TOPICS IN PACKING AND SCHEDULING

A Dissertation
Presented to
The Academic Faculty

By

Christopher Muir

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

Georgia Institute of Technology

August 2022

TOPICS IN PACKING AND SCHEDULING

Thesis committee:

Dr. Alejandro Toriello
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Siva Theja Maguluri
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Santanu Dey
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Luke Marhsall
CORE
Microsoft Research

Dr. Mohit Singh
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: July 6th, 2022

To my late grandfather Lance Muir.

ACKNOWLEDGMENTS

There are numerous people who have been instrumental in the production of this thesis and to whom I owe thanks.

My parents, Michael and Cindy Muir, for their boundless love and encouragement. Without their support none of this would have been possible.

My advisor Prof. Alejandro Toriello, for his guidance and the many opportunities he made available for me. I will always be grateful to him for guiding my development as a researcher and letting me explore my academic and career interests.

The faculty of the Georgia Tech School of Industrial and Systems Engineering, for their contribution to my education both in and outside of the classroom. I give special thanks to Profs. Santanu Dey, Mohit Signh, and Siva Theja Maguluri for being part of my thesis committee and for providing valuable feedback.

Drs. Ishai Menache and Luke Marshall, for the opportunity to work with them at Microsoft Research. Not only did this work form the basis for part of this thesis, it was a valuable opportunity to learn about research careers outside of academia. I would like to further thank Dr. Marshall for serving on my thesis committee.

The wonderful friends I have made while at Georgia Tech. Daniel Ulch, Adam Behrendt, Anthony Trasatti, Arden Baxter, Hannah Lagerman, Lacy Greening, Alex Forney, Andrew ElHabr, Hassan Mortagy, and the others I shouldn't have forgotten, my time at Tech would not have been the same without them.

Dr. Jim Ostrowski, for his mentorship over the years; he was my first introduction to academic research and instrumental in my decision to pursue a Ph.D. I also thank the other members of the University of Tennessee Optimization Lab: Ethan Deakins, Dr. Ben Knueven, Dr. Tony Rodriguez, and the many others for their camaraderie and support.

Lastly, the National Science Foundation, for supporting me through their Graduate Research Fellowship program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Packing and scheduling models include some of the most fundamental problems in operations research and computer science. These broad classes include a wide range of models with applications including logistics, production planning, wireless network design, circuit design, and cloud computing, to name a few. In this thesis we study three such models: dynamic node packing, interval scheduling with economies of scale, and temporal bin packing with half-capacity jobs; each extends on a well-known problem in packing and scheduling. While the problems are generally distinct, this research was broadly inspired by applications in cloud computing, specifically by problems cloud service providers face when servicing requests for virtual machines.

In Chapter 2, we propose a dynamic version of the node packing problem. In this model, instead of being given the edges upfront, we model them as independent Bernoulli random variables. At each step, the decision maker selects an available node and then observes edges adjacent to this node. The goal is a policy that maximizes the expected value of the resulting packing. We model the problem as a Markov decision problem and conduct a polyhedral study of the problem's achievable probabilities polytope. We develop a variety of valid inequalities based on paths, cycles, and cliques.

In Chapter 3, we study interval scheduling problems exhibiting economies of scale. An instance is given by a set of interval jobs and a cost function. Specifically, we focus on the max-weight function and non-negative, non-decreasing concave functions of total schedule weight. The goal is a partition of the jobs minimizing the total cost with the constraint that jobs within the same schedule cannot overlap. We propose a set covering formulation and a column generation algorithm to solve its linear relaxation, providing efficient pricing algorithms for the studied cases. To obtain integer solutions, we extend the column generation approach using branch-and-price.

In Chapter 4, we study a different model with interval jobs. In this problem, interval

jobs are partitioned into bins such that at most two jobs in a bin overlap at any given time. The decision maker is tasked with minimizing the time-average number of bins required to pack all jobs. We call this problem temporal bin packing with half-capacity jobs; it is a special case of the general temporal bin packing problem with bounded parallelism. We study the worst-case performance of a well-known static lower bound, and, motivated by this analysis, we introduce a novel lower bound and integer programming formulation based on formulating the problem as a series of matching problems. We provide theoretical guarantees on the relative strengths of the static bound, the matching-based bound, and various linear programming bounds.

CHAPTER 1
INTRODUCTION

The global cloud computing market was valued at nearly $400 billion in 2021 and is projected to continue to grow [1]. This economic growth has led to a corresponding growth in energy consumption; data centers now account for approximately 1% of worldwide energy use [2]. These trends have increased interest in improving energy efficiency, from both an economic and environmental perspective. In a data center, energy consumption is driven primarily by servers, through direct consumption or indirectly through cooling. Energy usage in certain systems can be reduced by improving server utilization. Furthermore, by improving server utilization, cloud service providers also increase the volume of requests they can service. Optimizing server utilization can be framed in many ways, but frequently these problems focus on managing of requests for virtual machines. Virtual machines need to be assigned to servers while respecting the servers' capacities; typically, the goal is to minimize the number of active servers, as this is directly proportional to energy consumption and increases the volume of usable capacity, but more complex objectives are also applicable. The problem is further complicated by the need to incorporate additional practical considerations, such as uncertainty.

Virtual machine allocation and its related problems can often be modelled using variants of well-known packing and scheduling problems. In this thesis we address three such problems: dynamic node packing, interval scheduling with economies of scale, and temporal bin packing with half-capacity jobs.

In Chapter 2, we study dynamic node packing, a dynamic variant of the classical node packing problem, also called the stable set or independent set problem. The problem is defined by a node set, a node weight vector, and an edge probability vector. For every pair of nodes, an edge is present or not according to an independent Bernoulli random variable

defined by the corresponding entry in the probability vector. At each step, the decision maker selects an available node that maximizes the expected weight of the final packing, and then observes edges adjacent to this node. We formulate the problem as a Markov decision problem and show that it is NP-Hard even on star graphs. Next, we introduce relaxations of the problem's achievable probabilities polytope, analogous to the linear and bilinear edge-based formulations in the deterministic case; we show that these relaxations can be weak, motivating a polyhedral study. We derive classes of valid inequalities arising from cliques, paths, and cycles. For cliques, we completely characterize the polytope and show that it is a submodular polyhedron. For both paths and cycles, we give an implicit representation of the polytope via a cut-generating linear program of polynomial size, based on a compact dynamic programming formulation. Our computational results show that our inequalities can greatly reduce the basic linear relaxation's gap, particularly when the instance's expected density is high.

In Chapters 3 and 4 we consider two different problems using interval jobs, which are defined by a fixed interval over which the job is present. Specifically, an interval job i is specified by some start time s_i and end time e_i, with $s_i < e_i$. Two jobs i, j overlap if $[s_i, e_i) \cap [s_j, e_j) \neq \emptyset$. Given a set of interval jobs J, a *conflict graph* is created by adding a node for each job in J and an edge between two nodes if their corresponding jobs overlap. A conflict graph constructed in this manner is an interval graph. Interval graphs have many useful properties that we utilize in Chapters 3 and 4, including the following.

1. Interval graphs are *perfect*, implying their coloring number is equal to the size of the graph's maximum clique.

2. An interval graph with n nodes has $O(n)$ maximal cliques, and they can be determined in $O(n)$ time.

3. The clique-node incidence matrix of an interval graph satisfies the consecutive ones property and is therefore totally unimodular (TU).

Among other things, these properties imply that on interval graphs the minimum coloring, maximum clique, and maximum node packing problems can be solved in polynomial time with relatively simple algorithms. We discuss these results in more detail in Chapters 3 and 4.

In Chapter 3, we study interval scheduling problems exhibiting economies of scale. An instance is given by a set of interval jobs and a function representing the cost of scheduling a subset of jobs on the same machine. Specifically, we focus on the max-weight function and non-negative, non-decreasing concave functions of total schedule weight. The goal is a partition of the jobs that minimizes the total schedule cost, where overlapping jobs cannot be processed on the same machine. We propose a set cover formulation and a column generation algorithm to solve its linear relaxation. For the max-weight function, which is already NP-hard, we give a polynomial-time pricing algorithm; for the more general case of a function of the total weight, we have a pseudo-polynomial algorithm. To obtain integer solutions, we extend the column generation approach using branch-and-price. We computationally evaluate our methods on two different functions, using both random instances and instances derived from cloud computing data; our algorithm significantly outperforms known integer programming formulations (when these are available) and is able to provably optimize instances with hundreds of jobs.

In Chapter 4, we consider a different interval scheduling problem known as temporal bin packing with half-capacity jobs. In this problem, interval jobs are partitioned into bins with the constraint that bins can fit at most two jobs simultaneously. The objective is to determine an assignment of jobs to bins that minimizes the time-average number of active bins; this problem is known to be NP-Hard. We demonstrate that a well-known static lower bound may have a significant gap even in relatively simple instances. Motivated by this analysis, we introduce a novel lower bound and an integer programming formulation, both based on an interpretation of the problem as a series of connected matching problems. The strengths of the static bound, the new matching-based bound, and various linear program-

ming bounds are compared in detail. We computationally evaluate our methods using both synthetic and application-based instances. We demonstrate that our formulation is able to improve on a previously known formulation; we also study the empirical performance of our proposed bounds, showing they are able to improve on the static bound, particularly when the instances are sparse.

Finally, in Chapter 5, we conclude and summarize some potential research directions.

CHAPTER 2
DYNAMIC NODE PACKING

This chapter is adapted from the upcoming publication [3].

2.1 Introduction

The node packing problem, also known as the stable set or independent set problem, is a fundamental model in combinatorial optimization. Taking an undirected graph as input, the decision maker seeks the most valuable subset of pairwise non-adjacent nodes, either in terms of weight or cardinality. Node packing is well known to be equivalent to both the maximum clique and minimum vertex cover problems, and is also related to other classical problems, such as vertex coloring and set packing.

Node packing has received much attention from the optimization, operations research, graph theory, and computer science communities, in part due to its fundamental nature, but also owing to its varied applications. One such important application area is scheduling [4, 5]. In the variant of single-machine job scheduling where each job has an associated time interval(s) in which it must be performed if accepted, the decision maker may construct a graph in which job-interval pairs are nodes, and nodes are adjacent when the corresponding job-interval pairs are conflicting, i.e., when they overlap in time or job. This idea is generalized by resource-constrained project scheduling models, where jobs require multiple resources and have more complex interactions, such as precedence requirements; an example of this type of model appears in [6].

Another important application area for node packing is wireless network design and operation. In large-scale wireless networks, it is often desirable to cluster nodes in order to facilitate more efficient communication. This is done by selecting *cluster heads* that drive the communication in and out of a cluster. Ideally, every node in a network should belong

to a cluster and cluster heads should be out of each other's immediate range. The problem of optimally selecting cluster heads can be modeled as a weighted node packing problem [7]. Another example in wireless network design stems from selecting RFID readers to activate in a given network, as readers can interfere with each other if they both try to read from the same target. The problem of selecting which readers to activate at a given time to maximize the total number of read targets can be modeled as a weighted node packing problem [8].

Additional application examples include computer vision [9], train routing [10], coding theory [11], and military planning [12]. For a further discussion of applications, we refer the reader to [13, 14] and the references within.

In our proposed model, each node pair is associated with a probability that the corresponding edge appears in the graph. Nodes are selected dynamically, and after the decision maker commits to selecting a node, each potential edge between the chosen node and other nodes is sampled independently as a Bernoulli random variable with its corresponding probability. The decision maker's goal is to maximize the expected cardinality or weight of the resulting packing. This model generalizes classical deterministic node packing in two ways: It incorporates probabilistic graph topology akin to the Erdős-Rényi random graph model, and it introduces aspects of dynamic decision making.

One of our primary motivations for studying this model comes from dynamic scheduling problems arising in areas such as cloud computing. Consider a single-machine scheduling scenario in which jobs may have, for instance, known start times but random processing times that are only revealed once the job is scheduled. Therefore, we cannot construct an entire conflict graph beforehand, but we do have some probabilistic knowledge of possible edges present in the graph. This scheduling problem and others like it can be modeled using our dynamic node packing problem with additional modifications, such as allowing correlations between edges, and requiring the nodes to be considered in a particular order. As a first step, in this chapter we consider a more fundamental model that does not have

additional features.

Our contributions can be summarized as follows.

1. We propose a dynamic node packing model that incorporates uncertainty in the edge set; the model is new, to our knowledge. We formulate the model as a Markov decision process and study its theoretical difficulty; in particular, it is NP-Hard even on star graphs.

2. We introduce basic relaxations of the model analogous to the standard linear and bilinear edge-based formulations of the deterministic node packing problem, and demonstrate that both relaxations can scale linearly with node count. Motivated by this negative result, we perform a polyhedral study of the dynamic node packing polytope and derive valid inequalities arising from cliques, paths and cycles.

3. We perform a computational study of the dynamic node packing problem, demonstrating the empirical effectiveness of our proposed inequalities. As a secondary contribution, our empirical results also reveal the remarkably good performance of a probabilistic and weighted generalization of the minimum-degree ordering heuristic.

The remainder of the chapter is organized as follows. The next section includes a brief literature review. Section 2.3 introduces and formulates the problem, and gives preliminary results. Section 2.4 then includes our polyhedral study, with Section 2.5 detailing our computational study. Finally, Section 2.6 concludes.

2.2 Literature Review

The deterministic node packing problem has been studied extensively; in addition to being one of the most well-known NP-Hard problems [15], it is also hard to approximate within a factor of $n^{1-\epsilon}$ [16]. Numerous algorithms have been proposed for solving the node packing problem: Current leading exact algorithms are primarily based on branch-and-bound, such as those compared in [17]. The methods generally differ based on how the upper- and

lower-bounds are computed during the branching process. Heuristics are also often employed to solve the problem approximately; despite the negative approximability results, simple heuristics are observed to perform quite well in practice [18].

The node packing polytope has also been studied extensively as a way to derive polyhedral relaxations and bounds. Given a graph $G = (N, E)$ and associated weight vector $w \in \mathbb{R}^N_+$, the standard edge formulation for the node packing problem is

$$\begin{aligned} \max \ & \sum_{i \in N} w_i x_i & \\ \text{s.t. } & x_i + x_j \leq 1 & \forall \{i,j\} \in E \\ & x_i \in \{0,1\} & \forall i \in N. \end{aligned}$$

The linear relaxation of this formulation is half-integral, meaning the extreme points of the corresponding polytope take on values in $\{0, 1/2, 1\}$. Furthermore, for the cardinality case, in any extreme point optimal solution of the relaxation, any variable taking value one also appears in at least one optimal solution [19]. This edge formulation yields a weak linear relaxation in general, only capturing the integer hull in the case of bipartite graphs. Many polyhedral results for the node packing problem utilize specific types of graph structures to derive valid inequalities that strengthen the linear relaxation. Early results in this vein provided important classes of valid inequalities arising from cliques, odd holes, and odd antiholes [20, 21]. Additional examples include grilles [22], webs [23], and wheels [24].

The node packing problem may also be formulated as a bilinear programming problem,

$$\begin{aligned} \max \ & \sum_{i \in N} w_i x_i & \\ \text{s.t. } & x_i x_j = 0 & \forall \{i,j\} \in E \\ & 0 \leq x_i \leq 1 & \forall i \in V. \end{aligned}$$

This model is commonly then relaxed to a semidefinite programming formulation. The value of this relaxation is equivalent to the Lováz ϑ function [25], which can then be used to solve the problem exactly in the case of perfect graphs [26]. Claw-free graphs are another special case where the problem is polynomially solvable [27]. For further information regarding the node packing polyhedron and node packing algorithms, we refer the reader to [28, 29, 30] and the references within.

Combinatorial optimization in the presence of uncertainty has received much attention in recent years, and many different ways of capturing uncertainty have been proposed. The models most closely related to ours are those in which some input parameters are only known probabilistically, and we are interested in static or adaptive algorithms that optimize the expected total reward. Examples of problems in which this type of paradigm has been studied include scheduling [31], equipment replacement [32], knapsack [33, 34, 35, 36], traveling salesman and vehicle routing [37, 38], and general packing [39]. These problems may be framed as Markov decision processes; for a general reference we refer the reader to [40].

Often in these models the curse of dimensionality makes exact methods intractable, motivating heuristics and bounding methods collectively known as approximate dynamic programming. The methods falling under this umbrella term vary significantly; the ones most relevant to our purposes include the study of polyhedra of achievable probabilities, e.g., [41, 42, 43], and approximate linear programming techniques, e.g., [44, 33, 45].

To the best of our knowledge, a dynamic model of this type has not been previously introduced for node packing. However, an alternative way of modeling uncertainty is via online models, where the decision maker optimizes against an adversarial input sequence. In particular, an online node packing model was proposed in [46]: Nodes are presented sequentially, and the decision maker must either irrevocably admit the node into the packing or permanently reject it. When a node is presented, it includes potential adjacencies to any previously admitted node. Traditional worst-case analysis shows that any deterministic

algorithm in this setting has a competitive ratio of $1/(n-1)$. To address this, various relaxations and alternative models of the the online model have been proposed, including multi-solution models [46], stochastic input models [47], models with advice [48, 49], and known graph models [50].

Another area of relevant research is node packing in random graphs. In the Erdős-Rényi random graph $G(n, 1/2)$, the largest node packing is almost surely in $\{2\log_2(n) - 1, 2\log_2(n)\}$, and a greedy algorithm almost surely finds a node packing of size $\log_2(n)$ [51]. Furthermore, the ϑ function is a loose bound on random graphs, almost surely giving a value of $\Theta(\sqrt{n})$ in $G(n, 1/2)$ [52]. In fact, the stronger Lovász-Schrijver hierarchy is also weak in random graphs; for $G(n, 1/2)$, after r iterations of the hierarchy, the value of the relaxation is almost surely $\Theta(\sqrt{n/2^r})$, and the hierarchy almost surely requires $\Theta(\log n)$ iterations for a tight bound [53].

2.3 Problem Statement and Preliminary Results

The *dynamic node packing* (DNP) problem is defined by a node set $N = \{1, \ldots, n\}$, probability vector $p \in [0, 1]^{\binom{n}{2}}$, and node weight vector $w \in \mathbb{R}^N_+$. Each p_{ij} represents the probability that an edge is present between nodes $i, j \in N$; for convenience, we denote the complementary probability as $q_{ij} = 1 - p_{ij}$. When p_{ij} is constant across all pairs, we recover the Erdős-Rényi random graph model (we discuss this special case in detail below and in Section 2.4).

The decision maker constructs a packing sequentially, one node at a time. Upon selection of a node i, for every other remaining node $j \neq i$, the edge $\{i, j\}$ appears according to an independent Bernoulli random variable with probability p_{ij}. After the potential edges are sampled, the neighbor set of i, $\Gamma(i)$, is known exactly; the decision maker collects i's weight w_i and can no longer add any node $j \in \Gamma(i)$ to the packing. The process continues, with the decision maker selecting an available node until no eligible nodes remain. The objective is a policy that maximizes the expected total weight of the final packing. DNP

may be formulated as an n-stage Markov decision process (MDP) defined on states $S \subseteq N$, with initial state N, and defined by the following recursion and boundary condition:

$$v_S^* = \max_{i \in S} \{ w_i + \mathbb{E}\big[v^*_{S \setminus (i \cup \Gamma(i))} \big] \}, \qquad \emptyset \neq S \subseteq N \tag{2.1a}$$

$$v_\emptyset^* = 0. \tag{2.1b}$$

The transition probabilities from state S to state $S^{'}$, given selection of node $i \in S$, are calculated as

$$\mathbb{P}(S'|S,i) = \begin{cases} \prod_{j \in S'} q_{ij} \prod_{j \in S \setminus (i \cup S')} p_{ij} & S' \subseteq S \setminus i \\ 0 & S' \not\subseteq S \setminus i. \end{cases}$$

In other words, the probability of transitioning from state S to S' after selecting node i is the probability of edges between i and nodes in $S \setminus (i \cup S')$ realizing, and of edges between i and nodes in S' not realizing.

The *value* linear program (LP) associated with the above recursion is then

$$\min \ v_N \tag{2.2a}$$

$$\text{s.t. } v_S - \mathbb{E}\big[v_{S \setminus (i \cup \Gamma(i))} \big] \geq w_i \qquad \forall \emptyset \neq S \subseteq N, \forall i \in S \tag{2.2b}$$

$$v_\emptyset \geq 0. \tag{2.2c}$$

The value LP provides a strong dual to the MDP. Specifically, the value v_N^* given by (2.1) is the optimal objective of (2.2), and any feasible solution provides an upper-bound on v_N^*.

The dual of (2.2) is known as the *policy* LP. Any feasible solution to the policy LP implies a policy in the MDP, with each variable corresponding to a state-action pair that can be interpreted as the probability of reaching that state and then selecting the corresponding

action. The policy LP for the DNP is formulated as

$$\max \sum_{S \subseteq N} \sum_{i \in S} w_i x_{S,i} \tag{2.3a}$$

$$\text{s.t.} \sum_{i \in N} x_{N,i} = 1 \tag{2.3b}$$

$$\sum_{i \in S} x_{S,i} = \sum_{S' \supsetneq S} \sum_{j \in S' \setminus S} \mathbb{P}(S|S', j) x_{S',j} \qquad \forall \emptyset \neq S \subsetneq N \tag{2.3c}$$

$$x_{S,i} \geq 0 \qquad \forall S \subseteq N, \forall i \in S. \tag{2.3d}$$

As there are 2^n possible subsets of N, this LP contains $\Theta(n2^n)$ variables and $\Theta(2^n)$ constraints.

2.3.1 DNP on Star Graphs

In the general case, DNP is NP-Hard, as deterministic node packing is a special case in which all edge probabilities are binary. However, there are many cases in which the deterministic node packing problem is not only easy, but trivial. One such case is star graphs; a star graph is a tree of depth 1, a graph with one central root node (which we denote as 0 here) and n leaves connected only to the root. In the deterministic model, the decision maker may only select the root node if they do not select any leaf node and vice versa. The optimal objective value is then obviously $\max\{w_0, \sum_{i=1}^{n} w_i\}$, with the optimal solution being either to take the center node or to take all the leaf nodes.

Surprisingly, even on star graphs the DNP is NP-Hard; we prove this next. For simplicity of notation, in this section we use N to represent the leaf nodes only (and not the root), and denote the edges of the star graph by their corresponding leaf node, so that $\{0, i\}$ is simply i.

Observation 2.3.1. *Solving DNP on star graphs is equivalent to selecting the optimal subset $S \subseteq N$ of leaves to add to the packing before trying to add the root node 0.*

In other words, the DNP on star graphs reduces to a static subset selection problem. This observation stems from two facts: First, if the decision maker commits to taking a subset of leaf nodes before the root, the order does not matter. Second, whenever the root node is added to the packing or is covered by one of its leaf nodes, any uncovered leaf may be added to the packing. This observation leads to the following formulation of the DNP on star graphs,

$$\max_{S\subseteq N}\left\{\sum_{i\in S} w_i + \left(1-\prod_{i\in S} q_i\right)\sum_{j\notin S} w_j + \prod_{i\in S} q_i\left(w_0 + \sum_{j\notin S} q_j w_j\right)\right\}. \tag{2.4}$$

The first term represents the immediate value of nodes added to the packing before trying to add the root node; the second term represents the probability-weighted value of the scenario in which at least one of the nodes in S covers the root node; and the third term represents the probability-weighted value of the root node being successfully added to the packing and then trying to add the leaf nodes in $N \setminus S$.

