	23.5	2.27%
10 Major Facilities	SPH	21611.4	9.75%	17360.4	3.17%	12976.9	1.69%
20 Supply Points	HH	8475.1	10.44%	3817.1	2.80%	3845.1	1.73%
	Best Integer Solution	43251.7	45.44%	32755.3	2.33%	14423.8	1.38%

A.11 *Effect of Withdrawal Rate on the Disease Spread*

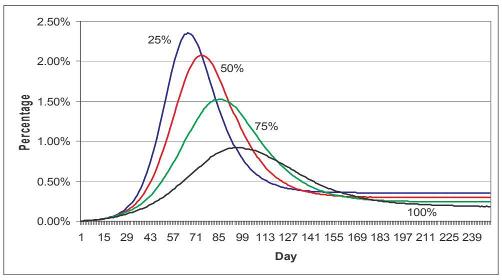
Table 27: Effect of different withdrawal rates from work on the disease spread for $R_0 = 1.5$

Withdrawal Rate	Peak Infectivity	Peak Day	IAR
25%	3.14%	66	54.94%
50%	2.27%	74	47.14%
75%	1.44%	86	37.99%
100%	0.78%	95	28.90%

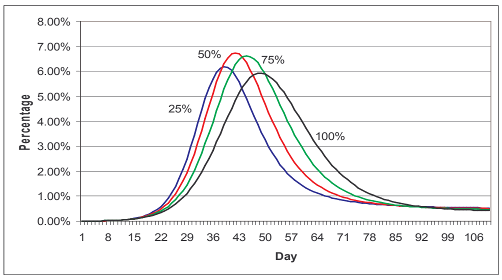
Table 28: Effect of different withdrawal rates from work on the disease spread for $R_0 = 2.1$

Withdrawal Rate	Peak Infectivity	Peak Day	IAR
25%	8.63%	39	80.40%
50%	7.64%	42	77.05%
75%	6.39%	45	71.88%
100%	5.08%	49	65.47%

A.12 *Effect of Withdrawal Rate on the Work Absenteeism*



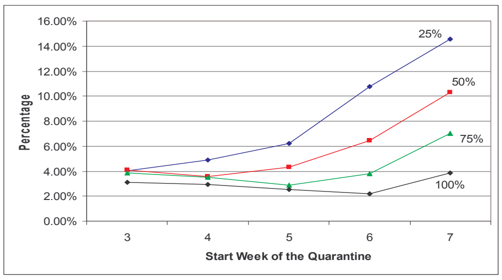
(a) $R_0 = 1.5$



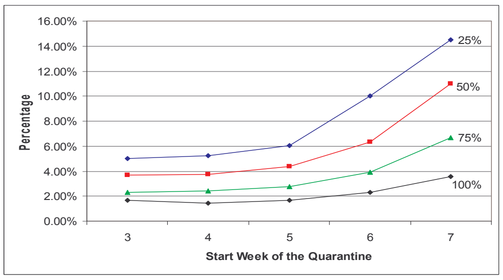
(b) $R_0 = 2.1$

Figure 24: Effect of withdrawal rate on the work absenteeism for different R_0 values

A.13 Effect of Quarantine on the Work Absenteeism



(a) 4-week quarantine



(b) 12-week quarantine

Figure 25: Effect of withdrawal rate and quarantine start time on the peak work absenteeism for $R_0 = 1.8$

APPENDIX B

VARIATIONS OF THE HEURISTICS PROPOSED FOR JOB SPLITTING PROBLEM

B.1 Variations of MRH

1. MRH1 : This is the basic heuristic as explained in Section 3.5.1.

2. MRH2 : While applying MRH1, consider the longest run (L_1) and the number of slots assigned to each type, say y_i, in that run. Assign y_i slots to type i and update its demand by subtracting $\min\{y_i L_1, d'_i\}$ from its demand (where d'_i is the updated demand) and fix this run. Update the remaining demand for all the types. Iteratively apply MRH2 to the updated demand by reducing the number of runs, r, by 1 until all the demands are satisfied.

3. MRH3 : While applying MRH1, consider the longest run (L_1). First assign the types with demand equal to L_1, if such a type exists, to a slot in this run. Then, go to (a).