Theorem 2.3.2. *The DNP on star-graphs is NP-Hard.*

Proof. First, by rearranging and applying a natural logarithm, (2.4) is equivalent to

$$\max_{S\subseteq N}\left\{\sum_{i\in S} \ln q_i + \ln\left(w_0 - \sum_{k\in N} p_k w_k + \sum_{i\in S} p_i w_i\right)\right\}. \tag{2.5}$$

We show (2.5) is NP-Hard via a reduction from the partitioning problem. Given a set of numbers N with positive weights a_i, $i \in N$, the partitioning problem asks for a subset $S \subseteq N$ such that $\sum_{i\in S} a_i = \sum_{j\notin S} a_j$; without loss of generality, we assume that $\sum_{i\in N} a_i = 2$. By setting parameters for (2.5) as $q_i = e^{-a_i}$, $p_i = 1 - e^{-a_i}$, $w_i = a_i/(1-e^{-a_i})$, and $w_0 = \sum_{i\in N} a_i$, (2.5) becomes

$$\max_{S\subseteq N} -\sum_{i\in S} a_i + \ln(\sum_{i\in S} a_i).$$

Proceeding similarly to [54], a set $S \subseteq N$ with $\sum_{i\in S} a_i = 1$ exists if and only if the optimal

solution to (2.5) is -1. □

This result is surprising in two ways. First, as mentioned before, node packing on star graphs is trivial in the deterministic case, but by simply adding uncertainty in the edge set the problem becomes NP-Hard. Second, the DNP on star graphs loses the dynamism of the general case; the order in which leaves are added to the packing only matters with respect to whether they are selected before attempting to add the root.

Our proof relies on a reduction from the partition problem, and thus only establishes that the DNP on star graphs is weakly NP-hard. We next sketch a pseudo-polynomial-time dynamic programming algorithm. In (2.5), we can interpret the inclusion of node i in S as paying an immediate cost of $-\ln q_i$ (since $\ln q_i$ is non-positive), and then receiving a terminal reward depending on the final value of $\sum_{i\in S} p_i w_i$. Assuming $w \in \mathbb{Z}^{N\cup 0}$ and that the probabilities are rational, let K be a scaling factor such that $Kp_iw_i \in \mathbb{Z}$ for $i \in N$; note that K is bounded above by the least common multiple of the denominators defining the p_i's. We define states as (i, W) for $i = 1, \ldots, n+1$ and $W = 0, 1, \ldots, K\sum_{i\in N} p_i w_i$, where state (i, W) denotes the decision maker currently considering node i with $K\sum_{j\in S} p_j w_j = W$. The initial state is $(1, 0)$; letting $\sigma_{1,0}$ represent its optimal value, we get the recursion

$$\sigma_{i,W} = \max\{\ln q_i + \sigma_{i+1,W+Kp_iw_i}, \sigma_{i+1,W}\}$$
$$\sigma_{n+1,W} = \ln\left(w_0 - \sum_{k\in N} p_k w_k + W/K\right).$$

The recursion can be solved in $O(nK\sum_{i\in N} p_i w_i)$ time, and is therefore polynomial for instances where $K\sum_{i\in N} p_i w_i$ is polynomial in n; this holds, for example, if $K\max_i\{p_i w_i\}$ is a constant. Furthermore, because of the logarithm function, the recursion assumes exact real arithmetic; in practice this can be replaced with a numerical approximation of sufficient precision.

As a final note on the hardness result, the proof relies on varying both node weights and edge probabilities. If either the node weights or probabilities are uniform, the problem

is polynomially solvable. If weights are uniform (the cardinality case), a trivial optimal solution is to take all leaf nodes before the root. If probabilities are uniform, suppose leaves are ordered by non-increasing value of weight, $w_1 \geq \cdots \geq w_n$. A simple exchange argument shows that an optimal set is of the form $\{1, \ldots, i\}$ for some $i \in N$; therefore, we only need to consider the $n + 1$ sets of this form and select the one maximizing (2.4).

2.3.2 Relaxations of the DNP Polytope

The DNP can be solved using standard MDP techniques such as backwards induction or the value and policy LP's [40]. However, these exact approaches become intractable for even small instance sizes. Additionally, since DNP is NP-Hard for stars, it is unlikely efficient algorithms exist for any instance containing stars as induced subgraphs. One alternative is to consider relaxations of the DNP that may be computed efficiently and provide an upper bound on the true objective. Specifically, we are interested in relaxations of the polytope obtained by projecting the policy LP (2.3) into the space of *achievable* node probabilities, where each variable represents the probability of ever selecting the node. Denote this polytope by Q, and define it as

$$Q := \left\{ z \in \mathbb{R}^N : \exists x \text{ satisfying (2.3b)–(2.3d) such that } z_i = \sum_{S \ni i} x_{S,i} \right\}.$$

That is, we consider relaxations of the problem

$$\max_{z \in Q} \sum_{i \in N} w_i z_i. \tag{2.6}$$

Note that Q is full-dimensional in $\mathbb{R}^N$. A simple relaxation of (2.6) is to consider a probabilistic version of the standard edge formulation of node packing. Given two nodes i, j, the probability of selecting both of them is at most q_{ij}. Using this, we arrive at the

relaxation

$$\max \sum_{i \in N} w_i z_i \tag{2.7a}$$

$$\text{s.t. } z_i + z_j \leq 1 + q_{ij} \qquad \forall i, j \in N \tag{2.7b}$$

$$0 \leq z_i \leq 1 \qquad \forall i \in N. \tag{2.7c}$$

Many of the inequalities included in this relaxation are necessary to describe Q. The non-negativity constraint in (2.7c) is facet-defining for Q for every $i \in N$, while the upper bound is facet-defining as long as $p_{ij} < 1$ for $j \in N \setminus i$. Similarly, for any pair of nodes i, j with $p_{ij} > 0$, constraints (2.7b) are facet-defining for Q as long as no $k \in N \setminus \{i, j\}$ has $p_{ik} = p_{jk} = 1$.

As in deterministic node packing, this relaxation is quite loose. For example, consider the cardinality DNP on the random graph $G(n, 1/2)$. A feasible solution for (2.7) sets $z_i = 3/4$, $i \in N$, which results in an objective value of $3n/4$. In other words, the objective scales linearly with n.

Unfortunately, unlike the deterministic case, passing to a bilinear relaxation is not much better. The relaxation analogous to the deterministic bilinear formulation is

$$\max \sum_{i \in N} w_i z_i \tag{2.8a}$$

$$\text{s.t. } z_i z_j \leq q_{ij} \qquad \forall i, j \in N \tag{2.8b}$$

$$0 \leq z_i \leq 1 \qquad \forall i \in N. \tag{2.8c}$$

The formulation 2.8 is not a valid relaxation for all instances, but it is valid for the $G(n, p)$ model and this restricted case suffices to demonstrate the point. Using the same instance as before, a feasible solution sets $z_i = 1/\sqrt{2}$, $i \in N$, resulting in an objective value of $n/\sqrt{2} \approx 0.71n$, hardly improving on the linear relaxation; meanwhile, the largest node

packing in $G(n, 1/2)$ is $\Theta(\log n)$ almost surely [51], and this quantity upper bounds the DNP since it allows the decision maker to observe the entire graph before choosing a packing. We discuss the DNP's optimal value for this instance in more detail in the next section.

The problem with both of these relaxations is that they only capture the probability relationships between pairs of nodes. It is possible to create tighter relaxations by introducing valid inequalities derived from larger structures within the instance; that is our goal.

2.4 Polyhedral Study

In this section we perform a polyhedral study of the polytope of achievable probabilities for the DNP. These inequalities can be added to the linear relaxation (2.7) to create a stronger upper bound. Specifically, we focus on two types of inequalities, those arising from cliques with uniform probabilities, and those arising from paths and cycles.

2.4.1 Cliques with Uniform Probabilities

A clique is a subset of pairwise adjacent nodes within a graph; we are interested here in the probabilistic analogue, node subsets for which every pair i, j has the same edge probability $p \in (0, 1)$, and call them a *probabilistic clique*. To differentiate such substructures that may appear in a larger instance from entire instances with this structure, we use K_n^p for the former, and keep the random graph notation $G(n, p)$ for the latter. We begin by showing that the DNP on $G(n, p)$ is efficiently solvable via a simple greedy policy.

Proposition 2.4.1. *The policy of selecting a highest-weight remaining node is optimal for DNP on* $G(n, p)$.

Proof. Take some optimal policy π and consider the first time it does not choose a highest-weight remaining node. Let S be the subset of remaining nodes, i be the node the policy selects, and j be a node with the highest remaining weight. We can write the expected

value of the current state S as

$$w_i + \sum_{k \in S \setminus \{i\}} \mathbb{P}_\pi(k) w_k,$$

where $\mathbb{P}_\pi(k)$ is the probability of getting to pick node k under π. Now consider the policy π' that only differs from π in that we swap nodes j and i, selecting node j now and node i at any point where we would have selected j. The value of π' at subset S then becomes

$$w_j + \sum_{k \in S \setminus \{j\}} \mathbb{P}_{\pi'}(k) w_k.$$

By construction $\mathbb{P}_\pi(j) = \mathbb{P}_{\pi'}(i)$, the difference in value between the two policies at subset S is

$$w_j - w_i + \mathbb{P}_{\pi'}(i)(w_i - w_j) > 0.$$

So we can create a better policy by modifying π as described, a contradiction. □

Although straightforward, this result allows us to derive an efficient recursive formula for the expected value of the DNP on probability cliques. For the remainder of this section, we denote the expected value of the policy on $G(n,p)$ as $v_{n,p}$ in the cardinality case (where any available node is chosen, since $w_i = 1$ for all $i \in N$) , and $v^w_{n,p}$ in the weighted case. We also assume that nodes are labeled in non-increasing order of weight, so $w_1 \geq w_2 \geq \cdots \geq w_n$.

Proposition 2.4.2. *The expected value of the DNP on $G(n,p)$ in the cardinality case is defined recursively as*

$$v_{n,p} = 1 + \sum_{i=0}^{n-1} \binom{n-1}{i} p^i q^{n-1-i} v_{n-1-i,p},$$

where $q = 1 - p$, $v_{0,p} = 0$ and $v_{1,p} = 1$.

Proof. In the cardinality case, all nodes are equivalent up to relabeling. As such, it suffices

to define the states of our recursion based on the number of nodes in the resulting subgraph, which at state n is a binomial random variable with parameters $n-1$ and p. □

We can equivalently write $v_{n,p} = 1 + \mathbb{P}(2) + \cdots + \mathbb{P}(n)$, where again $\mathbb{P}(i)$ is the probability of adding i to the packing. Therefore, $v_{i,p} - v_{i-1,p} = \mathbb{P}(i)$; that is, the probability of adding i can be expressed as a difference in two expected values. With this we can compute the expected value of the weighted DNP using the following result.

Theorem 2.4.3. *For $G(n,p)$ and weight vector $w \in \mathbb{R}^N_+$ satisfying $w_1 \geq w_2 \geq \ldots \geq w_n$,*

$$v^w_{n,p} = w_1 + w_2(v_{2,p} - v_{1,p}) + \cdots + w_n(v_{n,p} - v_{n-1,p}).$$

Proof. It follows from Proposition 2.4.1 that we only need to consider the policy of selecting nodes in non-increasing weight order. We may then write the expected value as

$$v^w_{n,p} = w_1 + w_2\mathbb{P}(2) + \cdots + w_n\mathbb{P}(n).$$

Then by substituting $\mathbb{P}(i)$ with $v_{i,p} - v_{i-1,p}$ we arrive at our result. □

Theorem 2.4.3 and Proposition 2.4.2 allow for the direct computation of the optimal objective value of the DNP on probabilistic cliques. Proposition 2.4.2 also implies a class of valid inequalities for Q. For $G(n,p)$ and $S \subseteq N$, we obtain the inequality

$$\sum_{i \in S} z_i \leq v_{|S|,p}. \tag{2.9}$$

In addition to being valid for the clique itself, (2.9) is valid in the general case, when K^p_n is embedded in a larger instance, as the left side of the inequality is maximized when a policy attempts to add every node in the clique to the packing first. Having established their validity, a natural subsequent question concerns the facial dimension of the constraints.

Lemma 2.4.4. *For an instance given by $G(n,p)$,* (2.9) *is facet-defining for Q, for any $\emptyset \neq S \subseteq N$.*

Proof. Suppose first that $S = N$, and consider the matrix of points

$$\begin{bmatrix} 1 & (v_{2,p} - v_{1,p}) & (v_{3,p} - v_{2,p}) & \cdots & (v_{n,p} - v_{n-1,p}) \\ (v_{n,p} - v_{n-1,p}) & 1 & (v_{2,p} - v_{1,p}) & \cdots & (v_{n-1,p} - v_{n-2,p}) \\ (v_{n-1,p} - v_{n-2,p}) & (v_{n,p} - v_{n-1,p}) & 1 & \cdots & (v_{n-2,p} - v_{n-3,p}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (v_{2,p} - v_{1,p}) & (v_{3,p} - v_{2,p}) & (v_{4,p} - v_{3,p}) & \cdots & 1 \end{bmatrix}.$$

The rows correspond to the n cyclic shifts of the selection ordering $(1, 2, \ldots, n)$ and each satisfies (2.9) at equality. The above matrix is a circulant matrix, each entry is positive, and the sequence $(1, (v_{2,p} - v_{1,p}), (v_{3,p} - v_{2,p}), ..., (v_{n,p} - v_{n-1,p}))$ is monotonically decreasing. It follows from [55, Proposition 24] that the matrix is non-singular, i.e., we have n affinely independent points. As Q is full-dimensional, the result follows.

For $S \subsetneq N$, relabel nodes so that members of S appear before other nodes. We can construct an analogous $|S| \times |S|$ matrix by applying the policy only to nodes in S; we can then append zero entries for columns corresponding to $N \setminus S$ to get $|S|$ points satisfying (2.9) at equality. For the remaining $n - |S|$ rows, apply the policy corresponding to the ordering $(1, 2, \ldots, |S|)$ and then choose $i \in N \setminus S$ if it's still available. This creates a block-lower-triangular matrix; we have already argued above that the upper-left block is non-singular, and subsequent diagonal blocks are all of size one with positive entries equal to $(v_{|S|+1,p} - v_{|S|,p})$, thus the entire matrix is also non-singular, and the result follows. □

In addition to being valid in the general case when K_n^p appears as a substructure of a larger instance, the inequalities remain facet-defining under mild conditions.

Theorem 2.4.5. *Suppose a DNP instance contains K_n^p, and let S be its node set. Constraint* (2.9) *is facet-defining if and only if no node in $N \setminus S$ is adjacent to every node in S with*

probability 1.

Proof. The proof is analogous to that of Lemma 2.4.4. For nodes $i \in N \setminus S$, we proceed as in the second part of the proof: Without loss of generality, suppose $p_{1i} < 1$. (Otherwise, permute nodes in S so the first one satisfies this property.) Then after applying the policy on S, i is still available with some positive probability, and thus the matrix is again non-singular. For the reverse direction, if some $i \in N \setminus S$ is connected to all nodes in S with probability 1, any point in Q that satisfies (2.9) at equality must have $z_i = 0$. □

Finally, we show that these inequalities are sufficient to describe the polytope of achievable probabilities.

Theorem 2.4.6. *For $G(n, p)$, Q is given by constraints* (2.9) *for $\emptyset \neq S \subseteq N$ and non-negativity constraints.*

Proof. We proceed by arguing that for any weight vector $w \in \mathbb{R}^N$, the optimal value is the consequence of an upper bound resulting from a conic combination of the mentioned constraints. First, we argue that we only need to consider weight vectors with positive entries. If there is an $w_i \leq 0$, in any optimal solution the we can set $z_i = 0$ and consider the problem in a lower dimension. So assume $w > 0$ and $w_1 \geq w_2 \geq \ldots \geq w_n$. Let z^* be optimal; the optimal expected value is

$$\begin{aligned}\sum_{i=1}^{n} w_i z_i^* &= w_n \sum_{i=1}^{n} z_i^* + (w_{n-1} - w_n) \sum_{i=1}^{n-1} z_i^* + \ldots + (w_1 - w_2) z_1^* \\ &\leq w_n v_{n,p} + (w_{n-1} - w_n) v_{n-1,p} + \ldots + (w_1 - w_2) v_{1,p} \\ &= w_1 + (v_{2,p} - v_{1,p}) w_2 + \ldots + (v_{n,p} - v_{n-1,p}) w_n,\end{aligned}$$

where the inequality follows from applying constraints (2.9) to each summand. Theorem 2.4.1 then implies that this inequality holds at equality for an optimal solution. □

Corollary 2.4.7. *For $G(n, p)$, Q is a submodular polyhedron. Constraints* (2.9) *can be separated in polynomial time via a greedy separation routine.*

Proof. The expected value $v_{n,p}$ is concave in n, and therefore $v_{|S|,p}$ is submodular in S. Given any $z \in [0,1]^N$, we can separate over constraints (2.9) by ordering z's coordinates in non-increasing order: Suppose we relabel nodes so that $z_1 \geq z_2 \geq \cdots \geq z_n$; then it suffices to check (2.9) for sets $\{1\}, \{1,2\}, \ldots, N$. □

In some instances, it may be unlikely to find a large clique with *uniform* probabilities, reducing the effectiveness of constraints (2.9). This can be partially remedied by relaxing the constraints: Given a clique with non-uniform probabilities, a valid constraint of form (2.9) can be generated using the minimum probability among any pair in the clique.

Naturally, there are other valid inequalities for the non-uniform case as well; the next section treats one such case.

Triangles

We next derive valid inequalities for triangles with possibly non-uniform edge probabilities. We denote the three nodes comprising the triangle as i, j, k and without loss of generality assume $p_{ij} \geq p_{jk} \geq p_{ik} > 0$. For triangles, any policy reduces to a selection ordering; furthermore, the policy's expected cardinality is in fact entirely determined by the first node selection, as we show next.

Proposition 2.4.8. *For triangles, a policy's expected cardinality is determined by the first node choice.*

Proof. Consider policies in which i is selected first. There are two orders in this case, i, j, k and i, k, j. The expected cardinalities of these selection orders are respectively

$$1 + q_{ij} + q_{ik}(p_{ij} + q_{ij}q_{jk}) = 1 + q_{ij} + p_{ij} - p_{ij}p_{ik} + q_{ij}q_{jk}q_{ik} = 2 - p_{ij}p_{ik} + q_{ij}q_{jk}q_{ik},$$

and

$$1 + q_{ik} + q_{ij}(p_{ik} + q_{ik}q_{jk}) = 1 + q_{ik} + p_{ik} - p_{ij}p_{ik} + q_{ij}q_{jk}q_{ik} = 2 - p_{ij}p_{ik} + q_{ij}q_{jk}q_{ik}.$$

That is, the expected size of the packing is the same for both orderings. □

This result implies that the following inequality is valid for any policy in which $z_i = 1$:

$$z_j + z_k \leq 1 - p_{ij}p_{ik} + q_{ij}q_{jk}q_{ik}.$$

We may combine this with $z_i = 1$ to rewrite the inequality as

$$z_i + (z_j + z_k)/\alpha_i \leq 2, \tag{2.10}$$

where $\alpha_i = 1 - p_{ij}p_{ik} + q_{ij}q_{ik}q_{jk}$. Inequality (2.10) is not necessarily valid for policies in which node i is not selected first. To create an inequality that is valid for the entire polytope, we lift the z_i variable to create a new inequality of the following form

$$\beta_i z_i + (z_j + z_k)/\alpha_i \leq 1 + \beta_i. \tag{2.11}$$

We need a β_i in (2.11) satisfying

$$\beta_i \geq \frac{(z_j + z_k)/\alpha_i - 1}{1 - z_i} \tag{2.12}$$

for all feasible values of z_i, z_j, z_k.

Proposition 2.4.9. *Suppose* $N = \{i, j, k\}$. *Inequality* (2.11) *is valid for* Q *when*

$$\begin{aligned}\beta_i = \max\Bigg\{ &\frac{(1 + q_{jk})/\alpha_i - 1}{1 - q_{ij}(p_{jk} + q_{jk}q_{ik})}, \frac{(1 + q_{jk})/\alpha_i - 1}{1 - q_{ik}(p_{jk} + q_{jk}q_{ij})}, \\ &\frac{(1 + q_{jk}(p_{ik} + q_{ik}q_{ij}))/\alpha_i - 1}{(1 - q_{ik})}, \frac{(1 + q_{jk}(p_{ij} + q_{ij}q_{ik}))/\alpha_i - 1}{(1 - q_{ij})} \Bigg\}.\end{aligned} \tag{2.13}$$

Proof. We argue the inequality is valid for all orders. For orders in which i is selected first, the inequality is valid by construction. There are four possible orderings in which i is not selected first. Consider the two orderings in which node j is selected first; these

result in the two points $(q_{ij}(p_{jk} + q_{jk}q_{ik}), 1, q_{jk})$ and $(q_{ij}, 1, q_{jk}(p_{ij} + q_{ij}q_{ik}))$. Similarly, the orderings in which k is selected first result in the points $(q_{ik}(p_{jk} + q_{ij}q_{jk}), q_{jk}, 1)$ and $(q_{ik}, q_{jk}(p_{ik} + q_{ij}q_{ik}), 1)$. Each of the four terms in (2.13) correspond to evaluating (2.12) at one of the four points. Therefore, taking the maximum over the four points results in a valid lifted inequality. □

A constraint of the form (2.11) can be generated for each node in the triangle, corresponding to that node being selected first. Note that if $p_{ij} = p_{jk} = p_{ik}$, this inequality is equivalent the corresponding clique inequality (2.9).

Proposition 2.4.10. *Constraints* (2.11) *are facet-defining for Q on triangles.*

Proof. It is easy to check that the inequality $z_i + (z_j + z_k)/\alpha_i \leq 2$ is facet-defining for the intersection of Q and the hyperplane defined by $z_i = 1$. The result then follows because we have applied maximal lifting to obtain β_i. □

Finally, we show how these constraints define Q on triangles, when added to the linear relaxation (2.7).

Theorem 2.4.11. *The constraints* (2.11) *generated for each first node choice i, j, k, along with* (2.7b) *and* (2.7c)*, fully describe Q on triangles.*

Proof. We show how to construct the extreme point corresponding to the selection order i, j, k using the given constraints. The probability vector corresponding to this selection order is

$$(1, q_{ij}, q_{ki}(p_{ij} + q_{ij}q_{jk})).$$

Consider the constraints

$$z_i \leq 1, \qquad z_i + z_j \leq 1 + q_{ij}, \qquad \beta_i z_i + (z_j + z_k)/\alpha_i \leq 1 + \beta_i.$$

Taking all three at equality, the first two imply $z_i = 1$ and $z_j = q_{ij}$. Substituting these into the third equation yields

$$\begin{aligned} z_k &= \alpha_i - q_{ij} = 1 - p_{ij}p_{ik} + q_{ij}q_{ik}q_{jk} - q_{ij} \\ &= p_{ij}p_{ik} - p_{ij} + q_{ij}q_{ik}q_{jk} = p_{ij}q_{ik} + q_{ij}q_{ik}q_{jk} = q_{ik}(p_{ij} + q_{ij}q_{jk}). \end{aligned}$$

The argument follows similarly for the other orderings.

For the extreme points in which zero, one, or two nodes are selected with non-zero probability, e.g., $(0, 0, 0)$, $(1, 0, 0)$, $(1, q_{ij}, 0)$, it is simple to verify that they are already extreme points of the polytope defined by only (2.7b) and (2.7c). □

2.4.2 Paths

In our context, a (probabilistic) path is a non-repeating sequence of nodes where each node is adjacent to the node that precedes it with positive probability. In this section we analyze the DNP on paths and use it to derive a cut-generating LP based on paths for the general case. We also extend the approach to cycles below.

Given an n-node path with nodes labeled in the path's order, $1, 2, \ldots, n$, we denote the optimal expected value of the DNP on the path as $u_{1,n}$. Similarly, for any i-j sub-path, $i \leq j$, we use $u_{i,j}$. Because there are only quadratically many sub-paths, the recursion (2.1) simplifies to

$$u_{i,j} = \max_{i \leq k \leq j} \{ w_k + p_{k-1,k}u_{i,k-2} + q_{k-1,k}u_{i,k-1} + p_{k,k+1}u_{k+2,j} + q_{k,k+1}u_{k+1,j} \} \qquad 1 \leq i \leq j \leq n \tag{2.14a}$$

$$u_{i,j} = 0 \qquad i > j. \tag{2.14b}$$

In the recursion, we consider which node in a path to choose; this necessarily cuts the path in two, but it may do so in several ways, depending on whether the node's edges realize or

not.

Recursion (2.14) runs in $O(n^3)$ time, as there are $O(n^2)$ sub-paths and we consider $O(n)$ node choices for each. As a consequence, we can efficiently solve the LP formulation

$$\begin{aligned}
&\min\ u_{1,n} \\
&\text{s.t. } u_{i,j} - p_{k-1,k}u_{i,k-2} - q_{k-1,k}u_{i,k-1} \\
&\qquad\qquad - p_{k,k+1}u_{k+2,j} - q_{k,k+1}u_{k+1,j} \geq w_k && 1 \leq i \leq k \leq j \leq n \\
&u_{i,j} = 0 && i > j,
\end{aligned}$$

and its dual,

$$\max \sum_{1\leq i\leq k\leq j\leq n} w_k x^k_{i,j} \tag{2.15a}$$

$$\text{s.t. } \sum_{k=1}^{n} x^k_{1,n} = 1 \tag{2.15b}$$

$$\begin{aligned}
\sum_{k=i}^{j} x^k_{i,j} = &\sum_{\ell=1}^{i-1} q_{i-1,i} x^{i-1}_{\ell,j} + \sum_{\ell=1}^{i-2} p_{i-2,i-1} x^{i-2}_{\ell,j} \\
&+ \sum_{\ell=j+1}^{n} q_{j,j+1} x^{j+1}_{i,\ell} + \sum_{\ell=j+2}^{n} p_{j+1,j+2} x^{j+2}_{i,\ell}
\end{aligned} \qquad 1 \leq i \leq j \leq n \tag{2.15c}$$

$$x^k_{i,j} \geq 0 \qquad \forall 1 \leq i \leq k \leq j \leq n. \tag{2.15d}$$

In (2.15), $x^k_{i,j}$ represents the probability of encountering the i-j path and choosing node k.

As with cliques, our main goal is not to solve the DNP on paths, but rather to derive valid inequalities we can apply to the general problem. We summarize this in the next result.

Theorem 2.4.12. *Let $\hat{z} \in [0,1]^N$. Then $\hat{z} \in Q$ for the 1-n path if and only if the following*

LP has optimal value equal to zero:

$$\min\ u_{1,n} - \sum_{i=1}^{n} \lambda_i \hat{z}_i \tag{2.16a}$$

$$\text{s.t. } u_{i,j} - p_{k-1,k} u_{i,k-2} - q_{k-1,k} u_{i,k-1} - p_{k,k+1} u_{k+2,j} - q_{k,k+1} u_{k+1,j} \geq \lambda_k \qquad \forall 1 \leq i \leq k \leq j \leq n \tag{2.16b}$$

$$u_{i,j} = 0 \qquad \forall i > j. \tag{2.16c}$$

If the LP is unbounded, any ray $(\hat{\lambda}, \hat{u})$ *with negative objective value generates the valid inequality*

$$\sum_{i=1}^{n} \hat{\lambda}_i z_i \leq \hat{u}_{1,n},$$

which cuts off $\hat{z}$*.*

Proof. As in the general case (2.6), $\hat{z}$ is an achievable probability if some $\hat{x}$ feasible in (2.15) satisfies

$$\hat{z}_k = \sum_{\substack{i \leq k \\ j \geq k}} \hat{x}^k_{i,j}, \qquad k \in N.$$

Applying Farkas' lemma to this equation and the feasible region of (2.15) yields (2.16). □

In the general DNP, for any fixed path we may iteratively improve the linear relaxation using (2.16) to generate cutting planes. Next, we discuss extending this approach to cycles.

Cycles

The probabilistic path concept naturally extends to cycles, and our approach can also capture Q for cycles in a cut-generating LP. Extending our path notation, we now suppose we have a probabilistic cycle with nodes $1, \ldots, n$, where i is connected to $i + 1$ with positive probability and 1 is connected to n with positive probability as well. To hopefully ease the notational burden in our exposition, we adopt in this section the convention that all indices use modular arithmetic; for example, 0 is identified with n and $n + 1$ is identified with 1.

Note also that we address the special case of triangles above; we are able to describe Q for triangles explicitly, while here we give an implicit description that applies to any cycle. Therefore, we assume without loss of generality that $n \geq 4$.

The recursion (2.14) can be extended to cycles. After the decision maker selects a first node, the resulting subgraph is in fact a path, and therefore after the first decision the recursion proceeds exactly as in that case. The only difference is that this path may have a starting node with a higher index than its end node, so we adopt modular arithmetic in our notation. Define the optimal expected value of the cycle as u_N, and extend our previous notation to $u_{i,j}$ for any $i, j \in N$. In particular, if $i \leq j$, this corresponds to the path $(i, i+1, \ldots, j)$, whereas if $i > j$, the variable corresponds to the path $(i, i+1, \ldots, n, 1, 2, \ldots, j)$. We then obtain the recursion

$$
\begin{aligned}
u_N = \max_{i \in N} \{ & w_i + p_{i-1,i} p_{i,i+1} u_{i+2,i-2} + q_{i-1,i} p_{i,i+1} u_{i+2,i-1} \\
& + p_{i-1,i} q_{i,i+1} u_{i+1,i-2} + q_{i-1,i} q_{i,i+1} u_{i+1,i-1} \} \\
u_{i,j} = \max_{i \leq k \leq j} \{ & w_k + \mathbb{1}_{k \geq i+2} p_{k-1,k} u_{i,k-2} + \mathbb{1}_{k \geq i+1} q_{k-1,k} u_{i,k-1} \\
& + \mathbb{1}_{k \leq j-2} p_{k,k+1} u_{k+2,j} + \mathbb{1}_{k \leq j-1} q_{k,k+1} u_{k+1,j} \} \qquad , \qquad i \neq j \\
u_{i,i} = w_i, \quad & i \in N.
\end{aligned}
$$

We use indicator functions to avoid notational overlap, but otherwise the recursion is similar to (2.14), with the exception of u_N. The complexity remains $O(n^3)$, as we have only doubled the number of paths. From this recursion, we can proceed exactly as in the path case to derive a cut-generating LP to use in the general case, when cycles appear as substructures. Moreover, although this LP doubles the number of variables, we significantly increase the number of cutting planes we can generate, since a single cycle of length n captures n different paths with n nodes.

2.5 Computational Study

The primary objective of our computational experiments is to evaluate the effectiveness of inequalities derived in our polyhedral study, in terms of their bound improvement over the basic linear relaxation (2.7). To benchmark these various bounds, we also study a probabilistic extension of the *minimum degree ordering heuristic*; see, e.g., [18]. At any state $S \subseteq N$, the heuristic chooses

$$\operatorname*{argmax}_{i \in S} \left\{ w_i - \sum_{j \in S \setminus i} p_{ij} w_j \right\}.$$

In other words, the heuristic adds a node to the packing that maximizes the immediate net expected value, where the positive element of the value is the chosen node's weight, while the negative element is the probabilistically discounted value of the node's possible neighbors. In the cardinality case, this reduces to choosing the node with the minimum expected degree in S, and further reduces precisely to the minimum degree ordering in the deterministic case. We refer to this heuristic as the *max expected weight heuristic* (MEWH) in our experiments.

For instances in which all non-zero probabilities are uniform, we also define a dual-based value function approximation heuristic. Specifically, the heuristic approximates the expectation in the value function (2.1) by the dual prices of probabilistic clique constraints at an optimal solution of (2.7) strengthened with constraints (2.9). Let $\mathcal{C}$ be the set of probability cliques and y be the corresponding vector of dual prices; at any state $S \subseteq N$ we select a node according to

$$\operatorname*{argmax}_{i \in S} \left\{ w_i + \sum_{C \in \mathcal{C}} y_C \mathbb{E}\left[v_{|C \cap (S \setminus (i \cup \Gamma(i)))|, p} \right] \right\},$$

where p again denotes the uniform edge probability and $v_{n,p}$ is defined in Proposition 2.4.2.

Each expectation is calculated as

$$\mathbb{E}\left[v_{|C\cap(S\setminus(i\cup\Gamma(i)))|,p}\right] = \begin{cases} v_{|C\cap S|,p} - 1 & i \in C \\ \sum_{j=0}^{|\gamma(i)\cap C\cap S|} \binom{|\gamma(i)\cap C\cap S|}{j} p^j (1-p)^{|\gamma(i)\cap C\cap S|-j} v_{|C\cap S|-j,p} & i \notin C, \end{cases}$$

where $\gamma(i) \subseteq N \setminus i$ denotes nodes adjacent to i with probability p. Recall that although $\mathcal{C}$ may contain an exponential number of probabilistic cliques, an optimal extreme point solution of the LP has at most n corresponding constraints with non-zero dual prices. This method can be viewed as maximizing the immediate reward of the chosen node's weight and the expectation of the sum of dual prices multiplied by the expected size of a packing in the appropriate probability clique. The multipliers $v_{|C\cap S|,p}$ scale dynamically as nodes are packed and covered. We refer to this heuristic as the *dual clique heuristic* (DCH). In addition to providing another benchmark, the DCH allows us to test our tightened relaxation's potential to guide heuristic policies.

As a final benchmark, we also consider the expected value of an instance's max-weight node packing, the generalization of the expected max-cardinality packing in random graphs. Unlike the DNP, this benchmark allows the decision maker to observe the entire graph's realization before selecting a packing. In stochastic programming terms, we allow the decision maker to violate non-anticipativity by giving them earlier access to information. Equivalently, it is the "hindsight-optimal" value, what the decision maker would have liked to do with the benefit of hindsight; therefore, we refer to it as HSO below. Whereas our polyhedral bounds are computed via LP, the MEWH, DCH, and HSO must be approximated via simulation.

2.5.1 Instances

We conduct experiments using two types of instances designed to evaluate the performance of our bounds and benchmarks on sparser and denser instances. For dense instances, we use

random graphs to generate a topology. Specifically, to generate each instance, we sample a random graph from $G(100, p_1)$, and assign each realized edge the uniform probability p_2, where $p_1, p_2 \in \{0.5, 0.75, 0.9\}$. For each of the nine value pairs of p_1 and p_2, we generate ten instances in this manner, all with the cardinality objective, yielding 90 total instances, each with 100 nodes. Note that the effect of the two probability parameters is multiplicative: In an instance's *realization*, the probability that any node pair will have an edge is $p_1 p_2$, which ranges between 0.25 and 0.81.

For sparse instances, we use binary trees. Each instance's topology is given by a full binary tree of depth six, which has $2^6 = 64$ leaves and $2^7 - 1 = 127$ total nodes. We assign node weights uniformly at random from $[1, 10]$, and each edge in the tree has uniform edge probability $p \in \{0.5, 0.75, 0.9\}$. For each value of p, we generate ten instances that only vary in their node weights, for a total of 30 instances.

2.5.2 Experiments

For dense instances, we test four bounds: Relaxation (2.7), (2.7) strengthened by path and cycle cuts, relaxation (2.7) strengthened by probabilistic clique cuts, and (2.7) strengthened by path, cycle, and clique cuts.

For each instance, we generate path and cycle cuts by greedily searching for 200 paths of maximum length ten and then, if possible, connecting the start and end of the path to form a cycle. At each iteration, given a current probability vector, we solve (2.16) for each path and the analogous LP for each cycle, and then re-optimize. We use three stopping criteria: Either we find no new cuts, the absolute change in objective is less than 0.5, or we reach a maximum of 50 iterations. We elected to generate path and cycle cuts in this manner after exploratory tests suggested better bounds came from many shorter paths, rather than a smaller number of longer paths.

To compare probabilistic cliques, for each topology we compute the 5,000 largest maximal probabilistic cliques, resulting in 5,000 initial cuts. We then re-optimize, adding vi-

olated cuts (2.9) for node subsets until we find none, or reach the same alternative termination conditions as with paths and cycles. For the experiments that combined the two inequality classes, we perform both types of cut generation in each each iteration.

For the sparse instances, since their topology is based on binary trees, we can only compare the relaxation (2.7) and (2.7) with path inequalities. However, unlike in the dense instances, we can enumerate all maximal paths (corresponding to all pairs of leaves), and thus add any violated path inequality using (2.16); this is what we do in the experiments until we find no more violated inequalities or, again, the alternate termination criteria are met.

For the simulation-based benchmarks, we calculate each by simulating 100 realizations of the instance and taking sample averages. For MEWH, this simply amounts to running the heuristic on each realization. For DCH, we use the optimal solution to the relaxation (2.7) with probabilistic clique constraints added, and also run the heuristic on each realization. We only test the DCH on dense instances, as the sparse instances have no probabilistic cliques beyond node pairs. For HSO, we solve a deterministic node packing problem on each realization; for dense instances, this entails solving 100 node packing integer programs (IP), while for the sparse instances the realization is guaranteed to be a forest, and thus an LP suffices.

We conduct all experiments on a MacBook with a 2.7GHz Dual-Core Intel i5 processor. Our base code and simulation use Python 3.7.3, and the LP and IP solves use Gurobi 9.0. For all instances, the solution of (2.7) takes less than one second. For dense instances, adding path and cycle inequalities to the bound increases the solve time to between 10 and 20 seconds, while adding probability clique inequalities raises solution times to between five seconds and five minutes, generally increasing with the size of the cliques. For the simulation-based benchmarks, the entire MEWH simulation runs in one or two seconds, the DCH takes approximately one minute per instance, while the HSO simulation takes an average of three minutes per instance. On the sparse instances, adding path inequalities

to (2.7) results in solve times of about three minutes, as the number of paths to check is $\binom{64}{2} = 2,016$.

2.5.3 Summary of Results

Table 2.1 below presents average results for the dense instances. We report results as the geometric mean of the ratio between the corresponding value and the HSO bound over the 10 instances of each type. We choose this presentation because the HSO bound is independent of our proposed methods and such bounds are known to be strong for many dynamic problems.

Table 2.1: Geometric mean and standard deviation of dense instance results as ratios of the HSO bound.

p_1	p_2	(2.7)	(2.7) + P/C	(2.7) + Clq.	(2.7) + P/C + Clq.	MEWH	DCH
0.5	0.5	4.45 ± 0.05	3.93 ± 0.04	2.29 ± 0.01	2.29 ± 0.01	0.83 ± 0.01	0.74 ± 0.01
0.5	0.75	5.18 ± 0.04	4.63 ± 0.04	2.24 ± 0.05	2.24 ± 0.05	0.85 ± 0.01	0.74 ± 0.01
0.5	0.9	5.41 ± 0.05	5.06 ± 0.04	1.98 ± 0.35	1.98 ± 0.33	0.88 ± 0.01	0.73 ± 0.03
0.75	0.5	6.17 ± 0.07	5.41 ± 0.06	2.37 ± 0.02	2.37 ± 0.02	0.79 ± 0.02	0.72 ± 0.02
0.75	0.75	7.69 ± 0.06	6.84 ± 0.06	2.41 ± 0.07	2.41 ± 0.07	0.82 ± 0.01	0.71 ± 0.02
0.75	0.9	8.54 ± 0.13	7.96 ± 0.11	2.27 ± 0.49	2.27 ± 0.47	0.86 ± 0.01	0.71 ± 0.04
0.9	0.5	7.31 ± 0.07	6.40 ± 0.07	2.64 ± 0.02	2.63 ± 0.02	0.75 ± 0.02	0.71 ± 0.01
0.9	0.75	9.66 ± 0.15	8.57 ± 0.14	2.95 ± 0.10	2.94 ± 0.10	0.78 ± 0.02	0.72 ± 0.01
0.9	0.9	11.04 ± 0.09	10.28 ± 0.08	2.98 ± 0.57	2.97 ± 0.56	0.82 ± 0.01	0.70 ± 0.02

From these results, we see that the probabilistic clique cuts are able to greatly improve the bound provided by the relaxation (2.7), particularly so when the expected density is high, where they nearly cut the ratio by 75%. However, their overall performance is better when the expected density is low, which is in line with the performance of linear relaxations of the deterministic node packing problem. Path and cycle cuts do not decrease the bound as much as clique cuts in all of the dense instances. Possible improvements could come from

increasing the number or length of the path and cycle cuts, but there is a corresponding computation time increase, and our preliminary experiments did not show better bounds with longer paths.

Surprisingly, the MEWH performs relatively well across the board, consistently achieving a ratio between 0.75 and 0.88, with the best performance coming when $p_1 = 0.5$ and $p_2 = 0.9$, i.e., when the topology has relatively low density but edges are very likely to realize. The DCH performs worse than the MEWH across all instances, achieving ratios between 0.70 and 0.74. Its performance is fairly consistent among all the instances, in absolute terms performing best when the expected density is low. Interestingly, the DCH maintains its performance better than the MEWH as the expected density increases. This is likely a result of the probabilistic clique constraints increasing in strength as the expected density rises.

Table 2.2 below summarizes results for the sparse instances, following a similar format to Table 2.1. We observe that all the tested bounds perform better than they did in dense instances. The base relaxation (2.7) has at most a 9% gap and improves with p; this is not surprising, because when $p = 1$, (2.7) coincides with the linear relaxation of the deterministic model's edge formulation, which is tight for trees. The path cuts are able to reduce the gap to nearly zero in all cases, improving again as p increases. Finally, the MEWH also performs extremely well here, with a gap under 1% in all cases.

Table 2.2: Geometric mean and standard deviation of sparse instance results as ratios of the HSO bound.

p	(2.7)	(2.7) + Path	MEWH
0.5	1.091 ± 0.008	1.011 ± 0.004	0.992 ± 0.002
0.75	1.071 ± 0.009	1.005 ± 0.003	0.993 ± 0.003
0.9	1.033 ± 0.004	1.001 ± 0.001	0.992 ± 0.004

2.5.4 Impact of Non-Uniform Probabilities

In the previous experiments, all instances have a uniform non-zero probability. In particular, for the dense instances we obtained the best bound with the probabilistic clique inequalities (2.9); however, if the non-zero probabilities are not uniform, we must use a relaxed version of (2.9) with the minimum probability of all pairs in the clique. We now explore the impact of non-uniform probabilities on the quality of this upper bound.

As in the previous section, we consider graph topologies generated using $G(100, p_1)$ for $p_1 \in \{0.5, 0.75, 0.9\}$. We now choose edge probabilities uniformly at random from the intervals $[0.75 - \delta, 0.75 + \delta]$ for $\delta \in \{0, 0.05, 0.15, 0.25\}$; for $\delta = 0$ this is equivalent to our prior experiment with $p_2 = 0.75$. For each instance we compare the upper bound (UB) provided by (2.7) strengthened with path, cycle, and clique constraints against the HSO bound. The chart in Figure 2.1 plots the ratio UB/HSO as a function of the half-width of the edge probability intervals.

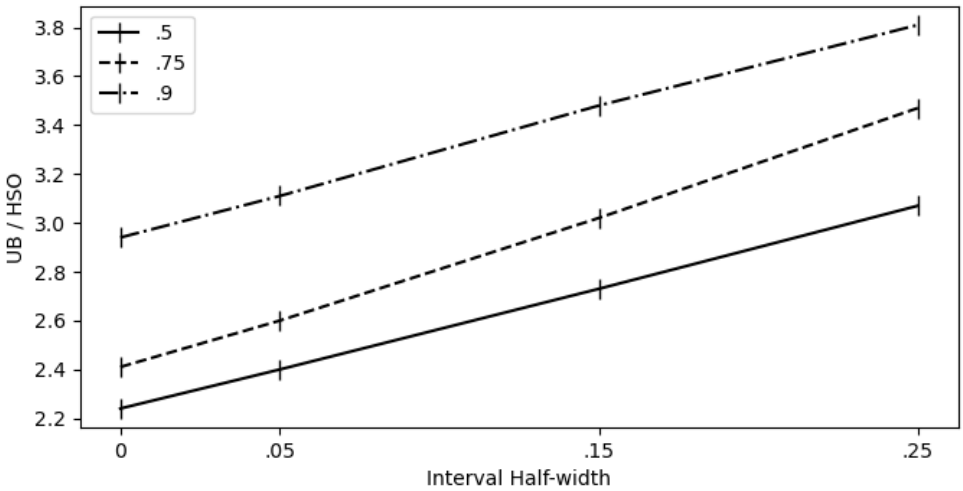

Figure 2.1: Effect of edge probability variation on bound quality.

As expected, the bound's quality is negatively affected by increasing variation in the

edge probabilities; increasing the half-width from 0 to 0.25 increases the ratio by approximately 30% to 45% in relative terms for all three values of p_1.

2.6 Conclusion

We introduced a dynamic model of the weighted node packing problem. Our model generalizes the deterministic node packing problem by introducing both elements of uncertainty and dynamic decision making. Specifically, we provide a model in which edge information is revealed dynamically during the node packing process. We formulate the DNP as a Markov decision process and focus on studying the corresponding polytope of achievable probabilities. Our study yields the achievable probability polytope for two important structures; for cliques, we gave an explicit representation and showed that it is a submodular polyhedron, while for paths and cycles we implicitly characterized the polytope via a separation routine using a cut-generating LP. Our inequalities are instrumental in reducing the upper bound of the edge-based linear relaxation; for dense instances, clique cuts can reduce the gap by nearly 75%, while in sparse instances based on trees, our path inequalities suffice for near-optimality.

Nevertheless, there is a significant gap left to close, especially in denser instances. For example, a natural question is how to extend our results on probabilistic cliques with non-uniform edge probabilities beyond the case of triangles, although our preliminary results in this direction show that even for probabilistic analogues of K_4 the number of facets is very large, and their different structures are quite varied. Another promising direction is to study probabilistic analogues of other simple graph structures, such as claws, which are known to be important in the deterministic case.

CHAPTER 3

INTERVAL SCHEDULING WITH ECONOMIES OF SCALE

3.1 Introduction

Interval scheduling, sometimes also called fixed interval scheduling, is a broad class of problems arising in operations research, computer science, production, scheduling, and logistics. Generally, interval scheduling problems task the decision maker with scheduling a set of jobs by assigning them to machines. Each job is defined by a fixed start and end time and each machine can only process one job at a time. A feasible solution is a partition of jobs into schedules such that jobs within the same schedule do not overlap in time. Various objectives are used in interval scheduling problems and are typically specific to the application.

Interval scheduling and its related problems have numerous applications. Some of these include channel assignment [56], fleet planning [57], bus scheduling [58], satellite scheduling [59, 60], and circuit design [61]. We are primarily interested in interval scheduling for its applications in the area of cloud scheduling. An important problem in cloud services is the allocation of virtual machine (VM) requests to machines. Each VM request has a start time, end time and a set of resource requirements, and the goal is to assign the VMs to machines such that each machine's capacity is never exceeded. Such allocation problems lead to a temporal bin packing problem, with typical objectives including either to minimize the total number of machines required to process all VMs [62, 63], or to minimize some operating cost, e.g., power consumption [64]. Interval scheduling is a special case in which each machine can only process one VM at a time, as when managing requests for large services, e.g., intensive machine learning systems.

In this work, we study the cost minimization form of interval scheduling. Given some

function that computes a cost for each schedule, the decision maker seeks a partition of the jobs that minimizes the sum of schedule costs. We study cost functions exhibiting economies of scale; specifically, we focus on the max-weight function and concave functions of total schedule weight. Interval scheduling with the max-weight function and many concave functions is known to be NP-Hard, and despite various theoretical results, work on exact solution approaches is limited. With this motivation, we propose an approach based on column generation for a set covering formulation. We summarize our primary contributions as follows:

1. To the best of our knowledge, we are the first to address the exact optimization of certain NP-Hard variants of interval scheduling.