   (a) Consider the type with the largest demand, say l. Then, assign a slot to type l and update its demand by subtracting $\min\{d'_l, L_1\}$. Then, considering the updated demand return to (a). Continue this until all the slots of this run are assigned to a type.

   Fix this largest run and apply MRH3 to the updated demand with one less run.

4. MRH4 : This is a variation of MRH3. While applying MRH3, after fixing the longest run, check whether it is possible to extend this run. Check the remaining demand of the types assigned to a slot in this largest run. Divide each of these

remaining quantities by y_i (the number of the slots assigned to type i in this run) and calculate the minimum among these, say e_1. Then, update the length of this largest run to $L_1 + e_1$ and update the remaining quantities of the types that are assigned to a slot in this run.

5. MRH5 : In MRH4, to increase the possibility of extending the run, first assign the types with demand greater than L_1, not the ones equal to L_1 as it is done in MRH3 and MRH4.

6. MRH6 : In MRH3, first consider the types that are already assigned to the largest run in the heuristic. Assign L_1 units of demand from these types to the slots as long as the slots are filled. That is, if the remaining demand is less than L_1 for a type, do not assign it to this run. Then, for the remaining slots, consider the remaining updated demand. Start from the type with the largest demand, say i. Assign it to a slot and update its demand by subtracting $\min\{L_1, d'_i\}$ where d'_i is the updated demand. Continue until all the slots of this run are assigned.

7. MRH7 : While applying MRH6, if the number of remaining types at any step becomes larger than the number of remaining slots (remaining runs times m), then directly produce the remaining demand without any splitting. (This will guarantee that the number of runs at the beginning, r, will be equal to the one found at the end.)

8. MRH8 : In MRH7, first start by assigning the types with demand greater than L_1, not the ones equal to L_1. Then, if necessary, consider the ones with demand less than or equal to L_1.

B.2 Variations of BCH

1. BCH1: This is the basic heuristic as explained in Section 3.5.2.

2. BCH2: In BCH1, after fixing a run, if there are types with remaining demand larger than the quantity assigned to a slot in this run, change the assignment of this slot to the type with a larger demand to minimize the waste. In addition, for the case where $m > n$ if the cost of producing the remaining types, if any, in a single run is greater than the cost of satisfying their demands in the previous run, if possible, then produce them in this last run without making an extra run.

3. BCH3: In BCH2, after the slots that are assigned less quantity are replaced with the larger ones, try to increase the length with no extra waste. That is, after the replacement step, there is a new assignment of the types to slots. Do the quantity assignment to slots as it is done in MRH1 (Assign $\lceil \frac{d_i}{y_i} \rceil$ or $\lceil \frac{d_i}{y_i} \rceil - 1$ to a slot that is assigned to type i where y_i is the number of slots in this run that are assigned to type i). Then, if the smallest quantity assigned to a slot is larger than the length that has been set at the beginning, let this smallest amount be the length of this run. Otherwise, leave the length same as set at the beginning. (This will, possibly, increase the length of the run without increasing the waste).

4. BCH4: Similar to BCH3 with one exception. Apply the replacement idea in BCH2 to the single run case ($n < m$), as well.

5. BCH5: In BCH4, try to increase the length of the single run instance after the replacement step. That is, apply the idea in BCH3 also to the single run instance.

6. BCH6: In BCH5, after the replacement step, the assignment of the slots changes. Do the quantity assignments to slots as it is done in MRH1. Order the slots according to the nonincreasing order of quantities assigned to them and apply the basic idea in BCH1 again to find the final length of this run.

REFERENCES

[1] Alves, C. and Valerio de Carvalho, J. M., "A branch-and-price-and-cut algorithm for the pattern minimization problem," *RAIRO - Operations Research*, vol. 42, pp. 435–453, 2008.

[2] American Red Cross Metropolitan Atlanta Chapter, "Pandemic influenza preparedness and response plan (draft)." 2006.

[3] American Red Cross Metropolitan Atlanta Chapter, "Food planning for pandemic flu project plan (draft)." 2008.

[4] Baker, B. S., Brown, D. J., and Katseff, H. B., "A 5/4 algorithm for two dimensional packing," *Journal of Algorithms*, vol. 2, pp. 348–368, 1981.

[5] Baker, B. S., Jr., E. G. C., and Rivest, R. L., "Orthogonal packing in two dimensions," *SIAM Journal on Computing*, vol. 9, pp. 846–855, 1980.

[6] Ballou, R. H., "Dynamic warehouse location analysis," *Journal of Marketing Research*, vol. 5, pp. 271–276, 1968.

[7] Bass, F. M., "A new product growth model for consumer durables," *Management Science*, vol. 15, pp. 215–227, 1969.

[8] Beasley, J. E., "An exact two-dimensional non-guillotine cutting tree search procedure," *Operations Research*, vol. 33, no. 1, pp. 49–64, 1985.

[9] Bekrar, A., Kacem, I., and Chu, C., "A comparative study of exact algorithms for the two dimensional strip packing problem," *Journal of Industrial and Systems Engineering*, vol. 1, no. 2, pp. 151–170, 2007.

[10] Bettinelli, A., Ceselli, A., and Righini, G., “A branch-and-price algorithm for the two-dimensional level strip packing problem,” *4OR: A Quarterly Journal of Operations Research*, vol. 6, pp. 361–374, 2008.

[11] Bureau of Labor Statistics, U.S. Department of Labor, *Career Guide to Industries, 2006-07 Edition, Printing.* 2006.

[12] Cahill, E., Crandall, R., Rude, L., and Sullivan, A., “Space-time influenza model with demographic, mobility, and vaccine parameters,” in *Proceedings of the 5th Annual Hawaii International Conference on Statistics, Mathematics, and Related Fields*, 2005.

[13] Canel, C., Khumawala, B. M., Law, J., and Loh, A., “An algorithm for the capacitated, multi-commodity multi-period facility location problem,” *Computers and Operations Research*, vol. 28, pp. 411–427, 2001.

[14] Carrat, F., Luong, J., Lao, H., Sall, A., Lajaunie, C., and Wackernagel, H., “A “small-world-like” model for comparing interventions aimed at preventing and controlling influenza pandemics,” *BMC Medicine*, vol. 4, no. 26, 2006.

[15] Cava, M. A., Fay, K. E., Beanlands, H. J., McCay, E. A., and Wignall, R., “The experience of quarantine for individuals affected by SARS in Toronto,” *Public Health Nursing*, vol. 22, no. 5, pp. 398–406, 2005.

[16] Center for Disease Control and Prevention, “National influenza pandemic plan.” 2005.

[17] Chin, T. D. Y., Foley, J. F., Doto, I. L., Gravelle, C. R., and Weston, J., “Morbidity and mortality characteristics of Asian strain influenza,” *Public Health Reports*, vol. 75, pp. 149–158, 1960.

[18] Chowell, G., Ammon, C. E., Hengartner, N. W., and Hyman, J. M., "Transmission dynamics of the great influenza pandemic of 1918 in Geneva, Switzerland: Assessing the effects of hypothetical interventions," *Journal of Theoretical Biology*, vol. 241, pp. 193–204, 2006.

[19] Coffman, E. G., Jr., Garey, M. R., Johnson, D. S., and Tarjan, R. E., "Performance bounds for level-oriented two-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 9, pp. 801–826, 1980.

[20] Cooper, L., "Heuristic methods for location-allocation problems," *SIAM Review*, vol. 6, pp. 37–53, 1964.

[21] Cox, K., "Pandemic influenza planning: Guidelines and information for Georgia public school districts," Technical Report, Georgia Department of Education, 2008.

[22] Das, T. K., Savachkin, A. A., and Zhu, Y., "A large-scale simulation model of pandemic influenza outbreaks for development of dynamic mitigation strategies," *IIE Transactions*, vol. 40, pp. 893–905, 2008.

[23] DeMatteis, J. J. and Mendoza, A. G., "An economic lot-sizing technique," *IBM Systems Journal*, vol. 7, no. 1, pp. 30–36, 1968.

[24] Department of Health, Social Services and Public Safety, "Northern Ireland contingency plan for health response for an influenza pandemic." 2008.

[25] Diegel, A., Montocchio, E., Waiters, E., Schaikwyk, S., and Naidoo, S., "Setup minimising conditions in the trim loss problem," *European Journal of Operational Research*, vol. 95, pp. 631–640, 1996.

[26] Diestel, R., *Graph Theory*. Springer-Verlag, New York, 2000.

[27] Dowsland, K. A. and Dowsland, W. B., "Packing problems," *European Journal of Operational Research*, vol. 56, pp. 2–14, 1992.