2. We propose efficient pricing algorithms for the cases of a max-weight function and more general functions based on a schedule's cumulative weight.

3. For the specific case of the max-weight function on path instances, which is known to be polynomially solvable, existing formulations and our proposed set cover formulation are fractional; we provide a tight linear programming formulation. (In a path instance, each job overlaps only with its immediate predecessor and successor.)

4. We present a detailed computational study demonstrating the effectiveness of our approach; in particular, we are able to exactly optimize instances with up to several hundred jobs.

The outline of the chapter is as follows. In the next section we provide an overview of the relevant literature. In Section 3.3, we give a formal description of our model, provide the relevant preliminaries, and introduce our set covering formulation. In Section 3.4, we describe our column generation algorithm, present details of our pricing algorithms, and extend our approach to branch-and-price. In Section 3.5, we study the special case of the max-weight schedule cost on a path. Section 3.6 details the results of our computational study, and Section 3.7 concludes.

3.2 Literature Review

The literature on interval scheduling is extensive. In the most basic variant, the goal is to minimize the number of machines required to process all jobs. This can be solved in linear time using a first-fit rule [65]. Much of the interval scheduling literature focuses on a related problem that asks, for some integer k, what is the maximum cardinality or maximum-weight set of jobs that can be processed on k machines. [66] provide a $O(n + k)$ algorithm for the cardinality case and a $O(nS(n))$ algorithm for the weighted case, where $S(n)$ is the running time of a shortest-path algorithm on a graph with $O(n)$ edges. [67] propose a linear programming approach, using the fact that the conflict graph's clique incidence matrix is TU; this leads to an integral linear program capable of solving the weighted k-machine problem. The survey [68] provides a detailed overview of common variants of k-machine interval scheduling. Example variants include allowing for machine downtime [69], non-identical machines [67, 70], and jobs requiring multiple processing intervals [71].

Closely related to our work is the literature on submodular coloring, its complement, submodular clique partitioning, and submodular interval scheduling, which is submodular coloring restricted to interval graphs. In [72], the authors study value-monotone submodular functions and provide a 5-approximation algorithm for interval graphs and a 7-approximation for perfect graphs when the cost function is a non-decreasing weight-defined concave function. [73] explore a broader class of submodular functions, referred to as value-polymatroidal; the authors provide a dynamic programming algorithm for co-interval graphs. [74] present a robust coloring algorithm and provide a 4-approximation algorithm for interval graphs for any concave function. Additionally, [74] prove NP-Hardness for coloring interval graphs with strictly concave functions. This result is based on a proof that minimum-entropy coloring is NP-Hard for interval graphs [75]. [76] study coloring in general graphs with the max-weight function, in which the cost of a color class is equal to weight of the highest-weight node contained in the class, and prove that it is NP-Hard for

interval graphs.

Methodologically, our work leverages column generation, which is used to solve linear programs with exponentially many variables. Such problems often arise when solving the linear relaxation of exponentially-sized reformulations of integer programs. These reformulations are desirable because they typically provide stronger dual bounds compared to compact formulations, and solutions to the relaxation can be used to generate heuristic primal solutions. For a general reference on column generation, we refer the reader to [77]. Finding integer solutions to column generation models is challenging, as standard IP techniques like branch-and-bound are ill-suited. This motivates the development of the branch-and-price algorithm [78], which combines column generation with branch-and-bound. Branch-and-price algorithms have been used successfully in many applications; some examples related to our problem include graph coloring [79], max-weight coloring [80], sum coloring [81], robust coloring [82], partition coloring [83], clique partitioning with minimum size [84], and interval scheduling with a resource constraint [85].

3.3 Problem Statement and Preliminaries

We consider interval scheduling problems taking as input a set of n interval jobs $J = \{1, 2, 3, ..., n\}$, a vector of job weights $w \in \mathbb{Z}_+^n$, and a cost function $f : 2^J \to \mathbb{R}_+$. For each job i we assume its start and end times satisfy $s_i, e_i \in \mathbb{Z}_+$. We assume without loss of generality that jobs are ordered by non-decreasing start times, then by end times if necessary. We refer to subsets $S \subseteq J$ as *schedules*, and S is a *feasible schedule* if jobs in S do not overlap in time. We use $G = (N, E)$ to denote the instance's conflict graph.

The decision maker is interested in partitioning J into disjoint feasible schedules S_1, S_2, ..., S_k such that the combined cost of the schedules is minimized. Letting $\mathbb{S}$ be the set of all

feasible schedules, the problem is

$$\min_{S_1,\ldots,S_k \in \mathbb{S}} \left\{ \sum_{\ell=1}^{k} f(S_\ell) : \quad \bigcup_{\ell=1}^{k} S_\ell = J; \quad S_\ell \cap S_m = \emptyset,\ \ell \neq m \right\}. \tag{3.1}$$

Problem (3.1) is equivalent to a minimum-cost coloring problem on the instance's conflict graph. We consider the following cost functions f:

1. Max-Weight: $f(S) = \max_{i \in S} w_i$
2. Non-Negative, Non-Decreasing, Concave: $f(S) = h\left(\sum_{i \in S} w_i\right)$ for some non-negative, non-decreasing concave function h

Both of these functions belong to the class of *value-polymatroidal* functions [73] and *value-monotone submodular* functions [72]. Additionally, both are known to be theoretically difficult; the max-weight function and many weight-defined concave functions make (3.1) NP-Hard [76, 74].

Problem (3.1) with the max-weight function is a special case of the general *max-weight coloring* problem restricted to interval graphs. The max-weight coloring problem has a natural mixed-integer linear programming (MILP) formulation: Given a graph $G = (N, E)$, let Δ be the maximum degree in G, and $\mathbb{C}$ its set of maximal cliques. If G is a conflict graph, we additionally use $\mathbb{C}$ to represent the corresponding sets of jobs in the underlying scheduling instance. The max-weight coloring problem is then formulated as

$$\min_{x,y} \sum_{k=1}^{\Delta+1} y_k \tag{3.2a}$$

$$\text{s.t.} \sum_{i \in C} x_{i,k} \leq 1 \qquad \forall C \in \mathbb{C}, \forall k \in \{1, 2, .., \Delta + 1\} \tag{3.2b}$$

$$\sum_{k=1}^{\Delta+1} x_{i,k} = 1 \qquad \forall i \in N \tag{3.2c}$$

$$w_i x_{i,k} \leq y_k \qquad \forall i \in N, \forall k \in \{1, 2, .., \Delta + 1\} \tag{3.2d}$$

$$x_{i,k} \in \{0, 1\} \qquad \forall i \in N, \forall k \in \{1, 2, .., \Delta + 1\} \tag{3.2e}$$

$$y_k \geq 0 \qquad \forall k \in \{1, 2, ..., \Delta + 1\}. \tag{3.2f}$$

This formulation makes use of the observation that an optimal solution would never require more than $\Delta + 1$ colors.

For general graphs, (3.2) may be exponentially large depending on the graph's number of maximal cliques. When restricted to interval graphs, the model is polynomial as the number of maximal cliques is $O(n)$. Despite this advantage, in practice modern MILP solvers struggle to solve (3.2) for interval scheduling instances with even a moderate number of jobs, in part because of its symmetry.

3.3.1 Set Covering Formulation

We propose a formulation for (3.1) with exponentially many variables. Our model operates in the space of schedules, with each binary variable indicating whether to use a particular schedule. This yields

$$\min_{z} \sum_{S \in \mathbb{S}} f(S) z_S \tag{3.3a}$$

$$\text{s.t.} \sum_{S \in \mathbb{S}, S \ni i} z_S \geq 1 \qquad \forall i \in J \tag{3.3b}$$

$$z_S \in \{0, 1\} \qquad \forall S \in \mathbb{S}. \tag{3.3c}$$

Although feasible solutions should technically be partitions of the job set J, we relax the equality constraint to greater-than-or-equal in (3.3b), transitioning from a partition model into an equivalent covering model; the equivalence follows from the monotonicity of the function f. This helps improve the convergence of the column generation algorithm.

3.4 Solution Methodology

We solve the linear relaxation of (3.3) with column generation. The algorithm maintains a *restricted master problem* (RMP) that only contains a subset of the variables. Letting $\mathbb{S}_{\text{RMP}} \subseteq \mathbb{S}$ denote the set of schedules currently in the RMP, we obtain

$$\min_{z}\left\{ \sum_{S\in\mathbb{S}_{\text{RMP}}} f(S)z_S : \quad \sum_{S\in\mathbb{S}_{\text{RMP}},S\ni i} z_S \geq 1,\ i\in J; \quad z_S \geq 0,\ S\in\mathbb{S}_{\text{RMP}}\right\}. \tag{3.4}$$

Given a particular set of schedules $\mathbb{S}_{\text{RMP}}$, we solve (3.4) and use the optimal dual multipliers to check if any schedule $S \notin \mathbb{S}_{\text{RMP}}$ has negative reduced cost; this is known as the *pricing problem*. If we find any such schedules, we add them to $\mathbb{S}_{\text{RMP}}$ and re-solve. Otherwise, the current solution is optimal for the linear relaxation of (3.3).

After solving (3.4), the pricing problem is

$$\max_{S\in\mathbb{S}}\left\{\sum_{i\in S}\pi_i - f(S)\right\}, \tag{3.5}$$

where π_i is the dual multiplier of the covering constraint in (3.4) for job i. If the optimal value of (3.5) is positive, this corresponds to a schedule with negative reduced cost that can be added to $\mathbb{S}_{\text{RMP}}$. In the next two sub-sections, we respectively present methods for solving (3.5) when f is the max-weight function or any function of the total schedule weight.

3.4.1 Pricing: Max-Weight Function

Substituting $f(S) = \max_{i\in S} w_i$ in (3.5) yields the pricing problem

$$\max_{S\in\mathbb{S}}\left\{\sum_{i\in S}\pi_i - \max_{i\in S} w_i\right\}. \tag{3.6}$$

Proposition 3.4.1. *Problem* (3.6) *can be solved in* $O(nP(n))$ *time, where* $P(n)$ *is the complexity of solving an interval packing problem with* n *jobs.*

Proof. Fixing $\max_{i \in S} w_i = \bar{w}$, (3.6) reduces to

$$\max_{S \in \mathbb{S}} \left\{ \sum_{i \in S} \pi_i : \quad w_i \leq \bar{w}, i \in S \right\}.$$

That is, the problem reduces to finding a schedule maximizing the sum of the π_i values, using only jobs with $w_i \leq \bar{w}$. This is an *interval packing* problem, a special case of the *node packing* problem restricted to interval graphs. As there are at most n distinct values for $\bar{w}$, problem (3.6) can be solved by solving at most n interval packing problems. □

Proposition 3.4.1 implies that the complexity of (3.6) is determined by the complexity of the resulting interval packing problem. Interval packing is well known to be polynomially solvable. We present two algorithms for its solution, both of which we implement as part of our branch-and-price methodology. The first of these methods is based on linear programming; given dual prices π, we formulate the interval packing problem as

$$\max_{x} \left\{ \sum_{i \in J} \pi_i x_i : \quad \sum_{i \in C} x_i \leq 1, \ \forall C \in \mathbb{C}; \quad x_i \geq 0, \ \forall i \in J \right\}.$$

For general graphs, this LP is not necessarily integral; however, for interval graphs the constraint matrix satisfies the consecutive ones property, implying the formulation is integral, and thus an optimal extreme point solution yields the desired packing. Additionally, as the number of maximal cliques in an interval graph is $O(n)$, it can be solved efficiently using any standard linear programming algorithm.

A full round of pricing in our scheme requires solving multiple interval packing instances; each instance is specified by an upper bound $\bar{w}$ on the weight of jobs admissible to the packing. This upper bound can be incorporated into the LP by fixing $x_i = 0$ when w_i is greater than $\bar{w}$. A benefit of the linear programming approach is the ability to warm-start; the sub-problems are solved in increasing order with respect to the weight upper bound, and each LP can be warm-started from the previous optimal solution, as primal feasibility is maintained.

The second method to solve (3.6) is based on dynamic programming (DP). We define states $i \in \{0, 1, 2, ..., n, n+1\}$, where state $i = 1, 2, .., n$ corresponds to job $i \in J$ being added to the schedule, and states $0, n+1$ serve as artificial start and stop points. At each state, the decision maker determines which job is next in the schedule, or ends the schedule. The action set at i is $A_i = \{j \in J : s_j \geq e_i\} \cup \{n+1\}$, with rewards $r_j = \pi_j$ for $j \in J$ and $r_{n+1} = 0$. The initial state is 0 and its action set is $A_0 = J \cup \{n+1\}$. Using v_i to denote the optimal value at state i, we obtain the recursion

$$v_i = \max_{j \in A_i} \{r_j + v_j\}, \quad \forall i \in J \cup \{0\}, \qquad\qquad v_{n+1} = 0. \tag{3.7}$$

The above DP has $\Theta(n)$ states and $O(n)$ actions per state, making the total complexity $O(n^2)$. While generally leading to a slower empirical running time than the LP method, the DP model more easily incorporates side constraints on the structure of feasible schedules; specifically, it allows merging and separation constraints that require or forbid certain jobs from running consecutively in a schedule. These constraints are important as they correspond to branching disjunctions in our branch-and-price algorithm described in Section 3.4.3.

It is possible to obtain a DP with linear complexity by instead making admit/reject decisions at each job state. The jobs are considered in arrival order and if a job is admitted the decision maker receives the reward π_i and the state transitions to the next job that could be feasibly added to the schedule. It is easy to verify that an algorithm of this type has $O(n)$ running time, but it loses the ability to incorporate separation and merge constraints, the primary advantage of the DP approach over LP. We discuss further details on this type of DP below in Section 3.4.2.

Identifying the most improving column requires solving $O(n)$ interval packing problems; however, any sub-problem may produce an improving column. An option is to conduct *partial pricing* and terminate after finding any improving column. This typically trades

a higher total iteration count for a lower time per iteration. Likewise, it is also possible to conduct a full round of pricing and return all improving columns instead of just the most improving. These implementation details can have a major impact on the running time and convergence of the column generation algorithm. See Section 3.6.1 for details on this design decision.

3.4.2 Pricing: Function of Total Weight

Next we consider the pricing problem (3.5) when f is a a function of the total schedule weight. That is, for some function h, the cost is $f(S) = h(\sum_{i \in S} w_i)$. This includes the special case in which h is non-negative, non-decreasing and concave. This leads to the pricing problem

$$\max_{S \in \mathbb{S}} \left\{ \sum_{i \in S} \pi_i - h\left(\sum_{i \in S} w_i \right) \right\}.$$

We use DP to solve this problem: Define states (i, W) for $i \in \{0, 1, 2, ..., n+1\}$ and $W \in \{0, 1, 2, ..., \sum_{i \in J} w_i\}$, where state (i, W) indicates that i was just added to the schedule and the sum of the scheduled jobs' weights is W. As with (3.7), we define the action set at state (i, W) as $A_i = \{j \in J : s_j \geq e_i\} \cup \{n+1\}$ and $A_0 = J \cup \{n+1\}$. The reward for taking action $j \in J$ is $r_j = \pi_j$ and $r_{n+1} = 0$. The initial state is $(0, 0)$. Using $u_{i,W}$ to denote the optimal value at state (i, W), we get the recursion

$$u_{i,W} = \max_{j \in A_i} \{r_j + u_{j,W+w_i}\} \qquad \forall i \in J \cup \{0\}, \forall W = 0, 1, ..., \sum_{i \in J} w_i \tag{3.8a}$$

$$u_{n+1,W} = -h(W) \qquad \forall W = 0, 1, ..., \sum_{i \in J} w_i. \tag{3.8b}$$

The DP has $\Theta(n \sum_{i \in J} w_i)$ states and each state has $O(n)$ actions; therefore, it is pseudo-polynomial with complexity $O(n^2 \sum_{i \in J} w_i)$. If $\sum_{i \in J} w_i$ is a polynomial function of n, the complexity is polynomial, e.g., when h is a function of the schedule's cardinality. As before, an alternate DP formulation with lower complexity is possible, at the cost of less

modeling power; we discuss this approach next.

Again, we define states (i, W) for $i \in J$ and $W \in \{0, 1, ..., \sum_{i \in J} w_i\}$, where i now indicates deciding whether to add i to the schedule or not. Specifically, the actions at (i, W) are to add i, receiving reward π_i and transitioning to state $(\eta_i, W + w_i)$, where $\eta_i = \min\{j \in J : s_j \geq e_i\}$, or to reject i, receiving no reward and transitioning to $(i + 1, W)$. The initial state is $(0, 0)$. Letting $\sigma_{i,W}$ denote the optimal value at state (i, W), we have

$$\sigma_{i,W} = \max\{\pi_i + \sigma_{\eta_i, W+w_i}, \sigma_{i+1,W}\} \qquad \forall i \in \{1, 2, ..., n-1\}, \forall W = 0, 1, ..., \sum_{i \in J} w_i \tag{3.9a}$$

$$\sigma_{n,W} = \max(\pi_n - h(W + w_n), -h(W)) \qquad \forall W = 0, 1, ..., \sum_{i \in J} w_i. \tag{3.9b}$$

This DP has $\Theta(n \sum_{i \in J} w_i)$ states and two actions per state, so the total complexity is $\Theta(n \sum_{i \in J} w_i)$. This complexity is better than (3.8) by a factor of n, and is likely the faster pricing method when there are no separation or merge constraints, i.e., at the root node of a branch-and-price tree. Lastly, as with the max-weight function, it is possible to identify multiple improving columns. Both DP recursions include information on the best schedule starting from job j for all jobs $j \in J$; any such schedule may also provide an improving column.

3.4.3 Branch-and-Price Algorithm

We now extend the column generation approach to an exact algorithm for (3.3) by combining column generation and branch-and-bound. The LP at each node in the search tree is solved via column generation. Instead of branching directly on the variables in (3.3), we adopt a branching scheme common in the vehicle routing literature: We define arc variables $\theta_{i,j}$ for each pair $i, j \in J$, $i < j$, of non-overlapping jobs, representing the decision that j immediately follows i in some schedule. Given a fractional solution z^*, we define

$\sum_{S \in \mathbb{S}} \mathbb{1}_{\{(i,j) \in S\}} z^*_S = \theta_{i,j}$, where $\mathbb{1}_{\{(i,j) \in S\}}$ indicates that j immediately follows i in schedule S. Given a fractional $\theta_{i,j}$, we partition the solution space using the constraints $\theta_{i,j} = 1$ and $\theta_{i,j} = 0$. This is equivalent to partitioning the space based on merge and separation constraints: j must immediately follow i, or j cannot immediately follow i. Both recursions (3.7) and (3.8) can incorporate these branching constraints by updating their respective action sets. In the $\theta_{i,j} = 0$ branch, we eliminate the action j from states at i. For the $\theta_{i,j} = 1$ branch, j is the only action at i, and j is removed from the action set of all other states. We outline additional details of our branch-and-price implementation in Section 3.6.1.

3.4.4 Primal Heuristics

To complement the lower-bounds obtained from solving the linear relaxation of (3.3), we propose a constructive heuristic and local-search algorithm for both the max-weight and concave functions. We use a greedy packing heuristic to obtain an initial feasible solution. Starting from the initial set of jobs $U = J$ and an empty set of schedules $V = \emptyset$, we solve an interval packing problem to obtain a feasible schedule S, update the sets to $U \leftarrow U \setminus S$ and $V \leftarrow V \cup \{S\}$, and repeat until $U = \emptyset$, which is achieved when V corresponds to a feasible solution to (3.1). While not explicitly required, we find that the heuristic's performance improves using modified weights $\hat{w}_i = w_i^2$; this biases the heuristic to prefer scheduling high-weight jobs together over scheduling many low-weight jobs.

We use a local-search heuristic to improve the initial solution found by the greedy heuristic. Given a set of schedules V, the heuristic considers pairs of schedules $S_\ell, S_m \in V$ and searches for a pair of schedules S'_ℓ, S'_m such that $f(S'_\ell) + f(S'_m) < f(S_\ell) + f(S_m)$ and $S'_\ell \cup S'_m = S_\ell \cup S_m$. The local search repeatedly checks all pairs $S_\ell, S_m \in V$, updating as better schedules are found. The search continues until no pair results in an improved solution.

The local search algorithm requires solving an optimization problem for the pairwise schedule improvement. This is equivalent to the interval scheduling problem (3.1) re-

stricted to instances with conflict graphs having cliques of size at most two, i.e., the conflict graphs are bipartite. For the max-weight function, this can be done directly using the formulation (3.2), limiting the number of schedules to two. This problem can be solved in polynomial time, as we establish with the following proposition.

Proposition 3.4.2. *Given schedules $S_\ell, S_m \subseteq J$, the problem*

$$\min\{\max_{i \in S'_\ell} w_i + \max_{i \in S'_m} w_i : S'_\ell \cup S'_m = S_\ell \cup S_m\}$$

can be solved in polynomial time.

Proof. First assume that the instance's conflict graph is connected. Ignoring symmetry, there is a single solution to this problem using two schedules. Start at the earliest arriving job, then begin assigning jobs in arrival order. By assumption the conflict graph's coloring number is two and this placement will produce a solution using two schedules, see [65]. To see this is the only solution, note that each job i's color is determined entirely by the color of earliest arriving job j that i overlaps. As the graph is connected, after choosing the color of the first arriving job the remaining decisions are fixed. If the conflict graph has more than one connected component, an optimal solution is obtained by constructing one schedule S'_ℓ containing the schedule with minimum max-weight in each component and placing the remainder in a second schedule S'_m. By construction, this will create a solution in which $\max_{i \in S'_\ell} w_i$ is minimized and $\max_{i \in S'_m} w_i = \max_{i \in S_\ell \cup S_m} w_i$; note, there will always be at least one schedule with cost $\max_{i \in S_\ell \cup S_m} w_i$. Finally, the construction of the schedules can be done in $O(n)$ time and the result follows. □

For non-negative, non-decreasing concave functions, we use a similar argument as in Proposition 3.4.2.

Proposition 3.4.3. *Given schedules $S_\ell, S_m \subseteq J$, the problem*

$$\min\{h(\sum_{i \in S'_\ell} w_i) + h(\sum_{i \in S'_m} w_i) : S'_\ell \cup S'_m = S_\ell \cup S_m\}$$

where h is a non-negative, non-decreasing concave function can be solved in polynomial time.

Proof. Consider the same algorithm as described in the proof of Proposition 3.4.2; instead of assigning to S'_ℓ the sub-schedules with minimum max-weight, assign the sub-schedule in each component with the larger total weight. By the same argument as before, this will create the schedule S'_ℓ with the maximum total weight. As each job must be scheduled, after placing the remaining jobs in S'_m, it will be the schedule of minimum total weight. Finally, as this solution maximizes the difference $h(\sum_{i \in S'_\ell} w_i) - h(\sum_{i \in S'_m} w_i)$, the concavity of h implies this solution minimizes $h(\sum_{i \in S'_\ell} w_i) + h(\sum_{i \in S'_m} w_i)$ and the result follows. □

3.5 Max-Weight Function on Paths

In this section, we focus on the max-weight function restricted to instances where the conflict graph is a path. Equivalently, each job $1 < i < n$ overlaps with its predecessor and successor, $i - 1$ and $i + 1$, and with no other jobs. Unlike the more general problem setting we consider, this special case is polynomially solvable [76]; however, there are instances in which the linear relaxations of both (3.2) and (3.3) are fractional. Motivated by this discrepancy, we present a tight LP formulation for this special case that leverages the algorithm in [76] and the fact that an optimal solution never uses more than three schedules.