[28] Dyckhoff, H., "A typology of cutting and packing problems," *European Journal of Operational Research*, vol. 44, pp. 145–159, 1990.

[29] Erlenkotter, D., "A comparative study of approaches to dynamic location problems," *European Journal of Operational Research*, vol. 6, pp. 133–143, 1981.

[30] Farley, A. A., "The cutting stock problem in the canvas industry," *European Journal of Operational Research*, vol. 44, pp. 239–246, 1990.

[31] Feldman, E., Lehrer, F. A., and Ray, T. L., "Warehouse location under continuous economies of scale," *Management Science*, vol. 12, no. 9, pp. 670–684, 1966.

[32] Ferguson, N. M., Cummings, D. A. T., Cauchemez, S., Fraser, C., Riley, S., Meeyai, A., Iamsirithaworn, S., and Burke, D. S., "Strategies for containing an emerging influenza pandemic in Southeast Asia," *Nature*, vol. 437, pp. 209–214, 2005.

[33] Ferguson, N. M., Cummings, D. A. T., Fraser, C., Cajka, J. C., Cooley, P. C., and Burke, D. S., "Strategies for mitigating an influenza pandemic," *Nature*, vol. 442, pp. 448–452, 2006.

[34] Ferguson, N. M., Keeling, M. J., Edmunds, W. J., Gani, R., Grenfell, B. T., Anderson, R. M., and Leach, S., "Planning for smallpox outbreaks," *Nature*, vol. 425, pp. 681–685, 2003.

[35] Ferguson, N. M., Mallett, S., Jackson, H., Roberts, N., and Ward, P., "A population-dynamic model for evaluating the potential spread of drug-resistant influenza virus infections during community-based use of antivirals," *Journal of Antimicrobial Chemotherapy*, vol. 51, pp. 977–990, 2003.

[36] Flahault, A., Deguen, S., and Valleron, A. J., "A mathematical model for the European spread of influenza," *European Journal of Epidemiology*, vol. 10, pp. 471–474, 1994.

[37] Flahault, A., Vergu, E., Coudeville, L., and Grais, R. F., "Strategies for containing a global influenza pandemic," *Vaccine*, vol. 24, pp. 6751–6755, 2006.

[38] Foerster, H. and Wascher, G., "Pattern reduction in one-dimensional cutting stock problems," *International Journal of Production Research*, vol. 38, no. 7, pp. 1657–1676, 2000.

[39] Fraser, C., Riley, S., Anderson, R. M., and Ferguson, N. M., "Factors that make an infectious disease outbreak controllable," *Proceedings of the National Academy of Sciences*, vol. 101, no. 16, pp. 6146–6151, 2004.

[40] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Fransisco, 1979.

[41] Georgia Accrediting Commission, http://www.coe.uga.edu/gac/standards.html, last accessed on January 15, 2008.

[42] Germann, T. C., Kadau, K., Longini, I. M., and Macken, C. A., "Mitigation strategies for pandemic influenza in the United States," *Proceedings of the National Academy of Sciences*, vol. 103, no. 15, pp. 5935–5940, 2006.

[43] Gibbs, W. W. and Soares, C., "Preparing for a pandemic," *Scientific American*, vol. 293, no. 5, pp. 44–54, 2005.

[44] Gilmore, P. C. and Gomory, R. E., "A linear programming approach to the cutting stock problem," *Operations Research*, vol. 9, pp. 849–859, 1961.

[45] Glass, R. J., Glass, L. M., Beyeler, W. E., and Min, H. J., "Targeted social distancing design for pandemic influenza," *Emerging Infectious Diseases*, vol. 12, no. 11, pp. 1671–1681, 2006.

[46] Glasser, J., Taneri, D., Thompson, W., Chuang, J., Wu, J., Tull, P., and Alexander, J., "Evaluation of targeted influenza vaccination strategies via population modeling," *preprint*, 2007.

[47] Grais, R. F., Ellis, J. H., and Glass, G. E., "Assessing the impact of airline travel on the geographic spread of pandemic influenza," *European Journal of Epidemiology*, vol. 18, no. 11, pp. 1065–1072, 2003.

[48] Grotschel, M., Lovasz, L., and Schrijver, A., "Polynomial algorithms for perfect graphs," *Annals of Discrete Mathematics*, vol. 21, pp. 325–356, 1984.