The intuition behind the algorithm is the following. Each schedule is defined by its highest-weight job, and one of these is always the max-weight job, $f(S_1) = \max_{k \in J} w_k$. Some job $i \in J$ defines the second schedule by having its highest-weight job, $f(S_2) = w_i$, and the third schedule then satisfies $f(S_3) \leq f(S_2)$. This implies that $j \in S_1$ if $w_j > w_i$; furthermore, for any two jobs $j, k \in S_1$, $j < k$ with an even number of jobs in between, a

feasible solution must have $\ell \in S_3$ for some job $\ell \in (j,k)$. (Conversely, if there is an odd number of jobs between j and k, they can be assigned in alternating fashion to S_1 and S_2.) Denote job i's minimal even pairs as

$$E(i) = \{(j,k) : \quad j,k \in J; \quad w_j, w_k > w_i; \quad j < k; \\ k - j = 1 \mod 2; \quad w_\ell \leq w_i,\ \ell \in (j,k)\}.$$

We illustrate idea with the six-job example in Figure 3.1(a), with weight vector $w = (6,1,2,5,3,4)$.

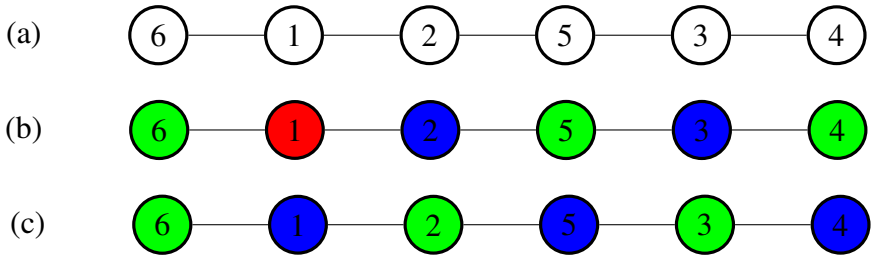

Figure 3.1: Example path instance with six jobs.

Suppose $f(S_2) = w_5 = 3$; then S_1 must contain the first, fourth and sixth jobs, as $w_1, w_4, w_6 > 3$. Clearly, we also assign the fifth job to S_2. However, we cannot assign both the second and third jobs to S_2, because they are adjacent; one of these must go in S_3, so we place the second job there, since it has the lower weight of the two. The resulting schedule is illustrated in Figure 3.1(b).

Suppose instead that $f(S_2) = w_4 = 5$. In this case, S_1 must contain the first job. However, this also means there are no even pairs, and we can simply assign jobs to S_1 and S_2 in alternating fashion. This schedule is illustrated in Figure 3.1(c). For this instance, these are essentially the only options: If we choose $f(S_2) \leq 2$, we force S_1 to contain adjacent jobs, which is infeasible. This can also be interpreted as having even pairs of the form $(j, j+1)$, which have no interior element and thus cannot include a job in S_3. Finally, if we choose $f(S_2) \in \{4,6\}$, we are forced into one of the previously illustrated schedules.

Formally, there is an optimal solution satisfying the following conditions [76, Theorem

7]:

1. The solution uses at most three schedules, S_1, S_2, S_3, where possibly $S_3 = \emptyset$.
2. $\max_{k \in J} w_k = f(S_1) \geq f(S_2) \geq f(S_3)$.
3. Given $f(S_2) = w_i$, S_3 contains exactly one job $\ell \in (j,k)$ from each $(j,k) \in E(i)$.

Based on this, we formulate

$$\min \sum_{i \in J} (w_i x_i + \lambda_i) \tag{3.10a}$$

$$\text{s.t.} \sum_{i \in J} x_i = 1 \tag{3.10b}$$

$$x_i = \sum_{j<\ell<k} y_{i,\ell} \qquad \forall i \in J, \forall (j,k) \in E(i) \tag{3.10c}$$

$$\lambda_i \geq \sum_{j<\ell<k} w_\ell y_{i,\ell} \qquad \forall i \in J, \forall (j,k) \in E(i) \tag{3.10d}$$

$$x_i, \lambda_i \geq 0 \qquad \forall i \in J \tag{3.10e}$$

$$y_{i,\ell} \geq 0 \qquad \forall i, \ell \in J. \tag{3.10f}$$

The LP (3.10) has $\Theta(n^2)$ variables and $O(n^2)$ constraints.

Lemma 3.5.1. *A feasible solution of* (3.10) *with integer x, y variables corresponds to a feasible solution of the max-weight interval scheduling problem on a path instance that satisfies conditions 1–3, and vice versa.*

Proof. Assume we have a feasible solution to (3.10) with integer x, y variables; suppose $x_i = 1$. The right-hand side of some constraint (3.10c) is zero if $E(i)$ contains any pair of consecutive jobs, so we may assume that all minimal even pairs $(j,k) \in E(i)$ have $k - j \geq 3$. Furthermore, these constraints require $y_{i,\ell} = 1$ for some $\ell \in (j,k)$; we define $S_3 = \{\ell \in J : y_{i,\ell} = 1\}$. By construction, S_3 is a feasible schedule, as only one $y_{i,\ell}$ variable equals 1 in each minimal even pair. To define S_1 and S_2, we begin by adding to

S_1 all jobs $j \in J$ with $w_j > w_i$; the remaining jobs are placed into S_1 and S_2 in alternating fashion: For each minimal even pair $(j,k) \in E(i)$ with $\ell \in S_3 \cap (j,k)$, jobs $m \in (j,\ell)$ go into S_1 if they have j's parity and into S_2 otherwise. Similarly, jobs $m \in (\ell,k)$ go into S_1 if they have k's parity and into S_2 otherwise. The operation is analogous but simpler for "odd pairs," where we simply place jobs into S_1 and S_2 in alternating fashion. This construction ensures both S_1 and S_2 are feasible, as we never add adjacent jobs to the same schedule. This solution satisfies 1, 2, and 3.

Now assume we have schedules S_1, S_2, S_3 satisfying 1–3. To construct an integer solution to LP (3.10), take $i \in \operatorname{argmax}_{k \in S_2} w_k$ and set $x_i = 1$, with $x_j = 0$ otherwise. Condition 3 requires that S_3 contain one job $\ell \in (j,k)$ for each $(j,k) \in E(i)$. Set the corresponding $y_{i,\ell} = 1$ and the remaining y variables to zero. Lastly, set $\lambda_i = \max_{\ell \in S_3} w_\ell$ and other λ variables to zero. Constraints (3.10c) are satisfied, as there is exactly one $y_{i,\ell}$ variable equal to one for each minimal pair in $E(i)$. Constraints (3.10d) are satisfied by construction. If $S_3 = \emptyset$, $E(i) = \emptyset$ as well, so (3.10c), (3.10d) are satisfied trivially. □

We can now establish that (3.10) solves the scheduling problem.

Proposition 3.5.2. *An optimal extreme point solution of* (3.10) *has integer x, y variables. Therefore,* (3.10) *solves the max-weight interval scheduling problem on path instances.*

Proof. We begin by analyzing the λ variables. At optimality, for each $i \in J$ some constraint (3.10d) must be tight to minimize each λ_i; thus

$$\lambda_i = \max_{(j,k) \in E(i)} \sum_{\ell \in (j,k)} w_\ell y_{i,\ell}.$$

For the maximizing interval (j,k) in this expression, (3.10c) requires $x_i = \sum_{\ell \in (j,k)} y_{i,\ell}$, so we minimize λ_i by setting $y_{i,m} = x_i$, where $m \in \operatorname{argmin}_{(j,k)} w_\ell$. Therefore, at optimality we have

$$\lambda_i = x_i \max_{(j,k) \in E(i)} \min_{j < \ell < k} w_\ell, \qquad i \in J,$$

where $\lambda_i = 0$ if $E(i) = \emptyset$. In particular, for optimal solutions λ is a linear function of the x variables.

By setting $c_i = \max_{(j,k)\in E(i)} \min_{j<\ell<k} w_\ell$, we project out y and λ so that (3.10) is equivalent to

$$\min\left\{\sum_{i\in J}(w_i + c_i)x_i : \sum_{i\in J} x_i = 1; x \geq 0\right\},$$

the optimization of a linear function over a simplex, which has integral extreme points. At optimality, $x_i = 1$ for some $i \in J$ and $x_j = 0$ otherwise. It follows from our reasoning above that we can take the y variables to be binary as well. The resulting optimal objective value corresponds to $f(S_2) + f(S_3)$ for an optimal set of schedules; recall that $f(S_1) = \max_{k\in J} w_k$ is a constant. □

3.6 Computational Study

In this section, we computationally evaluate our proposed methodology. Specifically, we examine the lower bound provided by the linear relaxation of (3.3), the upper bound given by our heuristics, and the performance of our branch-and-price algorithm, including its ability to improve both lower and upper bounds and to prove optimality.

In terms of objective functions, we use the max-weight cost as well as the square root, $f(S) = \sqrt{\sum_{i\in S} w_i}$, as a representative of non-decreasing, non-negative concave functions of total weight. We test our methods on both synthetic instances and instances derived from cloud computing usage data, and when possible also benchmark against the assignment formulation (3.2). We conduct all experiments on a MacBook with a 2.7GHz Dual-Core Intel i5 processor. Our base code uses Python 3.7.3, and the LP and IP solves use Gurobi 9.1 with default parameters unless otherwise noted. In the following sub-sections we discuss details of our implementation and instance design before summarizing our study's results.

3.6.1 Implementation Details

While a basic column generation algorithm can solve the linear relaxation of (3.3), in practice it is frequently enhanced with acceleration techniques, since the basic algorithm can suffer from slow convergence and other issues related to degeneracy. In our implementation, we use *in-out* column generation [86], which attempts to accelerate convergence by better selecting dual variables to use in the pricing problem: We maintain a dual feasible solution π_{in} in addition to the potentially infeasible dual solution π_{out} obtained from the current solution to the restricted LP (3.4). We solve the pricing problem on the dual solution $\hat{\pi} = \gamma\pi_{\text{out}} + (1-\gamma)\pi_{\text{in}}$, a convex combination of the "in" and "out" solutions using convex multiplier $\gamma \in (0, 1]$. If $\hat{\pi}$ is infeasible, we generate a new column; otherwise, we update $\pi_{\text{in}} \leftarrow \hat{\pi}$. In addition to improving convergence times, the in-out method provides a way of obtaining valid dual bounds if the column generation algorithm does not converge within a time limit. Based on initial tests, we set $\gamma = 0.2$ at the root node and $\gamma = 1$ afterwards, equivalent to running the standard column generation algorithm after the root node. We observed that the in-out acceleration method significantly reduced solve time at the root node, but was slightly slower than the basic algorithm for subsequent nodes; the latter could be solved in a small number of iterations and suffered from fewer computational difficulties compared to the root node, likely because of the additional constraints.

We use the faster pricing algorithms at the root node and then the slower, more general algorithms after adding branching constraints. Specifically, for the max-weight function we solve the root LP using the LP-based pricing method and solve the remaining LP's with DP (3.7). For concave functions, we use the faster DP (3.9) to solve the root LP. For both functions, we perform full pricing and incorporate all improving columns after each pricing iteration.

There are various strategies to explore the branch-and-bound tree. In our implementation, at each node we branch on the most fractional $\theta_{i,j}$ value, breaking ties at random; if there is no fractional $\theta_{i,j}$, the solution is feasible and we do not branch. We explore the

tree using an *iterative depth-first search* and process the 1-branch first, i.e., the branch with $\theta_{i,j} = 1$. The iterative depth-first search rule takes integer parameters α, β and follows standard depth-first search for the first α nodes. After we open α nodes, we restart the search from a random unprocessed node, reset the node count to zero, and update the threshold to $\alpha + \beta$. This process repeats until the search terminates. We designed our branching scheme in this manner after initial tests suggested that the lower bound obtained at the root node is often tight, and that the optimality gap tends to stem from the primal side. By biasing the search towards the 1-branches and performing iterative depth-first "dives", we attempt to steer the search to quickly obtain good feasible solutions while avoiding unpromising areas of the search space. In our experiments, we set the iterative depth-first search parameters to $\alpha = 20$ and $\beta = 10$.

Finally, our branch-and-price algorithm obtains upper bounds in two ways. First, at the root node we obtain a feasible solution using the greedy and local-search heuristics described in Section 3.4.4; additionally, we use the schedules generated by these heuristics as the initial columns in the restricted LP (3.4) at the root node. At all other nodes, we implement a rounding heuristic; this heuristic obtains a feasible solution by greedily rounding the node's LP solution into a feasible solution for (3.3). We sort the schedules corresponding to non-zero variables in the LP solution by cumulative schedule weight. We add the schedule with highest cumulative weight to the solution, then re-sort the remaining schedules considering only the weights of jobs not covered by a previously included schedule. This process repeats until we obtain a feasible solution.

3.6.2 Instance Design

We use two types of instances to test our methods, randomly generated synthetic instances and instances constructed from a public cloud computing data set. We generate a random n-job instance by first specifying $t_{\max}$, the latest point at which jobs may arrive; each job i is created by sampling a start time s_i from the integer uniform distribution over interval

$[0, t_{\max}]$ and sampling its length from the integer uniform distribution over $[1, 10]$.

For cloud instances, we use a publicly available Microsoft Azure data set [87]. The data set is comprised of two weeks of requests for virtual machine allocations; each request is associated with an arrival time and a departure time. The data set includes other application-specific data related to resource consumption; however, for our model we are only concerned with request start and end times. To generate an n-job instance, we select a random starting point in the data set, and take the next n arriving requests, creating jobs using their start and end times. We also check if the instance is sufficiently sparse to avoid easy cases in which feasible schedules contain only a few jobs. Specifically, we compute $\max_{C_i \in \mathbb{C}} |C_i|/n$, where $\max_{C_i \in \mathbb{C}} |C_i|$ is the cardinality of the largest clique in the conflict graph, and reject the instance when this ratio exceeds 0.3.

For both instance types we assign weights as uniform random integers in the interval $[1, 100]$. We also considered generating weights correlated to job length, but found in initial tests that uncorrelated weights lead to more difficult instances. We generate *small* instances with $n = 100$, *moderate* instances with $n = 250$, and *large* instances with $n = 400$. In addition, for max-weight synthetic experiments we also test *very large* instances with $n = 550$.

3.6.3 Max-Weight Function: Small and Moderate Synthetic Instances

For small and moderate instances with $n \in \{100, 250\}$, we test our branch-and-price algorithm and also benchmark it against the assignment-type formulation (3.2). We use $t_{\max} \in \{n, n/2, n/5\}$, and generate ten instances for each pair $(n, t_{\max})$. We test both methods with a time limit of three hours; the results are summarized in Table 3.1. For each set of instances, we compare the geometric mean of the optimality gaps at termination, the average running time of instances where the method proves optimality (in seconds), and the percentage of the instances that are solved to optimality. We also compute the geometric means of the optimality gaps at the root node.

Table 3.1: Branch-and-price and assignment formulation experiments with the max-weight function on small, moderate synthetic instances.

n	$t_{\max}$	Root Gap	B&P Gap	Sol. Time	% Sol.	Assign. Gap	Sol. Time	% Sol.
100	100	0.94%	0.00%	41.56	100%	0.00%	436.45	100%
100	50	3.25%	0.00%	43.04	100%	0.75%	1199.34	60%
100	20	1.66%	0.00%	15.01	100%	0.68%	405.86	30%
250	250	1.54%	0.00%	5892.44	100%	2.01%	2228.56	50%
250	125	2.64%	0.00%	2779.20	100%	3.02%	-	0%
250	50	3.11%	0.00%	1006.74	100%	4.46%	-	0%

The results verify that the proposed branch-and-price algorithm outperforms the assignment formulation. For every test instance, the branch-and-price algorithm is able to prove optimality within three hours. The class $(n, t_{\max}) = (100, 100)$ is the only one where the assignment benchmark solves all instances to optimality in the same time. Conversely, the assignment formulation did not prove optimality for any of the $(250, 125)$ or $(250, 50)$ instances. The average solution times are lower for the branch-and-price algorithm, and it scales better as the instances become denser, while the assignment formulation struggles with dense instances. As a final observation, the root LP's lower bound in the branch-and-price tree is often tight, with any gap stemming from the primal side. In contrast, with the assignment formulation the IP solver is often able to find a strong primal solution, but struggles to improve the lower bound.

3.6.4 Max-Weight Function: Large and Very Large Synthetic Instances

To assess the scalability of our branch-and-price algorithm for the max-weight function, we also considered larger synthetic instances. We use $n \in \{400, 550\}$ and $t_{\max} = n/5$. The choice of $t_{\max}$ is motivated by the prior experiments, which indicate that denser instances have larger root gaps but also that the algorithm may scale well. For each pair n, we generate ten random instances and use a six-hour time limit; Table 3.2 summarizes the results. We report the running times and the optimality gaps at termination and at the root

node. We also report the geometric means of final optimality gaps and the average solution time of instances that are solved to optimality.

Table 3.2: Branch-and-price experiments with the max-weight function on large and very large synthetic instances.

n	$t_{\max}$	Root Gap	B&P Gap	Sol. Time	n	$t_{\max}$	Root Gap	B&P Gap	Sol. Time
400	80	0.52%	0.00%	8697.42	550	110	3.68%	3.68%	TL
400	80	4.76%	0.00%	10468.31	550	110	3.12%	3.12%	TL
400	80	3.92%	0.00%	9235.76	550	110	5.39%	5.39%	TL
400	80	1.33%	0.00%	7371.16	550	110	2.94%	2.94%	TL
400	80	3.16%	0.00%	9328.55	550	110	2.31%	2.31%	TL
400	80	1.04%	0.00%	9642.15	550	110	0.95%	0.95%	TL
400	80	1.12%	0.00%	8437.06	550	110	3.16%	3.16%	TL
400	80	2.36%	0.00%	9890.78	550	110	2.76%	2.76%	TL
400	80	3.00%	0.00%	9531.93	550	110	2.86%	2.86%	TL
400	80	2.05%	0.00%	7893.05	550	110	4.54%	4.54%	TL
	Avg.	**1.92%**	**0.00%**	**9049.62**		**Avg.**	**2.92%**	**2.92%**	

The results indicate that the branch-and-price algorithm scales well to large instances, and is able to reliably prove optimality in less than six hours. For the largest instances, the algorithm is unable to reduce the root optimality gap. This difficulty stems from at least two sources: First, the larger job number necessitates a much larger tree to improve the bound and prove optimality. Second, and equally important, the LP solves at each node take significantly longer, meaning we can process a smaller number of nodes within the time limit. Nonetheless, already at the root node we obtain solutions and bounds yielding an average optimality gap of less than 3%. Based on previous experiments, we conjecture that this gap stems primarily from the primal side; improved heuristics could potentially help prove optimality.

3.6.5 Square Root Function: Synthetic Instances

For the square root function we limit ourselves to the root node; equivalently, we solve the linear relaxation of (3.3) and run primal heuristics. We conduct our experiment in this manner for two reasons: Our initial tests suggested that this approach often suffices to prove

optimality for small and moderate instances, while for large instances the LP relaxation itself takes a significant amount of time. For each combination of $n \in \{100, 250, 400\}$ and $t_{\max} = n/5$, we generate ten instances, and run our test with a six-hour time limit; Table 3.3 summarizes the results. For each instance class, we report the geometric mean of the optimality gaps, the average running time for instances where the column generation algorithm converges, the percentage of instances where the column generation algorithm converges, and the percentage of instances where we prove optimality.

Table 3.3: Column generation experiments with the square root function on synthetic instances.

n	$t_{\max}$	CG Gap	CG Time	% Conv.	% Sol.
100	20	0.00%	252.55	100%	100%
250	50	0.30%	4959.18	70%	70%
400	80	1.30%	-	0%	0%

The results indicate that we can prove optimality for all instances where the column generation algorithm converges. The column generation does not converge within the time limit for 30% of the moderate instances and all of the large instances, and we use the dual bounds provided by the in-out acceleration scheme. However, even for the large instances the average optimality gap is only 1.3%. Compared with the max-weight function, the pseudo-polynomial complexity of the pricing algorithm for the square root function makes solving the LP more computationally difficult, but the resulting lower bound is typically stronger.

3.6.6 Cloud Data Instances

As a final experiment, we test our methods on instances derived from cloud computing data, as described in Section 3.6.2. For the max-weight function, we test instances of all sizes, with $n \in \{100, 250, 400, 550\}$, and for the square root function we exclude very large instances and use $n \in \{100, 250, 400\}$; for each function and each n we generate ten

instances. For the max-weight function, we test the branch-and-price algorithm, and for the square root function we solve the root node's LP relaxation.

Table 3.4 summarizes the results for the max-weight function. We report the geometric means of the root and terminal optimality gaps, the average running times of solved instances, and the percentage of instances solved.

Table 3.4: Branch-and-price experiments with the max-weight function on cloud instances.

n	Root Gap	B&P Gap	Sol. Time	% Sol.
100	0.32%	0.00%	16.15	100%
250	0.53%	0.00%	864.48	100%
400	0.94%	0.15%	8589.58	80%
550	0.81%	0.81%	5584.67	40%

In general, we see that the application instances are easier than the synthetic instances. For example, the branch-and-price algorithm is able to solve 40% of the very large instances with the max-weight function, versus none of the analogous synthetic instances. Furthermore, the root optimality gaps are much lower than for the synthetic instances, less than 1% in all cases. An explanation for this behavior may be the density, which tends to be higher in the cloud data; correspondingly, the average solution times are similar to those of the synthetic instances with $t_{\max} = n/5$.

Table 3.5 summarizes the results for the square root function. We report average running times of instances where the column generation algorithm converges, the geometric mean of the optimality gaps at termination, the percentage of instances for which the algorithm converges, and the percentage of instances solved.

Overall, the column generation algorithm converges for more instances than the corresponding synthetic experiment, but this does not lead to a significant change in average optimality gaps. In contrast to the synthetic instances, the column generation algorithm may converge without proving optimality. For example, despite converging for all moderate instances but one, the resulting gaps are slightly lower than for the corresponding

Table 3.5: Column generation experiments with the square root function on cloud instances.

n	CG Gap	CG Time	% Conv.	% Sol.
100	0.01%	453.52	100%	90%
250	0.23%	6990.59	90%	60%
400	1.75%	17791.94	10%	0%

synthetic instances.

3.7 Conclusions

In this work, we propose an exact optimization approach for two classes of interval scheduling problems exhibiting economies of scale; to our knowledge, this is the first such approach proposed for these problems. Our approach is based on column generation and a set covering formulation, using feasible schedules as decision variables. For the max-weight function and functions of cumulative weight, we describe how to solve the linear relaxation of this formulation and provide efficient pricing algorithms. To obtain integer solutions, we extend this method to a full branch-and-price algorithm and provide a detailed account of the relevant design decisions. As a secondary result, we also provide a compact integral formulation for the max-weight function on path instances, a polynomially solvable case for which no such formulation was known. Our computational study provides evidence of our proposed method's effectiveness. We can provably optimize instances with as many as 400 jobs and can otherwise give solutions and bounds with very small gaps.

While we have demonstrated the effectiveness of the set covering formulation on our chosen objective functions, future work can extend the approach to other classes of functions. The major requirement is to determine when the pricing problem can be solved efficiently. Both of the problems studied belong to the more general class of submodular interval scheduling problems. In this case, the general form of the pricing problem is supermodular maximization subject to interval packing constraints. To our knowledge, this

specific problem has not been previously addressed in the literature.

Another area of future work concerns how to leverage polyhedral results for the column generation model. In our computational study, we observed that the root LP often provides a tight lower bound; however, it is easy to construct instances in which this is not the case, even when the conflict graph is a path. For these instances, the lower bound could possibly be strengthened by the addition of valid inequalities.

CHAPTER 4
TEMPORAL BIN PACKING WITH HALF-CAPACITY JOBS

4.1 Introduction

Temporal bin packing (TBP) is an problem of emerging importance in operations research and computer science. It generalizes the well-known bin packing problem, sometimes referred to as the static bin packing problem, by having jobs that arrive and depart over time. There are several variants, but generally the goal is an assignment of jobs to bins that minimizes some cost or performance measure while respecting bin capacities. We are interested in the objective of minimizing the time-averaged number of open or active bins required to process all jobs; in this objective, a bin is considered active only when some job is assigned to it, while other variants consider a bin active for the whole horizon if a job is ever assigned to it.

Our primary motivation stems from applications in cloud computing, where the model captures the assignment of virtual machines to servers while minimizing the average number of active servers, a proxy for energy usage, which is a significant operational cost in server banks. Even small relative improvements in server utilization can lead to large absolute gains; for example, [87] suggest a 1% packing efficiency improvement can lead to cost savings of roughly $100 million per year for Microsoft Azure.

Additional applications come from optical network design, in which a fiber cable system needs to be designed in a manner that satisfies demands for communication signals. Two signals within the same cable cannot be carried within the same channel, and each cable has a fixed number of channels; the goal is to design a system that minimizes the total required length of fiber. The special case of a line system, in which all signals can be thought of as travelling along a one-dimensional line, is equivalent to the temporal bin

packing problem.

We study the special case of TBP in which a bin can accommodate two jobs at a time, and which we denote TBP2. In the cloud computing context, this occurs in specialized systems that focus on serving resource-intensive requests, such as large services and certain machine learning systems. TBP2 has been studied previously and is known to be NP-Hard. Our work focuses on two directions: first, we provide a novel integer programming (IP) formulation; second, we propose various lower bounds for the problem, studying them both theoretically and empirically. We summarize our main contributions below.

1. We propose a novel formulation for TBP2 that interprets the model as a series of related matching problems.

2. We theoretically study multiple lower bounds based on both combinatorial and polyhedral techniques, and derive a bound hierarchy.

3. We conduct a computational study to evaluate the performance of our proposed IP formulation and compare multiple lower bounds, assessing both strength and scalability, and using both synthetic and application-based instances.

Our primary theoretical contribution, Theorem 4.6.7, is summarized graphically in Figure 4.1. Each node in the graph represents a different lower bound, and a directed edge from i to j indicates that j weakly dominates i, i.e. the lower bound produced by j is at least as large as i's. The bounds c_{ASGN} and c_{STAT} are adapted from static bin packing, while c_{PART} is obtained from the linear relaxation of an exponentially large set partition formulation; c_{CLQ}, c_{DEG}, and c_{MATCH} are novel bounds based on modeling TBP2 as a series of connected matchings.

The outline of the chapter is as follows. Section 4.2 summarizes relevant results from the literature. Next, in Section 4.3 we formulate TBP2 and present some preliminary results. In Section 4.4, we study clique instances and use them to derive a new lower bound. After that, Section 4.5 presents our IP formulation along with an additional formulation

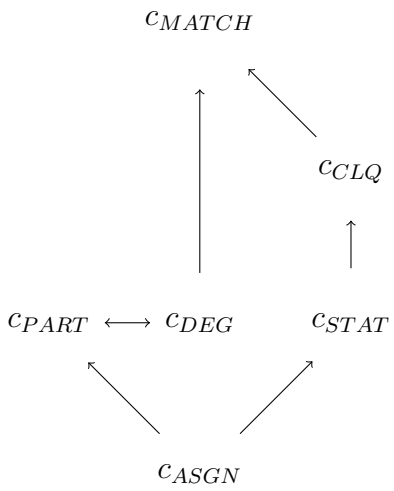

Figure 4.1: Graphical representation of Theorem 4.6.7. Arcs indicate that the bound at the tail is less than or equal to the bound at the head.

based on a set partitioning model. In Section 4.6, we provide our lower bound hierarchy, establishing the relative strengths of the discussed bounds. In Section 4.7, we present our computational study. We conclude in Section 4.8, while an appendix includes proofs omitted from the main body.

4.2 Literature Review

Static bin packing, or just bin packing, is among the most well-known NP-Complete problems [88]. Much of the bin packing literature focuses on approximation algorithms and their associated guarantees. The family of Fit algorithms is among the most studied, specifically First Fit and Best Fit [89]. If jobs are sorted in order of decreasing size, both the First Fit and Best Fit algorithms are tight $3/2$-approximations and no algorithm can have a better guarantee unless P = NP [90]; conversely, if jobs aren't sorted the algorithms are tight $17/10$-approximations [91]. Despite the negative approximability results in [90], [92] provide an algorithm with a worst-case additive gap of $O(\log^2(OPT))$ where OPT is the optimal number of bins. This method is based on solving the linear relaxation of the

Gilmore-Gomory formulation for the cutting stock problem [93]. More recently, [94] proposed an algorithm with a worst-case additive gap of $O(\log\log(OPT))$ using techniques from discrepancy theory.

In addition to its applications in approximation algorithms, the Gilmore-Gomory LP is also of note as it is conjectured that its optimal solution satisfies the *modified integer round-up* property, which suggests the number of bins needed in an optimal solution is at most the objective of the Gilmore-Gomory LP rounded up plus one. While the Gilmore-Gomory LP is known to provide strong bounds in practice, it requires the solution of an exponentially sized LP via column generation methods; research has aimed to find lower bounds with provable guarantees that are efficiently computable. For example, [95] propose multiple polynomially computable bounds arising from continuous relaxations of the problem, and the strongest is guaranteed to be at least $3/4$ of the optimal solution; [96] provides an alternate method of efficiently obtaining lower bounds through the use of dual feasible functions, and gives a bound with a $1/2$ multiplicative guarantee. For further references on bin packing we refer the reader to the surveys [97, 98].

The literature contains multiple notions of temporal bin packing. In our model the objective is to minimize the time-average number of active bins; an alternate model instead focuses on minimizing the total number of bins needed to pack all jobs. We first discuss results for the latter model, itself a special case of the more general vector bin packing model. In [99], the authors present a branch-and-price algorithm along with various methods for efficiently obtaining upper and lower bounds; [100] provide a matheuristic based on column generation methods. Other algorithms come from the vector bin packing literature and include branch-and-price [101], arc-flow approaches [102], and heuristics [103, 104, 105]. This TBP model has also been applied to problems in cloud scheduling. For instance, [106] use a variant to model a problem in virtual machine consolidation; [107] consider a variant in which the objective is a combination of both the total number of servers and the number of server "fire-ups", which [108] builds on by providing model reduction techniques and

an improved formulation.

The TBP model we consider, sometimes referred to as offline dynamic bin packing, has received attention as well. Most related to our work is the literature on the uniform job case, in which each bin has can accommodate g jobs simultaneously, where g is some positive integer. In [109], the authors show that TBP with uniform jobs is NP-Hard, even when $g = 2$, which corresponds to our model TBP2. The authors of [110] prove that a First Fit algorithm is a 4-approximation; [111] demonstrate how prior results on fiber minimization and line system design in [112, 113] yield two 2-approximations. In [114], the authors study special cases, providing a $\frac{gH_g}{H_g+g-1}$-approximation, where H_g is the g-th harmonic number, when all jobs intersect, and a $(2 - 1/g)$-approximation for *proper* instances, in which no job is properly contained within another.

The more general case in which jobs have non-uniform sizes is also of interest. The authors in [115] provide a 5-approximation and extend the result to a setting with flexible start times, while [116] describe a 4-approximation based on *dual coloring*. Most recently, [117] provide two algorithms with asymptotic approximation guarantees of $2\Pi_\infty$ and Π_∞, respectively, where the second algorithm has a larger additive term; the constant $\Pi_\infty \approx 1.69$ originates in the bin packing approximation results of [118]. In the context of cloud computing, [119] use an IP formulation to solve a variant in which all jobs have the same starting time.

4.3 Model Formulation and Preliminaries

Let $J = \{1, \ldots, n\}$ be a set of jobs. The temporal bin packing problem asks the decision maker to assign jobs to bins such that the time-average number of used bins is minimized, while respecting bin capacity. Each job $i \in J$ is specified by a start and end time $0 \leq s_i < e_i$. Without loss of generality, we assume that $\min_{i \in J} s_i = 0$ and $\max_{i \in J} e_i = 1$. In this work, we consider the case in which a bin can hold at most two jobs simultaneously and use the acronym TBP2 throughout for this problem; this special case of temporal bin

packing is known to be NP-Hard [109]. In the remainder of the chapter, we say a bin is *active* at time τ if it contains one or more jobs in that moment. A bin is *open* at time τ if it is active but not at capacity; i.e. it only has one job assigned in that instant.

It is useful to introduce a representation of TBP2 based on discretizing the time horizon; see e.g. [111]. Consider the increasing sequence of distinct start and end times $0 = \xi_0 < \xi_1 < \cdots < \xi_{t_{\max}}$, for index set $T = \{1, \ldots, t_{\max}\}$. For $t \in T$, the interval $I_t := [\xi_{t-1}, \xi_t)$ is the t-th period, with weight $w_t = \xi_t - \xi_{t-1}$. For some time point $\tau \in [0, 1]$, we use the notation $\tau \leq I_t$ to denote $\tau \leq \xi_{t-1}$, and $\tau \geq I_t$ for $\tau \geq \xi_t$, with similar notation for strict inequalities. We use $J(t)$ to denote the subset of jobs that are present during period t, that is, $J(t) = \{i \in J | [s_i, e_i) \cap I_t \neq \emptyset\}$.

Using this discretization, the problem can naturally be modelled as an IP in the style of [120]. Letting $\mathbb{B} = \{1, 2, 3..., b_{\max}\}$ be a set of bin indices where $b_{\max}$ is some sufficiently large number (such as n), we have the formulation

$$\min_{x,y} \sum_{b \in \mathbb{B}} \sum_{t \in T} w_t y_{b,t} \tag{4.1a}$$

$$\text{s.t.} \sum_{b \in \mathbb{B}} x_{i,b} = 1 \qquad \forall i \in J \tag{4.1b}$$

$$\sum_{i \in J(t)} x_{i,b} \leq 2 y_{b,t} \qquad \forall b \in \mathbb{B}, \forall t \in T \tag{4.1c}$$

$$x_{i,b}, y_{b,t} \in \{0, 1\}. \tag{4.1d}$$

The x variables denote job-to-bin assignments, and the y variables track when a bin is active. IP (4.1) also provides a means of obtaining a lower bound on the optimal time-average number of bins via its linear relaxation; we use c_{ASGN} to denote the objective of this relaxation.

Instead of minimizing the time-average number of bins, temporal bin packing can be equivalently thought of as maximizing the time-averaged *savings* of the solution [114]. Intuitively, the savings are the bins we do not open by assigning two jobs together. Letting

c^* be the original optimum, and z^* be the optimal time-average savings, based on our normalization to the time interval $[0, 1]$, these quantities are related by

$$c^* = \sum_{i \in J} (e_i - s_i) - z^*. \tag{4.2}$$

4.3.1 Static Bound

The bound c_{ASGN} is typically poor, motivating the need for stronger bounds that can be computed efficiently. For example, consider the following reformulation of IP (4.1),

$$\min_{x,y} \sum_{b \in \mathbb{B}} \sum_{t \in T} w_t y_{b,t} \tag{4.3a}$$

$$\text{s.t.} \sum_{b \in \mathbb{B}} x_{i,b,t} = 1 \qquad \forall i \in J, t \in T \tag{4.3b}$$

$$\sum_{i \in J(t)} x_{i,b,t} \leq 2 y_{b,t} \qquad \forall b \in \mathbb{B}, \forall t \in T \tag{4.3c}$$

$$x_{i,b,t-1} = x_{i,b,t} \qquad \forall i \in J, \forall b \in \mathbb{B}, \forall t \in T \setminus \{1\} \tag{4.3d}$$

$$x_{i,b,t}, y_{b,t} \in \{0, 1\}. \tag{4.3e}$$

IPs (4.1) and (4.3) are equivalent; the only difference is the expansion of the decision variables $x_{i,b}$ into $t_{\max}$ copies and the addition of temporal linking constraints (4.3d). A natural relaxation of (4.3) is to remove constraints (4.3d). If these constraints are removed, the problem decomposes into $t_{\max}$ static bin packing problems. In the general case, these sub-problems are themselves NP-Hard, but in practice solving the decomposed problems is easier than solving IP (4.1). In our case, the sub-problems admit an analytic solution, and we obtain the bound

$$c_{STAT} = \sum_{t \in T} w_t \lceil |J(t)|/2 \rceil. \tag{4.4}$$

As (4.4) is obtained by a relaxation, $c_{STAT} \leq c^*$; we refer to c_{STAT} as the *static bound*. For TBP with uniform job sizes, the solution to each sub-problem is obtained by rounding

the ratio of the period demand to bin capacity, and thus the static bound can viewed as a natural temporal extension of the L_1 bound for static bin packing [95]. It has also been previously called the *demand profile* bound in [111]. The static bound can be computed in $O(n \log(n))$ time.

4.3.2 Three-Period Instance

In this section we focus on the special case $t_{\max} = 3$, the simplest case in which $c_{STAT} < c^*$ is possible. Consider the following example, adapted from [113]: let $J = \{1, 2, 3\}$, $s_1 = s_2 = 0, s_3 = (1-\epsilon)/2$, and $e_1 = (1+\epsilon)/2, e_2 = e_3 = 1$, so that $w = ((1-\epsilon)/2, \epsilon, (1-\epsilon)/2)$. Then, $c_{STAT} = 1 + \epsilon$ while $c^* = 3/2 + \epsilon/2$, and therefore $c^*/c_{STAT} \to 3/2$ as $\epsilon \to 0$.

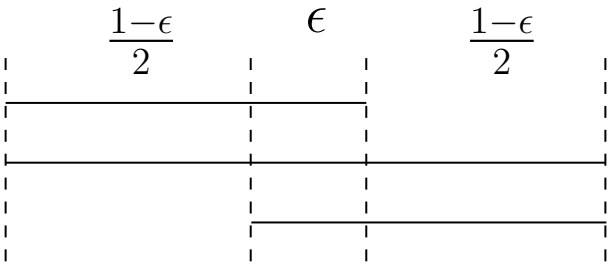

Figure 4.2: Three-period example.

We establish that this is the worst the static bound can do for any three-period instance with the following proposition.

Proposition 4.3.1. *In a three-period instance,* $c^* - c_{STAT} \leq 1/2$.

Proof. The proof is in Appendix A.1 □

4.3.3 Worst-Case Additive Gap

Although $c^* - c_{STAT} \leq 1/2$ for three-period instances, we now show that no such constant additive gap holds in general. We consider a subset of instances in which all jobs overlap; these kinds of instances are discussed in more detail below and in Section 4.6. Specifically, for a fixed $t_{\max}$, specify some period $\hat{t} \in T$. Create a job for each possible start, end

period pair before and after period $\hat{t}$, respectively. Abusing notation to denote as (t_1, t_2) the job that starts in period t_1 and ends after period t_2, we create a job (t_1, t_2) for each $(t_1, t_2) \in \{1, 2, .., \hat{t}\} \times \{\hat{t}, \hat{t}+1, ..., t_{\max}\}$. An example with $t_{\max} = 7, \hat{t} = 4$ is shown in Figure 4.3. Given $t_{\max}$ and $\hat{t}$, the total number of jobs is $n = \hat{t}(1 + t_{\max} - \hat{t})$. We use this

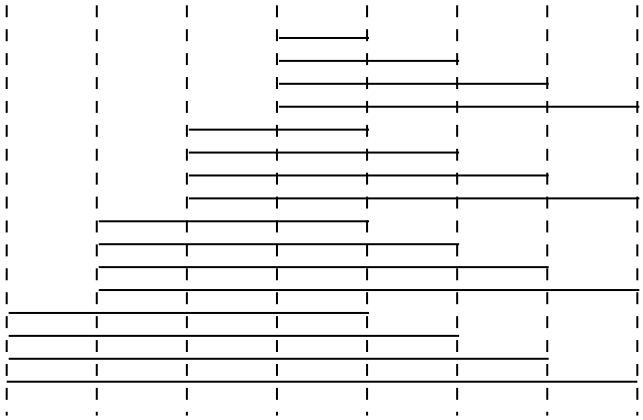

Figure 4.3: Example of instance from family with unbounded additive gap, where $t_{\max} = 7, \hat{t} = 4$.

family of instances to establish the following proposition.

Proposition 4.3.2. *There is no constant $\beta \geq 0$ such that $c^* - c_{STAT} \leq \beta$ for all instances.*

The proof is in Appendix A.2.

4.4 Clique Instances

The examples in Sections 4.3.2 and 4.3.3 belong to a broader class of instances; in this section we focus on this special case and use it to derive a lower bound method. A clique instance is specified by some time point at which all jobs overlap, i.e. the instance's conflict graph representation forms a clique. The conflict graph has a node for each job $i \in J$ and edges for each pair $i, j \in J$ that overlaps. For a clique instance with jobs C, we denote the time point at which all jobs overlap as τ_C; there can be infinitely many points defining the same clique, and we are indifferent to which of these points is chosen. Clique instances have been studied previously, see e.g. [114].

Because all jobs overlap, the problem reduces to a static packing; to guarantee feasibility the decision maker only needs to consider the packing at time τ_C. As only two jobs can be placed on the same bin, the problem of minimizing the time-average number of active bins reduces to that of obtaining a minimum-cost perfect matching in the conflict graph with edge costs $c_{i,j} = \max(e_i, e_j) - \min(s_i, s_j)$, for $i, j \in J$. If n is odd, we augment the graph with a dummy job with $s_i = e_i = \tau_C$.

The problem of finding a minimum-cost perfect matching is well-studied and is known to be efficiently solvable [121], but the standard matching approaches don't make use of the additional structure present in our problem. In [114], the authors show that if an instance has either $s_i = \tau_C$ or $e_i = \tau_C$ for all jobs $i \in J$, a First-Fit algorithm is optimal and $c_{STAT} = c^*$; we refer to these instances as one-sided instances. They also demonstrate that the well-known greedy H_g-approximation algorithm for set cover gives a $6/5$-approximation for clique instances.

We present an algorithm that generalizes the intuition used to solve one-sided instances.

Proposition 4.4.1. *For a clique instance,* $c^*/c_{STAT} \leq 3/2$.

Proof. Partition T into two groups T_- and T_+, where $T_- = \{t \in T | I_t < \tau_C\}$ and $T_+ = \{t \in T | I_t \geq \tau_C\}$. We can do this without loss of generality, as we can always split a period into two periods such that T_- and T_+ exactly partition T. Now consider the static bound over the periods before and after τ_C; let these be $c_- = \sum_{t \in T_-} w_t \lceil |J(t)|/2 \rceil$ and $c_+ = \sum_{t \in T_+} w_t \lceil |J(t)|/2 \rceil$, respectively. Note that $c_- + c_+ = c_{STAT}$.

Without loss of generality, assume that $c_- \geq c_+$; otherwise, invert the horizon. Consider the First-Fit algorithm, which pairs jobs in order of increasing start time. Let c^{FF} be the cost of the resulting solution, and let c_-^{FF} and c_+^{FF} be the solution's cost over T_- and T_+, respectively. When only considering jobs in T_-, the instance is one-sided; consequently,

the First-Fit solution is optimal within T_- and $c_-^{FF} = c_-$. Therefore,

$$c^{FF} = c_-^{FF} + c_+^{FF} = c_- + c_+^{FF} \leq c_- + \sum_{t \in T_+} w_t |J(t)|$$

$$\leq c_- + 2 \sum_{t \in T_+} w_t \lceil |J(t)|/2 \rceil = c_- + 2c_+ = c_{STAT} + c_+ \leq 1.5 c_{STAT}.$$

The first inequality follows as $|J(t)|$ is an upper bound on any solution in any period; the final one because $c_- \geq c_+$. □

This result implies that the First-Fit heuristic described in the proof is also a a $3/2$-approximation for clique instances.

4.4.1 Clique Bound

The static bound is obtained by allowing repacking between periods; cliques can be used in a similar manner, allowing repacking between cliques instead of periods. An instance's conflict graph can thus be decomposed into its maximal cliques, with each clique sub-instance solved separately.

Consider the conflict graph corresponding to jobs J; let m be the number of maximal cliques and $\mathbb{C} = \{C_1, C_2, \ldots, C_m\}$ denote the ordered set of maximal cliques arranged by increasing times τ_{C_i}. The clique bound is obtained by solving a sub-problem induced by each clique $C \in \mathbb{C}$ and then summing the objectives. To define the clique sub-problems, we need to specify breakpoints that mark where one clique ends and the next begins; these points specify where the repacking is allowed.

For clique C_i, define

$$s_{C_i} = \min_{j \in C_i \setminus C_{i-1}} s_j, \qquad e_{C_i} = \max_{j \in C_i \setminus C_{i+1}} e_j,$$

where $C_0 = C_{m+1} = \emptyset$. For consecutive cliques C_{i-1} and C_i, the breakpoint must satisfy $\gamma_i \in [e_{C_{i-1}}, s_{C_i})$; choosing breakpoints in this manner ensures that, for consecutive

breakpoints γ_{i-1}, γ_i, the instance created by truncating the horizon to the interval $[\gamma_{i-1}, \gamma_i]$ is a clique instance. For simplicity, we define $\gamma_0 = 0$ and $\gamma_m = 1$. Now assume that we are given feasible breakpoints $\gamma_i \in [e_{C_{i-1}}, s_{C_i})$, for each consecutive clique pair; let $c^*(C_i, \gamma_{i-1}, \gamma_i)$ be the optimal cost of the instance given by C_i over the interval $[\gamma_{i-1}, \gamma_i]$, weighted by the length of interval $[\gamma_{i-1}, \gamma_i]$. The clique lower bound for this γ is then

$$c_{CLQ}(\gamma) = \sum_{C_i \in \mathbb{C}} c^*(C_i, \gamma_{i-1}, \gamma_i). \tag{4.6}$$

Assume first that the clique breakpoints are given. Let $c_{STAT}(\gamma_{i-1}, \gamma_i)$ denote the static bound for periods contained in the interval $[\gamma_{i-1}, \gamma_i]$, splitting periods if necessary.

Proposition 4.4.2. *$c_{STAT} \leq c_{CLQ}(\gamma)$ for any feasible choice of γ.*

Proof. We have $c_{CLQ}(\gamma) = \sum_{C_i \in \mathbb{C}} c^*(C_i, \gamma_{i-1}, \gamma_i) \geq \sum_{C_i \in \mathbb{C}} c_{STAT}(\gamma_{i-1}, \gamma_i) = c_{STAT}$. The final equality follows as the breakpoints γ define a full partition of the horizon. □

Proposition 4.4.2 confirms that the clique bound is at least as good as the static bound regardless of the choice of breakpoints.