[49] Haessler, R. W., "Controlling cutting pattern changes in one-dimensional trim problems," *Operations Research*, vol. 23, no. 3, pp. 483–493, 1975.

[50] Haessler, R. W. and Sweeney, P. E., "Cutting stock problems procedures," *European Journal of Operational Research*, vol. 54, pp. 141–150, 1991.

[51] Halloran, M. E. and et al., "Modeling targeted layered containment of an influenza pandemic in the United States," *Proceedings of the National Academy of Sciences*, vol. 105, no. 12, pp. 4639–4644, 2008.

[52] Hashimoto, A. and Stevens, J., "Wire routing by optimizing channel assignment within large apertures," *Proceedings of Design Automation Conference*, pp. 155–169, 1971.

[53] Hawryluck, L., Gold, W. L., Robinson, S., Pogorski, S., Galea, S., and Styra, R., "SARS control and psychological effects of quarantine, Toronto, Canada," *Emerging Infectious Diseases*, vol. 10, no. 7, pp. 1206–1212, 2004.

[54] Heesterbeek, J. A. P., "A brief history of R_0 and a recipe for its calculation," *Acta Biotheoretica*, vol. 50, pp. 189–204, 2002.

[55] Hinojosa, Y., Puerto, J., and Fernández, F. R., "A multiperiod two-echelon multicommodity capacitated plant location problem," *European Journal of Operational Research*, vol. 123, pp. 271–291, 2000.

[56] Hoffbuhr, J., "Utilities prepare for potential pandemic," *American Water Works Association Journal*, vol. 98, no. 6, pp. 48–60, 2006.

[57] Holmes, E. C., Taubenberger, J. K., and Grenfell, B. T., "Heading off an influenza pandemic," *Science*, vol. 309, p. 989, 2005.

[58] Hopper, E. and Turton, B., "A genetic algorithm for a 2D industrial packing problem," *Computers and Industrial Engineering*, vol. 37, pp. 375–378, 1999.

[59] Hopper, E., *Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods.* PhD thesis, Cardiff University, UK, 2000.

[60] Hopper, E. and Turton, B. C. H., "An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem," *European Journal of Operational Research*, vol. 128, pp. 34–57, 2001.

[61] Hopper, E. and Turton, B. C. H., "A review of the application of meta-heuristic algorithms to 2D strip packing problems," *Artificial Intelligence Review*, vol. 16, pp. 257–300, 2001.

[62] Hormozi, A. M. and Khumawala, B. M., "An improved algorithm for solving a multiperiod facility location problem," *IIE Transactions*, vol. 28, pp. 105–114, 1996.

[63] Hsiao, J. J., Tang, C. Y., and Chang, R. S., "An efficient algorithm for finding a maximum weight 2-independent set on interval graphs," *Information Processing Letters*, vol. 43, pp. 229–235, 1992.

[64] http://www.fox11az.com/news/topstories/stories/arizona-20090117-flu-pandemic-prepared.4a61884.html, last accessed on June 17, 2009.

[65] Jakobs, S., "On genetic algorithms for the packing of polygons," *European Journal of Operational Research*, vol. 88, pp. 165–181, 1996.

[66] Jones, D. L., "Coordinated local planning for pandemic influenza." PanFlu Seminar District 8-2 - May 10, 2007.

[67] Kass, N. E., Otto, J., O'Brien, D., Minson, M., "Ethics and severe pandemic influenza: Maintaining essential functions through a fair and considered response," *Biosecurity and Bioterrorism*, vol. 6, no. 3, pp. 227–236, 2008.

[68] Kenyon, C. and Remila, E., "A near-optimal solution to a two-dimensional cutting stock problem," *Mathematics of Operations Research*, vol. 25, pp. 645–656, 2000.

[69] Keskinocak, P., Wu, F., Goodwin, R., Murthy, S., Akkiraju, R., Kumaran, S., and Derebail, A., "Scheduling solutions for the paper industry," *Operations Research*, vol. 50, no. 2, pp. 249–259, 2002.

[70] Keskinocak, P., Shi, P., Swann, J., Lee, B., "Seasonality and viral mutation in an influenza pandemic," *working paper*, 2009.

[71] Kuehn, A. A. and Hamburger, M., "A heuristic program for locating warehouses," *Management Science*, vol. 9, pp. 643–666, 1963.

[72] Larson, R. C., "Simple models of influenza progression within a heterogeneous population," *Operations Research*, vol. 55, no. 3, pp. 399–412, 2007.

[73] Lee, B. Y., Bedford, V. L., Roberts, M. S., and Carley, K. M., "Virtual epidemic in a virtual city: Simulating the spread of influenza in a US metropolitan area," *Translational Research*, vol. 151, no. 6, pp. 275–287, 2008.

[74] Lipsitch, M. and et al., "Transmission dynamics and control of severe acute respiratory syndrome," *Science*, vol. 300, pp. 1966–1970, 2003.

[75] Lodi, A., Martello, S., and Monaci, M., "Two-dimensional packing problems: A survey," *European Journal of Operational Research*, vol. 141, pp. 241–252, 2002.

[76] Lodi, A., Martello, S., and Vigo, D., "Recent advances on two-dimensional bin packing problems," *Discrete Applied Mathematics*, vol. 123, pp. 373–390, 2002.

[77] Lodi, A., Martello, S., and Vigo, D., "Models and bounds for two-dimensional level packing problems," *Journal of Combinatorial Optimization*, vol. 8, pp. 363–379, 2004.

[78] Longini, I. M., Nizam, A., Xu, S., Ungchusak, K., Hanshaoworakul, W., Cummings, D. A. T., and Halloran, M. E., "Containing pandemic influenza at the source," *Science*, vol. 309, pp. 1083–1087, 2005.

[79] Martello, S., Monaci, M., and Vigo, D., “An exact approach to the strip-packing problem,” *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 310–319, 2003.

[80] McDiarmid, C., “Pattern minimisation in cutting stock problems,” *Discrete Applied Mathematics*, vol. 98, pp. 121–130, 1999.

[81] MedHeadlines, http://medheadlines.com/2009/05/04/cdc-rethinking-school-closings-for-swine-flu, last accessed on June 15, 2009.

[82] Meltzer, M. I., Cox, N. J., and Fukuda, K., “The economic impact of pandemic influenza in the United States: Priorities for intervention,” *Emerging Infectious Diseases*, vol. 5, pp. 659–671, 1999.

[83] Mills, C. E., Robins, J. M., and Lipsitch, M., “Transmissibility of 1918 pandemic influenza,” *Nature*, vol. 432, pp. 904–906, 2004.

[84] Morse, S. S., Garwin, R. L., and Olsiewski, P. J., “Next flu pandemic: What to do until the vaccine arrives?,” *Science*, vol. 314, p. 929, 2006.

[85] Narula, S. C. and Ogbu, U. I., “An hierarchal location-allocation problem,” *Omega*, vol. 7, no. 2, pp. 137–143, 1979.

[86] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley and Sons, New York, 1999.

[87] Ohio Department of Health and Ohio Food Industry Foundation, “Pandemic influenza preparedness guide for retail food establishments.” 2006.

[88] Patel, R., Longini, I. M., and Halloran, M. E., “Finding optimal vaccination strategies for pandemic influenza using genetic algorithms,” *Journal of Theoretical Biology*, vol. 234, pp. 201–212, 2005.

[89] Rahmandad, H. and Sterman, J., "Heterogeneity and network structure in the dynamics of diffusion: Comparing agent-based and differential equation models," *Management Science*, vol. 54, no. 10, pp. 998–1014, 2008.

[90] Robelen, E. W., "Swine flu disruption has school officials looking for lessons," *Education Week*, 2009.

[91] Roy, T. J. V. and Erlenkotter, D., "A dual-based procedure for dynamic facility location," *Management Science*, vol. 28, no. 10, pp. 1091–1105, 1982.

[92] Rvachev, L. and Longini, I. M., "A mathematical model for the global spread of influenza," *Mathematical Biosciences*, vol. 75, pp. 3–22, 1985.

[93] Sadique, M. Z., Adams, E. J., and Edmunds, W. J., "Estimating the costs of school closure for mitigating an influenza pandemic," *BMC Public Health*, vol. 8, no. 135, 2008.

[94] Schrijver, A., Lovasz, L., Korte, B., Promel, H. L., and Graham, R. L., *Paths, Flows, and VLSI-Layout.* Springer-Verlag New York, Inc., New Jersey, 1990.