Moreover, the choice of breakpoints can affect the strength of the clique bound. Before considering the optimization of the breakpoints, we analyze the cost of an individual clique as a function of the breakpoints. Assume that for some $C_i \in \mathbb{C}$ and γ we have a feasible solution, represented as some matching $\mu \in M_i$, where M_i is the set of perfect matchings in C_i's conflict graph with the addition of a single dummy job if $|C_i|$ is odd; we can express the cost of this solution as a function of the clique's breakpoints. Let c_μ be the weighted cost of the matching truncated to the interval $[s_{C_i}, e_{C_i}]$. The cost of the matching as a function of μ, γ is

$$c(\mu, \gamma_{i-1}, \gamma_i) = c_\mu + (s_{C_i} - \gamma_{i-1})\phi_\mu^- + (\gamma_i - e_{C_i})\phi_\mu^+,$$

where ϕ_μ^- is the number of bins spanning $[\gamma_{i-1}, s_{C_i}]$ and ϕ_μ^+ the number of bins spanning

$[e_{C_i}, \gamma_i]$ given the matching μ. Therefore,

$$c^*(C_i, \gamma_{i-1}, \gamma_i) = \min_{\mu \in M_i} \{c_\mu + (s_{C_i} - \gamma_{i-1})\phi_\mu^- + (\gamma_i - e_{C_i})\phi_\mu^+\}.$$

As M_i is finite, the right-hand side is the minimum over a finite number of affine functions; therefore, it is a piecewise linear concave function of γ. Consequently, the problem of optimizing the clique bound can be expressed as maximizing a sum of piecewise linear concave functions, and formulated as

$$\max_{\sigma,\gamma \geq 0} \sum_{i=1}^{m} \sigma_i \tag{4.7a}$$

$$\sigma_1 \leq c_\mu + (\gamma_1 - e_{C_1})\phi_\mu^+ \qquad \forall \mu \in M_1 \tag{4.7b}$$

$$\sigma_m \leq c_\mu + (s_{C_{|C|}} - \gamma_{m-1})\phi_\mu^- \qquad \forall \mu \in M_m \tag{4.7c}$$

$$\sigma_i \leq c_\mu + (\gamma_i - e_{C_i})\phi_\mu^+ + (s_{C_i} - \gamma_{i-1})\phi_\mu^- \quad \forall i \in \{2, ..., m-1\} \forall \mu \in M_i \tag{4.7d}$$

$$e_{C_i} \leq \gamma_i \leq s_{C_{i+1}} \qquad \forall i \in \{1, 2, ..., m-1\}. \tag{4.7e}$$

For each clique C_i, M_i may contain exponentially many perfect matchings; consequently, (4.7) may contain exponentially many constraints. Nevertheless, we can optimize (4.7) efficiently using constraint generation, as described in the following proposition.

Proposition 4.4.3. *The separation problem for* (4.7) *can be solved by computing* $m = O(n)$ *perfect matchings, and thus* (4.7) *can be solved in polynomial time.*

Proof. Assume we are given a candidate solution (σ^*, γ^*) to (4.7); for each clique $C_i \in \mathbb{C}$ we check if there is a matching $\mu \in M_i$ with $c(\mu, \gamma^*_{i-1}, \gamma^*_i) < \sigma^*_i$. This is done by fixing the value of γ^* and then solving the corresponding clique problem over the interval $[\gamma^*_i, \gamma^*_{i+1}]$. As discussed in the previous section, this can be done by solving a minimum-cost perfect matching. For C_i, if we obtain a matching μ' with $c(\mu', \gamma^*_{i-1}, \gamma^*_i) < \sigma^*_i$, we add constraint

$$\sigma_i \leq c_{\mu'} + (\gamma_i - e_{C_i})\phi_+^{\mu'} + (s_{C_i} - \gamma_{i-1})\phi_-^{\mu'}.$$

There are $O(n)$ maximal cliques and they can be determined in $O(n)$ time; therefore, for any algorithm that computes a minimum-cost perfect matching on a graph with n nodes in $P(n)$ time, a full round of the separation routine runs in $O(nP(n))$ time. Since there are polynomial-time algorithms for minimum-cost perfect matching, the result follows from the equivalence of separation and optimization. □

We use c_{CLQ} to denote the optimal objective of (4.7), and refer to it as the clique bound. The practical performance of the constraint generation algorithm can be improved by noting that each solve of the (4.7) provides a feasible choice of breakpoints γ^* and an upper bound on c_{CLQ}. During the separation routine, as each clique is checked for an improving matching, each iteration computes a lower bound given the current set of breakpoints. The algorithm can be terminated if the current upper bound and lower bounds are sufficiently close.

4.5 New Formulations

In this section we provide novel IP formulations for TBP2. Our main formulation uses the fact that at most two jobs can be packed simultaneously within a bin, implying the total savings in the bin are equal to the total overlap of jobs within the bin. Let $O = \{(i,j) \mid \forall i,j \in J, [s_i,e_i) \cap [s_j,e_j) \neq \emptyset\}$ be the set of overlapping job pairs, and consider the formulation

$$\max_{\rho} \sum_{i,j \in O} o_{i,j}\rho_{i,j} \tag{4.8a}$$

$$\text{s.t.} \sum_{j \in C | j \neq i} \rho_{i,j} \leq 1 \qquad \forall C \in \mathbb{C}, \forall i \in C \tag{4.8b}$$

$$\rho_{i,j} \in \{0,1\}, \qquad i,j \in O, \tag{4.8c}$$

where $o_{i,j} = \min(e_i,e_j) - \max(s_i,s_j)$ is the measure of overlap for an overlapping pair. The ρ variables represent the decision to pair jobs i,j in the same bin, which only considers

jobs that have non-zero overlap. Constraints (4.8b) ensure that a job can only be paired with a single other job at a time. We refer to the above formulation as the *matching* formulation, as it models TBP2 as a set of connected matching problems. We argue for the correctness of this formulation in the following proposition.

Proposition 4.5.1. *IP* (4.8) *is a valid formulation for TBP2.*

Proof. The objective maximizes the overlap of jobs paired together in the solution. This is equivalent to maximizing the savings, which by (4.2) is equivalent to minimizing the assignment's cost. It then suffices to argue that a solution of (4.8) is feasible if and only if it corresponds to a feasible solution for TBP2. First, assume that we are given a feasible solution to TBP2. That is, consider a partition of J; for each $B \subseteq J$ in this partition, set $\rho_{i,j} = 1$ for each $i, j \in B \cap O$, and 0 otherwise. This creates a feasible solution to (4.8); constraint (4.8b) is not violated, as we assumed a feasible partition of J and each set B in this partition never packs more than two job simultaneously.

Now assume we are given a solution ρ for (4.8). Consider the graph given by nodes representing jobs, and edges where the corresponding ρ variables are equal to 1. Define a solution of TBP2 by assigning nodes in a connected component of this graph to the same bin; clearly every job belongs to some component. Let B be the node set of one of these components; B is a feasible assignment, as constraints (4.8b) ensure that at most two jobs in the same clique are in the same component. Observe that we only need to consider overlap at times corresponding to maximal cliques because if jobs overlap at any point, they are members of at least one common maximal clique. □

IP (4.8) is similar in size to IP (4.1), having $O(n^2)$ variables and $O(nm)$ constraints compared to $O(nb_{\max})$ variables and $O(\max(n, t_{\max}b_{\max}))$ constraints; however, (4.8) is typically somewhat larger given a reasonable choice of $b_{\max}$. One advantage of (4.8) over (4.1) is that it does not exhibit the same level of symmetry. Any feasible solution x, y to (4.1) can be transformed into an equivalent solution by permuting x, y along their bin

indices. We further compare the formulations' strength theoretically in Section 4.6 and empirically in Section 4.7.

As with (4.1), we obtain a bound from the linear relaxation of (4.8). We define z_{DEG} as the optimal objective value for the linear relaxation of IP (4.8) and c_{DEG} as the equivalent value converted to time-average bins via (4.2); we refer to the latter as the *degree* bound.

The formulation (4.8) can be interpreted as a sequence of linked matching problems. Within each clique $C \in \mathbb{C}$, the constraints (4.8b) are identical to degree constraints in a matching formulation. As such, for each clique $C \in \mathbb{C}$ we can include valid inequalities from the corresponding matching polytope; specifically, we can add the well-known blossom inequalities [121]:

$$\sum_{i,j \in S} \rho_{i,j} \leq \frac{|S|-1}{2} \quad \forall S \subseteq C, \quad |S| \text{ odd.} \tag{4.9}$$

While there are exponentially many blossom inequalities, we can separate over the collection for a single clique in polynomial time [122], and there are $O(n)$ maximal cliques. We use z_{MATCH} to refer to the optimal fractional solution to (4.8) including blossom inequalities, and c_{MATCH} as the equivalent time-average number of bins via (4.2). We call the latter of these the *matching* bound.

4.5.1 Partition Formulation

Before continuing, we present one additional formulation based on the a transformation to a set partition model,

$$\min_{\eta} \sum_{S \in \mathbb{S}} \ell_S \eta_S \tag{4.10a}$$

$$\text{s.t.} \sum_{S \in \mathbb{S}(i)} \eta_S = 1 \qquad i \in J \tag{4.10b}$$

$$\eta_S \in \{0,1\}, \qquad S \in \mathbb{S}, \tag{4.10c}$$

where $\mathbb{S}$ is the set of all subsets of J that can be placed in a single bin, and ℓ_S is the time-averaged active time of a bin with jobs S. The variables η represent the yes-or-no decision to use a bin containing exactly S. IP (4.10) is potentially exponentially large, with $O(2^n)$ variables; solving it requires special tools such as branch-and-price; the linear relaxation of (4.10) can be solved using column generation. We use c_{PART} to denote the objective of the linear relaxation of (4.10) and z_{PART} to be the equivalent optimal time-averaged savings via (4.2).

4.6 Comparison of Bounds

In this section we compare the theoretical performance of our previously discussed bounds: c_{ASGN}, c_{STAT}, c_{PART}, c_{DEG}, and c_{MATCH}. First we argue the relative weakness of c_{ASGN}.

Proposition 4.6.1. $c_{ASGN} \leq c_{STAT}$.

Proof. Consider a feasible, potentially fractional assignment of the x variables in IP (4.1). In each period t, a feasible solution satisfies $\sum_{i \in J(t)} \sum_{b \in \mathbb{B}} x_{i,b} = |J(t)|$. A feasible, fractional value for the y variables is $y_{t,b} = \sum_{i \in J(t)} \frac{x_{i,b}}{2}$. The total cost incurred in this period by this solution is $w_t \sum_{b \in \mathbb{B}} y_{t,b} = w_t \sum_{b \in \mathbb{B}} \sum_{i \in J(t)} x_{i,b}/2 = w_t |J(t)|/2 \leq w_t \lceil |J(t)|/2 \rceil$. $\square$

Proposition 4.6.2. $c_{ASGN} \leq c_{PART}$.

Proof. We show this by arguing that any solution to the linear relaxation of IP (4.10) implies an equivalent fractional solution to IP (4.1). Let η be a feasible fractional solution. Associate with each $\eta_S > 0$ some bin index $b \in \mathbb{B}$ such that each η_S is assigned a unique bin; this can always be done as we can add an arbitrary number of bins to IP (4.1) without altering the objective. We construct a solution x, y by taking $x_{i,b} = \eta_S$ for $i \in S$ and $y_{t,b} = \eta_S$ for t with $S \cap J(t) \neq \emptyset$; therefore, the bin index b corresponding to S accrues a cost of $\sum_{t \in T} y_{t,b} = \ell_S \eta_S$. The job assignment constraints are satisfied, as $\sum_{b \in \mathbb{B}} x_{i,b} = \sum_{S \in \mathbb{S}(i)} \eta_S = 1$. Capacity constraints are respected as, for each b, S pair, at

most two jobs, each with weight η_S, are present in each period and the right hand side of the capacity constraint is $2y_{b,t} = 2\eta_S$ whenever the bin is active during period t. □

These results are not surprising; even in static bin packing, the equivalent of c_{ASGN} is known to give poor bounds. Next, we show that c_{DEG}, c_{STAT} and c_{CLQ} are incomparable.

Proposition 4.6.3. *The bounds c_{DEG} and c_{STAT} are incomparable; that is, there exist instances in which one bound is larger than the other.*

Proof. We show this by providing two examples, one in which $c_{DEG} > c_{STAT}$ and one where $c_{DEG} < c_{STAT}$. For the former, consider a variation of the three-period example given in Figure 4.2 with $w_1 = 1/2, w_2 = w_3 = 1/4$. In this case $c_{DEG} = c^* = 3/2$ and $c_{STAT} = 5/4$.

For the latter, consider an instance with $n = 3$ and all jobs having $s_i = 0$ and $e_i = 1$. This case reduces to a static bin packing problem with bins that fit two jobs, and we have $c_{STAT} = c^* = 2$ and $c_{DEG} = 3/2$. □

We have a similar result for c_{DEG} and c_{CLQ}.

Proposition 4.6.4. *The bounds c_{DEG} and c_{CLQ} are incomparable.*

Proof. As $c_{CLQ} \geq c_{STAT}$, we only need show an example with $c_{DEG} > c_{CLQ}$. Consider an instance with $n = 4$, $(s_1, e_1) = (0, 3/7), (s_2, e_2) = (0, 1), (s_3, e_3) = (2/7, 1)$, and $(s_4, e_4) = (4/7, 5/7)$; see Figure 4.4. In this example, $c_{DEG} = c^* = 11/7$ and $c_{CLQ} = 9/7$. □

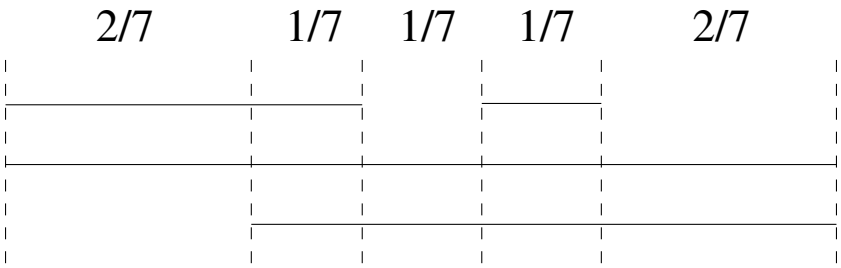

Figure 4.4: Example with $c_{DEG} > c_{CLQ}$.

Next, we demonstrate that the clique bound is weaker than the linear relaxation of (4.8) strengthened with blossom inequalities.

Proposition 4.6.5. $c_{CLQ} \leq c_{MATCH}$.

Proof. Assume we are given the clique bound c_{CLQ} and the optimal choice of breakpoints γ. Consider the LP

$$\max_{\rho \geq 0} \sum_{C \in \mathbb{C}} \sum_{i,j \in C} o_{i,j}^{C} \rho_{i,j}^{C} \tag{4.11a}$$

$$\text{s.t.} \sum_{j \in C | j \neq i} \rho_{i,j}^{C} \leq 1 \qquad \forall C \in \mathbb{C}, \forall i \in C \tag{4.11b}$$

$$\sum_{i,j \in S} \rho_{i,j}^{C} \leq \frac{|S| - 1}{2} \qquad \forall C \in \mathbb{C}, \forall S \subseteq C, |S| \; odd \tag{4.11c}$$

$$\rho_{i,j}^{C} = \rho_{i,j}^{D} \qquad \forall C, D \in \mathbb{C}, \tag{4.11d}$$

where $o_{i,j}^{C_i} = \min(\gamma_{C_i}, e_i, e_j) - \max(\gamma_{C_{i-1}}, s_i, s_j)$ is the overlap of jobs i, j truncated to the interval $[\gamma_{i-1}, \gamma_i]$. LP (4.11) has optimal objective z_{MATCH}. Now consider the relaxation in which constraints (4.11d) are removed. The LP now decomposes into m sub-problems; however, for each clique the resulting sub-problem is the maximization over a matching polytope, i.e., it yields integral solutions. Consequently, each of these sub-problems yields the same solution as the sub-problems used to compute the clique bound. The result then follows by converting the objective of the relaxed LP (4.11) to time-average number of bins. □

Next, we show that the degree bound is equivalent to the linear relaxation of the partition formulation; this proof is slightly longer and relegated to Appendix A.3.

Proposition 4.6.6. $c_{PART} = c_{DEG}$.

Proof. The proof is in Appendix A.3 □

This result is somewhat surprising; typically, this type of set partition formulation yields strong lower bounds compared to polynomially-sized formulations; however, here it is equivalent to the linear relaxation of (4.8) without any additional constraints. Furthermore, c_{DEG} itself sometimes provides worse bounds than the simple c_{STAT} bound.

We now state the main result of this section, which summarizes our lower bound hierarchy. The result is also depicted visually in Figure 4.1.

Theorem 4.6.7. *The following statements hold:*

1. $c_{ASGN} \leq c_{STAT}$.
2. $c_{ASGN} \leq c_{PART}$.
3. $c_{STAT} \leq c_{CLQ}$.
4. $c_{CLQ} \leq c_{MATCH}$.
5. $c_{DEG} = c_{PART}$.
6. $c_{DEG} \leq c_{MATCH}$.
7. c_{DEG} *and* c_{STAT} *are incomparable.*
8. c_{DEG} *and* c_{CLQ} *are incomparable.*

Proof. The result follows from Propositions 4.4.2, 4.6.1, 4.6.2, 4.6.3, 4.6.4, 4.6.5, and 4.6.6, along with the fact that c_{DEG} is obtained by relaxing the LP that produces c_{MATCH}. □

While this result establishes theoretical guarantees on how the various bounds perform relative to one another, it does not explain how large the gaps between bounds are in practice, or how the incomparable bounds perform empirically. Similarly, while all of the bounds can be computed in polynomial time, the stronger bounds tend to require more computational effort. We explore the computational performance of these bounds next.

4.7 Computational Study

In this section we report results from a computational study of our bounds and formulations. Our objective is twofold. First, we compare the matching-based formulation (4.8) against the more standard (4.1). Second, we assess the empirical performance of the various bounds

we compared theoretically in Section 4.6, with a focus on bound quality and scalability. We do not include c_{PART} and c_{ASGN} in this comparison; by Theorem 4.6.7, $c_{PART} = c_{DEG}$ and $c_{ASGN} \leq c_{STAT}$, and preliminary experiments suggest that these methods scale worse than c_{DEG} and c_{STAT}, respectively.

Except where noted, we conducted all experiments on a computer running Windows with a 3.20GHz Intel 6 Core i7 processor and 16 GB of RAM. We used Python 3.9.7 to build our experimental code, and Gurobi 9.5.1 for LP and IP solves, with default parameters unless otherwise stated. We used the NetworkX package for matching sub-problems and to separate blossom inequalities [123, 124]. In the following subsections, we describe and summarize the results of individual experiments, but first we describe various heuristic methods used for upper bounds.

4.7.1 Heuristics

Next, we briefly discuss the heuristics we used to obtain upper bounds in our experiments. The first of these methods comes from IP (4.8). We solve the LP relaxation with blossom inequalities, equivalent to computing c_{MATCH}, and then solve the resulting model as an IP with some fixed time limit. Our preliminary experiments suggest that if the LP can be solved quickly, the solver can also find good solutions in a reasonable time even if it cannot prove optimality.

For instances in which the above approach is impractical, e.g. when computing c_{MATCH} is intractable, we use a combination of a constructive heuristic and a local search heuristic. The constructive heuristic iteratively solves a packing problem that aims to place as much volume into a single bin as possible. Formally, starting with an empty set of bins $\mathcal{B}$ and a

set of jobs J, we solve the problem

$$\max_x \sum_{i \in J} \tilde{w}_i x_i \tag{4.12a}$$

$$\text{s.t.} \sum_{i \in C} x_i \leq 2 \qquad \forall C \in \mathbb{C} \tag{4.12b}$$

$$x_i \in \{0, 1\}, \tag{4.12c}$$

where $\tilde{w}_i$ is some appropriately chosen weight for job i, such as its length. This IP can be solved as a linear program, because its constraint matrix satisfies the consecutive ones property. Letting x be the optimal solution and $S = \{i \in J : x_i = 1\}$, we then set $\mathcal{B} = \mathcal{B} \cup \{S\}$ and $J = J \setminus S$. The process repeats until $J = \emptyset$, which occurs when every job belongs to some bin. In our implementation, instead of a job's length, we use $\tilde{w}_i = (e_i - s_i)^2$, where s_i and e_i are integer start and end times before we normalize the instance; this appears to improve performance based on preliminary testing.

We then improve the resulting solution through local search, which checks for all pairs of bins in $\mathcal{B}$ if there exists a better solution also using two bins. Given two bins $B_1, B_2 \in \mathcal{B}$, our implementation does this by solving (4.1) with $b_{\max} = 2$ and jobs in $B_1 \cup B_2$. This process repeats until either we find no pair of improving bins or reach a specified time limit.

4.7.2 Instance Design

We design synthetic instances using three integer parameters: the number of jobs $n > 0$, a maximum start time $s_{\max} > 0$, and the expected lifetime $\lambda > 0$. We construct an instance with n jobs by sampling for each job i a start time s_i, using a uniform integer distribution over $[0, s_{\max}]$, and a lifetime α_i, using a geometric distribution with success probability $1/\lambda$; that is, $e_i = \alpha_i + s_i$. We normalize the horizon to $[0, 1]$ by dividing start and end times by $\max_{i \in J} e_i$. For our experiments, we take $s_{\max} = n$; thus the density of each instance is controlled solely by the parameter λ.

We also obtain instances from a Microsoft Azure system that focuses on supporting machine learning applications. In this system, jobs either occupy 100%, 50%, or 25% of a server; for our experiments, we modify the instance to assume each job occupies 50%. We generate an instance by sampling roughly three months of arrival and departure data in a cluster; we perform this sampling over multiple server clusters to generate different instances. When sampling, some virtual machines are already present at the start of the sampled period, in which case we include their full lifetime in the instance.

4.7.3 IP Formulation Comparison

In the first experiment, we compare the performance of (4.1) and (4.8). We generate random synthetic instances as described, using $n = s_{\max} \in \{200, 400, 600\}$ and $\lambda \in \{5, 10, 20, 40, 80\}$, leading to a total of 15 instances classes. For each of the classes, we generate five instances, and for each instance we attempt to solve both formulations with a 600-second time limit. After each solve, we collect the best upper and lower bounds (converting bounds from (4.8) to time-averaged bins) and the resulting relative optimality gaps. We report the averages of the upper and lower bounds and the geometric means of the relative gaps at termination for each instance class in Table 4.1.

In each row of the table, we highlight the best average upper and lower bounds. On average, (4.8) obtains both better upper and lower bounds compared to (4.1), for all instance classes. Consequently, (4.8) also leads to significantly improved gaps compared to (4.1). For all 200- and 400-job instances, (4.8) has average gaps under 1% while (4.1) has gaps in the range of 5% to 8%. Even for the worst parameter combination for (4.8), 600 jobs with expected lifetime of 80, the average gap is only 1.53%. Additionally, for (4.8) gaps tend to increase with density, while for (4.1) the gaps are fairly consistent across both size and density.

$n = s_{\max}$	λ	IP (4.8) UB	LB	Gap	IP (4.1) UB	LB	Gap
200	5	**3.13**	**3.13**	0.00%	3.14	2.93	6.65%
	10	**5.08**	**5.08**	0.00%	5.15	4.81	6.57%
	20	**8.41**	**8.39**	0.25%	8.53	8.08	5.18%
	40	**9.51**	**9.46**	0.54%	9.75	9.23	5.39%
	80	**13.42**	**13.38**	0.30%	13.81	13.09	5.17%
400	5	**3.28**	**3.28**	0.00%	3.30	3.07	6.78%
	10	**5.66**	**5.66**	0.00%	5.80	5.35	7.75%
	20	**9.75**	**9.68**	0.72%	9.97	9.38	5.92%
	40	**15.54**	**15.40**	0.87%	15.99	15.11	5.53%
	80	**21.94**	**21.76**	0.81%	22.78	21.44	5.94%
600	5	**3.25**	**3.25**	0.00%	3.27	3.06	6.32%
	10	**5.43**	**5.43**	0.03%	5.55	5.11	7.93%
	20	**9.72**	**9.64**	0.85%	9.95	9.34	6.10%
	40	**17.13**	**16.91**	1.27%	17.66	16.62	5.92%
	80	**25.85**	**25.45**	1.53%	27.03	25.14	6.99%

Table 4.1: IP (4.1), IP (4.8) averaged results on random instances with a 600s time limit.