[95] Shulman, A., "An algorithm for solving dynamic capacitated plant location," *Operations Research*, vol. 39, no. 3, pp. 423–436, 1991.

[96] Sleator, D., "A 2.5 times optimal algorithm for packing in two dimensions," *Information Processing Letters*, vol. 10, no. 1, pp. 37–40, 1980.

[97] Smith, D. J., "Predictability and preparedness in influenza control," *Science*, vol. 312, pp. 392–394, 2006.

[98] Spinks, B. and Motorola, Inc., *Introduction to Integrated Circuit Layout.* Prentice-Hall, New Jersey, 1985.

[99] Steinberg, A., "A strip-packing algorithm with absolute performance bound 2," *SIAM Journal on Computing*, vol. 26, no. 2, pp. 401–409, 1997.

[100] Sweeney, D. J. and Tatham, R. L., "An improved long run model for multiple warehouse location," *Management Science*, vol. 22, no. 7, pp. 748–758, 1976.

[101] Szymanski, T. G., "Dogleg channel routing is NP-complete," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 1, pp. 31–41, 1985.

[102] Teghem, J., Pirlot, M., and Antoniadis, C., "Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem," *Journal of Computational and Applied Mathematics*, vol. 64, pp. 91–102, 1995.

[103] Trust for America's Health, "Pandemic flu preparedness: Lessons from the frontlines." 2009.

[104] Umetani, S., Yagiura, M., and Ibaraki, T., "One-dimensional cutting stock problem to minimize the number of different patterns," *European Journal of Operational Research*, vol. 146, pp. 388–402, 2003.

[105] U.S. Census Data, www.census.gov/main/www/cen2000.html, last accessed on January 15, 2008.

[106] Vanderbeck, F., "Exact algorithm for minimising the number of setups in the one-dimensinonal cutting stock problem," *Operations Research*, vol. 48, no. 6, pp. 915–926, 2000.

[107] Viboud, C., Boelle, P., Cauchemez, S., Lavenu, A., Valleron, A.,

Flahault, A., and Carrat, F., "Risk factors of influenza transmission in households," *British Journal of General Practice*, vol. 54, pp. 684–689, 2004.

[108] Wallinga, J., Teunis, P., and Kretzschmar, M., "Using data on social contacts to estimate age-specific transmission parameters for respiratory-spread infectious agents," *American Journal of Epidemiology*, vol. 164, no. 10, pp. 936–944, 2006.

[109] World Health Organization, "Avian influenza: Assessing the pandemic threat." 2005.

[110] World Health Organization Writing Group, "Nonpharmaceutical interventions for pandemic influenza, national and community measures," *Emerging Infectious Diseases*, vol. 12, no. 1, pp. 88–94, 2006.

[111] Wu, J. T., Riley, S., Fraser, C., and Leung, G. M., "Reducing the impact of the next influenza pandemic using household-based public health interventions," *PLoS Medicine*, vol. 3, no. 9, pp. 1532–1540, 2006.

[112] Wu, J. T., Riley, S., and Leung, G. M., "Spatial considerations for the allocation of pre-pandemic influenza vaccination in the United States," *Proceedings of the Royal Society B*, vol. 274, pp. 2811–2817, 2007.

[113] Yanasse, H. H. and Limeira, M. S., "A hybrid heuristic to reduce the number of different patterns in cutting stock problems," *Computers and Operations Research*, vol. 33, pp. 2744–2756, 2006.

[114] Zhang, D., Liu, Y., Chen, S., and Xie, X., "A meta-heuristic algorithm for the strip rectangular packing problem," *Lecture Notes in Computer Science*, vol. 3612, pp. 1235–1241, 2005.

VITA

Ali Ekici was born in Kırşehir, Turkey. He received Bachelor of Science degrees in Industrial Engineering and Mathematics from Middle East Technical University in 2003. After graduation, he started pursuing his Ph.D. in the H. Milton Stewart School of Industrial and Systems Engineering at the Georgia Institute of Technology. He received a Master of Science degree in Operations Research from Georgia Institute of Technology in 2006. His research experience and interests are in the field of emergency response logistics, network design/expansion, dynamic routing, decentralized decision making and industry applications of scheduling/packing.