4.7.4 Lower Bound Comparison

We now report on two experiments comparing the performance of the bounds discussed in Section 4.6. First, we compare the bounds on instances with average job lengths $\lambda \in \{5, 10, 20\}$, which are relatively sparse. We use $n = s_{\max} \in \{800, 1{,}600, 2{,}400\}$, and for each of the nine classes we generate five instances. For each realization, we compute $c_{STAT}, c_{CLQ}, c_{DEG}$, and c_{MATCH}, and record the bound value and the computation time. We use dual simplex in the LP solves when computing both c_{DEG} and c_{MATCH}. To obtain upper bounds, we use (4.8) with blossom inequalities as described in Section 4.7.1, with an 1,800-second time limit for instances with $\lambda = 5, 10$ and a 3,600-second time limit for $\lambda = 20$; we increase the time to compensate for the added difficulty at the higher density. We report the averages of the running times and the geometric means of the resulting gaps relative to the upper bound in Table 4.2.

In terms of empirical performance, the results suggest that $c_{DEG} \leq c_{STAT} \leq c_{CLQ} \leq c_{MATCH}$ in sparse instances. For all instances, c_{MATCH} has an average gap of at most

0.12%. The next best bound, c_{CLQ}, results in gaps within the range of 0.85% to 1.05%. The clique bound c_{CLQ} improves on c_{STAT} by roughly 0.6% in absolute terms for the sparsest instances, the difference decreasing with density. The static bound improves on the degree bound by about 2% in absolute terms for the sparsest instances, and the difference again decreases with density. In general, the bounds grow closer and that the resulting gaps decrease when the density increases. This is not entirely surprising, as at maximum density the instance returns to a static bin packing problem where, for a sufficiently large number of jobs, all of the methods give optimal or near-optimal lower bounds.

The presented gaps are also a function of upper bound quality. For this experiment, the upper bound quality decreases as the density and number of jobs increase. We are able to find optimal solutions for instances with $\lambda = 5$, upper bounds on average roughly 0.01% from optimal for $\lambda = 10$, and at most 0.12% from optimal for $\lambda = 20$.

With respect to time, the computational effort increases with both size and density. As expected, c_{STAT} is the fastest to compute, followed by c_{CLQ}, c_{DEG}, and c_{MATCH}. The clique bound can be computed on average in less than five seconds for sparse instances; c_{DEG} solves within a few seconds for instances with $\lambda = 5, 10$, but takes over two minutes for instances with $\lambda = 20$. Conversely, c_{MATCH} takes significantly more time, between approximately three and ten minutes on average for the instances with $n = 2,400$. We observe that c_{DEG}, c_{MATCH} exhibit the worst scaling both in terms of number of jobs and density.

We now report on experiments using denser instances. Given the poor scaling of c_{MATCH}, we introduce an approximate version, $\hat{c}_{MATCH}$, in which we set a time limit of 3,600 seconds, meaning we may terminate before separating all necessary blossom inequalities. As before, we use dual simplex in our LP solves. We define instance classes with $n = s_{\max} \in \{800, 900, 1,000\}$ and $\lambda \in \{40, 80\}$, again generating five instances in each class. We compute upper bounds using the constructive heuristic and local search algorithm discussed in Section 4.7.1, with the local search running until convergence; this

takes less than 1,600 seconds per realization. We summarize results in Table 4.3.

For these dense instances, $c_{STAT} \leq c_{CLQ} \leq c_{DEG} \leq \hat{c}_{MATCH}$; interestingly, the degree bound is stronger than the clique and static bounds in this case. The strongest bound still comes from $\hat{c}_{MATCH}$, even in the cases in which the blossom separation is only partial. As before, the gap differences decrease and the gaps improve as density increases; however, some of this may be a result of the heuristic's performance in addition to the bounds.

In terms of scaling, we see notable increases in running times compared to the sparser instances for all but the static bound. The clique bound requires approximately five times the computational effort for $n = 800$ when $\lambda = 80$, compared to $\lambda = 20$. The increases in running time are more severe for both c_{DEG} and $\hat{c}_{MATCH}$, with the latter's running time increasing by nearly a factor of 33 for the same instance classes.

		c_{STAT}		c_{CLQ}		c_{DEG}		c_{MATCH}	
$n = s_{\max}$	λ	Gap	Time	Gap	Time	Gap	Time	Gap	Time
800	5	1.59%	0.00	0.96%	1.29	3.65%	0.53	0.00%	27.22
	10	1.42%	0.00	1.05%	1.48	2.17%	0.99	0.03%	34.75
	20	1.01%	0.00	0.85%	2.25	1.05%	13.02	0.03%	80.01
1,600	5	1.53%	0.00	0.96%	1.74	3.47%	1.95	0.01%	100.98
	10	1.27%	0.01	0.96%	2.08	1.93%	3.67	0.03%	109.86
	20	0.99%	0.01	0.84%	3.16	1.10%	64.99	0.06%	282.53
2,400	5	1.56%	0.01	1.00%	2.08	3.52%	4.45	0.01%	212.63
	10	1.33%	0.01	0.97%	2.72	2.11%	6.37	0.04%	222.43
	20	1.05%	0.01	0.89%	4.79	1.20%	146.43	0.12%	637.81

Table 4.2: Comparison of $c_{STAT}, c_{CLQ}, c_{DEG}$, and c_{MATCH} on sparse instances.

		c_{STAT}		c_{CLQ}		c_{DEG}		$\hat{c}_{MATCH}$	
$n = s_{\max}$	λ	Gap	Time	Gap	Time	Gap	Time	Gap	Time
800	40	2.33%	0.01	2.23%	3.92	2.18%	126.64	1.61%	390.98
	80	1.95%	0.02	1.90%	12.44	1.74%	756.00	1.43%	2,593.07
900	40	2.52%	0.01	2.44%	3.94	2.36%	155.92	1.84%	464.87
	80	1.96%	0.02	1.91%	11.06	1.84%	1,083.42	1.49%	3,330.88
1,000	40	2.38%	0.01	2.29%	4.31	2.26%	223.44	1.71%	602.49
	80	1.93%	0.02	1.89%	11.13	1.75%	1,505.21	1.47%	3,496.53

Table 4.3: Comparison of $c_{STAT}, c_{CLQ}, c_{DEG}$, and $\hat{c}_{MATCH}$ on dense instances.

4.7.5 Application-Based Instances

Next, we evaluate our methods' performance on seven instances drawn from a real-world cloud system, Microsoft Azure, as described in Section 4.7.2. For each of these instances, we solve (4.8) with a 600-second time limit. We also test one additional instance constructed by combining all jobs from the original seven; for this instance, we solve the IP with a time limit of 7,200 seconds. In addition to the IP solves, we compute c_{STAT} and c_{CLQ}.

For each instance, Table 4.4 displays the number of jobs, plus average job length and standard deviation as a percentage of the horizon. The table also includes the best upper and lower bounds from the (4.8), c_{STAT}, c_{CLQ}, and corresponding running times. We highlight upper and lower bounds when they are provably optimal.

			IP (4.8)			c_{STAT}		c_{CLQ}	
n	Avg. Length	+/-	UB	LB	Time	LB	Time	LB	Time
299	0.32%	1.16%	**0.541**	**0.541**	0.08	0.538	0.00	**0.541**	0.93
140	3.99%	13.87%	**3.105**	**3.105**	0.15	3.085	0.00	3.090	0.91
567	0.18%	0.87%	**0.525**	**0.525**	0.65	**0.525**	0.00	**0.525**	1.54
484	13.69%	6.89%	**33.343**	33.241	600	33.338	0.00	**33.343**	11.60
798	0.62%	3.85%	**2.747**	**2.747**	3.75	2.721	0.00	2.728	1.62
1,872	0.03%	0.33%	**0.384**	**0.384**	0.09	0.382	0.00	**0.384**	3.96
3,774	0.19%	0.94%	3.764	3.762	600	3.749	0.00	3.753	8.55
7,934	0.91%	3.79%	38.435	36.495	7,200	36.512	0.01	36.525	846.00

Table 4.4: Evaluation of IP (4.8), c_{STAT}, and c_{CLQ} using real data.

Overall, these instances appear easier than the synthetic instances. For five of the original instances, we could solve (4.8) within the 600-second time limit. Furthermore, even for the instances in which the IP could not be solved within the time limit, both c_{STAT} and c_{CLQ} provide lower bounds with an absolute gap of 0.03 or smaller. For these instances, c_{CLQ} does not appear to improve much on c_{STAT}, even when the instance is sparse. Notably, for the 484-job instance, both c_{STAT} and c_{CLQ} improve on the best bound the solver obtains within the time limit, and c_{CLQ} provides a tight lower bound for this instance. Interestingly, the 484-job instance also has the longest jobs as a percentage of the horizon.

For the large, aggregate instance, (4.8) has a larger gap, approximately 5%. Both c_{STAT} and c_{CLQ} improve on the best bound found by the solver, but only marginally. This instance requires a significant increase in computing time; the LP relaxation of (4.8) takes over an hour to solve, and c_{CLQ} also takes significantly longer.

4.7.6 Large-Scale Instances

In our final set of experiments, we test the scalability of our methods using very large synthetic instances. To accommodate these larger instances, we use a different computational setup. We now use a Linux machine with a 64-core AMD Epyc CPU and 1 TB RAM, and implement our methods using C++. We still use Gurobi 9.5.1, but run LP solves using the barrier method with crossover disabled. We solve matching problems with the LEMON graph library [125]. We compute upper bounds using the previously described methods; see Section 4.7.1.

We compare the performance of c_{STAT}, a clique bound, and c_{DEG}. To reduce computing times, we do not optimize breakpoints to compute c_{CLQ}; instead, we heuristically choose breakpoints and only solve one matching per maximal clique. Specifically, for each $C_i \in \mathbb{C}$ we set $\hat{\gamma}_i = e_{C_i}$. This choice of breakpoints corresponds to taking the earliest feasible breakpoint between consecutive cliques.

We use instances with $n = 100,000$ and $\lambda \in \{5, 10, 20, 40, 80\}$, generating five random realizations for each choice of λ. For upper bounds, we use the constructive and local search heuristics with a 3,600-second time limit. For each choice of λ we compute average running times for c_{STAT}, $c_{CLQ}(\hat{\gamma})$, and c_{DEG}, and the geometric means of the corresponding gaps relative to the upper bound; we summarize results in Table 4.5.

The results show that $c_{CLQ}(\hat{\gamma})$ can improve on c_{STAT}'s gap by as much as 0.5% in absolute terms; as before, this occurs for the sparsest instances with $\lambda = 5$. As density increases, this difference decreases to 0.03% on average for $\lambda = 80$. Similar to previous experiments, c_{DEG} does noticeably worse than the static and clique bounds for $\lambda = 5, 10, 20$, and then

$n = s_{\max}$	λ	c_{STAT} Gap	Time	$c_{CLQ}(\hat{\gamma})$ Gap	Time	c_{DEG} Gap	Time
100,000	5	1.99%	0.11	1.54%	0.13	3.91%	17.46
	10	2.50%	0.11	2.26%	0.13	3.22%	36.52
	20	3.33%	0.11	3.21%	0.13	3.47%	116.53
	40	4.22%	0.11	4.16%	0.13	4.15%	991.85
	80	4.62%	0.11	4.59%	0.13	4.51%	11,199.77

Table 4.5: Comparison of c_{STAT}, $c_{CLQ}(\hat{\gamma})$, and c_{DEG} on very large instances.

improves on them for the denser instances. In terms of scaling, using the alternate computational setup we are able to compute c_{STAT} and $c_{CLQ}(\hat{\gamma})$ in less than a second on average. For c_{DEG}, we observe generally poor scaling with respect to density. For the sparsest instances, the LP solves in approximately 17 seconds on average, but this increases to over three hours for the densest instances. Lastly, contrary to the previous experiments, the gaps are larger for higher densities. We suspect that this is a consequence of the significant difficulty of finding high-quality feasible solutions for higher-density instances at this scale.

4.8 Conclusion

In this work, we studied temporal bin packing with half-capacity jobs. Using the equivalence between minimizing time-average bins and maximizing time-average savings, we provided a novel IP formulation based on matchings. Additionally, we studied various lower bounds for the problem. We demonstrated that the easily computed static bound can have an arbitrarily large additive gap. With this motivation, we studied clique instances, and derived a new lower bound approach that improves the static bound. We then compared these bounds, along with various linear programming bounds obtained from our new formulation and a set partition formulation. We derived a hierarchy of these bounds, specifically demonstrating how many of the bounds can be obtained as relaxations of our new formulation. Finally, we conducted a computational study using a variety of synthetic and application-based instances. We compared our novel formulation against a more standard

assignment IP, and demonstrated its improved performance. Additionally, we extended our theoretical comparison of bounds with an empirical study, showing that for small- to medium-sized instances, the LP relaxation of our new formulation supplemented with blossom inequalities provides a near-optimal lower bound. For larger instances, the clique bound scales well while improving on the static bound, particularly for sparse instances; for dense instances, new formulation's LP relaxation is stronger than both the static and clique bounds.

While we have shown the strength the new formulation and its linear relaxation, particularly when including blossom inequalities, we still find instances in which we cannot prove optimality within a reasonable time. One future avenue of research is to conduct a further polyhedral study of this formulation with the goal of determining valid inequalities that can help close this gap. Based on preliminary empirical observations, even relatively simple instances can have a complex facial structure, with many facets beyond blossom inequalities.

An additional area of future work is to determine how these results relate to temporal bin packing variants with more general job sizes; however, even the uniform case is quite challenging. Many of our results are based on matchings, and have hyper-graph matching analogues when job demands are uniformly $1/k$ of capacity for some integer k. But even for $k = 3$, these ideas become much less practical if directly extended, and may require much additional effort.

CHAPTER 5
CONCLUSION

In this thesis we explored three problems in packing and scheduling. We now summarize the major contributions of each chapter. In Chapter 2, we introduced a dynamic version of the node packing problem. We formulated the problem as an MDP and demonstrated that even in the restricted case of star graphs that the problem was NP-Hard. Motivated by the weakness of an edge-based relaxation of the problem's achievable probabilities polytope, we conducted a polyhedral study. From this study we dervied an explicit convex hull description when the underlying graph is a clique with uniform edge probabilities. For paths and cycles, we derived an implcit description via a cutting plane algorithm based on a compact dynamic programming formulation. Lastly, we conducted a computational study demonstrating that our inequalities are able to greatly improve the quality of the bound obtained by our relaxation.

We studied interval scheduling with economies of scale in Chapter 3. Specifically, we focused on special cases using the max-weight function and non-negative, non-decreasing concave functions of total schedule weight. Our solution method was based on a set cover formulation that was solved using a branch-and-price algorithm. To solve the linear relaxation, we derived a polynomial time and a pseduo-polynomial pricing algorithm for the max-weight and general functions of total schedule weight, respectively. As a secondary result, for the special case in which the conflict graph is a path, we provided an integral linear programming formulation. We concluded with a computational study demonstrating the effectiveness of our method. We showed the ability to provably optimize instances with hundreds of jobs and, for instances in which we could not prove optimality, provide solutions with provably small gaps.

Finally, in Chapter 4, we studied temporal bin packing with half-capacity jobs. For two

special cases, we studied the worst case performance of a static lower bound. Motivated by these examples, we introduced a new lower bound and a novel IP formulation based on matchings. We proved theoretical guarantees on how the static bound, the matching-based bound, and some linear programming bounds compare. We then extended this comparison empirically using both synthetic instances and instances from a cloud computing application. We also demonstrated that our new IP formulation outperforms a standard formulation on a similarly diverse set of instances.

5.1 Future Work

In Chapters 2,3, and 4 we included individual conclusions with statements of future work. We briefly summarize these potential research directions below.

Dynamic Node Packing

1. For general graph structures, the largest reduction in gap came from probabilistic clique inequalities. These inequalities assume uniform probabilities; for the non-uniform case we suspect that probabilistic clique inequalities will be similarly useful. We provided a convex hull description for a clique with three nodes and non-uniform probabilities, but empirical investigation suggests that even for a graph with four nodes the number of facets is very large.

2. There are additional graph structures that provide important inequalities for the non-dynamic case, e.g., claws. Studying these structures may lead to useful inequalities in the dynamic case, both for uniform and non-uniform probabilities.

Interval Scheduling with Economies of Scale

1. An immediate next step is to determine for what other cost functions our approach can be used. The primary determinant of whether or not this approach is feasible is the complexity of the underlying pricing problem. We are interested in extending the approach to a more general class of submodular functions, to which both of our

tested functions belong. In this case, the pricing problem is a supermodular maximization with interval packing constraints; to our knowledge this problem has not been previously studied.

2. In our preliminary investigation we determined that adding cutting planes to our algorithm did not improve performance. In practice the root relaxation was already tight or nearly tight for our test cases. Nevertheless, it is easy to construct simple instances in which the root relaxation is not tight. For these instances, the addition of cutting planes may be useful.

Temporal Bin Packing with Half-Capacity Jobs

1. The performance of the matching-based IP can be further improved by adding valid inequalities. We observe improvements from adding blossom inequalities, but additional inequalities could help solve more challenging instances. An initial examination of an instance with only two maximal cliques suggests the facial structure is complex.

2. An important next step is to determine how the results for temporal bin packing with half-capacity jobs can be extended to more general uniform job sizes. It is not obvious if the matching IP has a direct extension, even for one-third-capacity jobs. The clique bound almost extends directly, but the sub-problems now become a special type of hyper-edge matching, a problem known to be NP-Hard in general.

Appendices

APPENDIX A
REMAINING PROOFS

A.1 Proof of Proposition 4.3.1

In the three-period problem there are six possible start/end period configurations: three single-period jobs, two two-period jobs, and one three-period job. We use n_{t_1,t_2} to denote the number of jobs that start at the beginning of period t_1 and end at the end of period t_2, for $t_1, t_2 \in \{1, 2, 3\}$ with $t_1 \leq t_2$. Before arguing the main result, we require the following lemma.

Lemma A.1.1. *In a three-period problem, any two jobs i, j spanning two or three periods that share the same start and end periods ($s_i = s_j, e_i = e_j$) can be paired and removed from the instance without loss of optimality.*

Proof. Assume otherwise that we have two jobs i, j with $s_i = s_j = 0$ and $e_i = e_j$ that cannot be placed on the same bin in an optimal solution. First, consider the case in which jobs i, j have $e_i = e_j = 1$, i.e., they span all three periods. Let the two bins these jobs are placed in be B and D, respectively. Let c_B and c_D be the costs of the two bins; note $c_B = c_D = w_1 + w_2 + w_3$. Consider the alternate solution in which $B' = \{i, j\}$ and $D' = \{k | \forall k \in (B \cup D)/\{i, j\}\}$. The cost satisfies $c_{B'} + c_{D'} \leq 2(w_1 + w_2 + w_3) = c_B + c_D$ and B' and D' are feasible as B, D are feasible.

Now, consider the case in which $s_i = s_j = 0$ and $i, j \notin J(3)$; this case covers both two-period cases via symmetry. As before, assume that jobs i and j are put in bins B and D. If $n_{3,3} = 2$ we construct B' to contain i, j and the two single-period jobs in period 3 and place the remainder in D'. The bin costs satisfy $c_{B'} + c_{D'} \leq c_B + c_D$ by the same argument as with the three-period jobs by pairing jobs i, j each with one of the single-period jobs. Note bin B' could equivalently be split into two bins without changing the total cost, one

containing i, j and the other containing the single-period jobs. Now assume that $n_{3,3} < 2$; in this case set $B' = \{i, j\}$ and D' to be the remainder. If $J(3) = \emptyset$, B', D' are optimal again by the same argument as with the three-period jobs. Assume that $J(3) \neq \emptyset$; then,

$$c_{B'} + c_{D'} \leq 2(w_1 + w_2) + w_3 \leq c_B + c_D. \qquad \square$$

Lemma A.1.1 implies we can reduce a three-period instance by pre-processing pairs of two- and three-period jobs with matching start and end times. After applying this reduction, the resulting instance has $n_{1,2}, n_{2,3}, n_{1,3} \in \{0, 1\}$.

Lemma A.1.2. *In a three-period instance with* $n_{1,2}, n_{2,3}, n_{1,3} \leq 1$, $c^* - c_{STAT} \leq 1/2$.

Proof. Consider the cases with one or more of $n_{1,2}, n_{2,3}, n_{1,3} = 0$. Pack all non-single-period jobs on a single bin, and then pack all single-period jobs greedily, filling in open bins first before opening new bins. This placement results in a solution with cost equal to the static bound. Now assume there is one of each of the non-single-period jobs; these jobs require at least two bins. Assume that $w_1 \geq w_3$. If $n_{1,1} > 0$ or $n_{3,3} > 0$, a single-period job can be paired with the complementary two-period job and packed with the three-period job optimally by the same argument as in Lemma A.1.1. After removing these jobs, the instance returns to the first case and has no gap. Now assume that $n_{1,1}, n_{3,3} = 0$; the optimal packing must then incur an absolute gap of at least w_3. The single-period jobs in period 2 can be placed greedily, and the optimal solution matches the static bound in this period.

Finally, we conclude that the worst-case gap occurs when $w_1 = w_3 = (1-\epsilon)/2$, $w_2 = \epsilon$, $n_{1,1}, n_{3,3} = 0$, and $n_{1,2}, n_{2,3}, n_{1,3} = 1$. The result follows by taking the limit $\epsilon \to 0$. $\square$

The proposition follows by combining Lemmas A.1.2 and A.1.1.

A.2 Proof of Proposition 4.3.2

We prove this result by by showing that a sequence of the instances described in Section 4.3.3 has an increasing additive gap $c^* - c_{STAT}$. Consider instances with $t_{\max} = 4k - 1$ for

some positive integer k, $\hat{t} = (t_{\max} + 1)/2$, and uniform period weights. These parameters imply $n = (t_{\max}+1)^2/4$, which is even given our choice of $t_{\max}$. Furthermore, $|J(t)|$ is even for each $t \in T$, and for each $t \in \{1, 2, ..., \hat{t}\}$ the number of jobs starting in period t is $(t_{\max}+1)/2$. As we have an even number of jobs in each period, $c_{STAT} = \sum_{t \in T} |J(t)|/2t_{\max}$.

As this instance has a single maximal clique, we can use a matching formulation [114]. Consider the following formulation,

$$\min_{x} \sum_{i,j \in J} c_{i,j} x_{i,j} \tag{A.1a}$$

$$\text{s.t.} \sum_{j \in J | j \neq i} x_{i,j} \geq 1 \qquad \forall i \in J \tag{A.1b}$$

$$x_{i,j} \in \{0, 1\}, \qquad i, j \in J, \tag{A.1c}$$

where $c_{i,j} = \max(e_i, e_j) - \min(s_i, s_j) = \sum_{t \in T} \mathbb{1}_{\{\{i,j\} \cap J(t) \neq \emptyset\}}/t_{\max}$.

Let $\hat{c}_{i,j} = \sum_{i,j \in J} \mathbb{1}_{\{|\{i,j\} \cap J(t)|=1\}}/2t_{\max}$. Intuitively, if we pair jobs i and j, this coefficient measures the number of periods in which exactly one of the jobs is active but not the other. Since the relaxed solution of the c_{STAT} bound does not include any machines with only one active job at any time, we can rewrite the objective of (A.1) as $\sum_{i,j} c_{i,j} x_{i,j} = c_{STAT} + \sum_{i,j} \hat{c}_{i,j} x_{i,j}$. The quantity $c_{STAT} = \sum_{t \in T} |J(t)|/2t_{\max}$ does not depend on matching decisions, so we can equivalently optimize (A.1) with this new objective.

Next, we construct a solution for (A.1). Order the jobs by increasing start periods and decreasing end periods, and pair them in that order, setting the corresponding edge variables to one. Because we have an even number of jobs starting in each of the periods $1, 2, .., \hat{t}$, each of these jobs is paired with another job starting in the same period and ending one period apart; therefore, if jobs i, j are paired, then $2t_{\max}\hat{c}_{i,j} = 1$. As n is even, the cost of this solution under the objective $\hat{c}$ is $(n/2) \times (1/2t_{\max}) = n/4t_{\max}$.

We show this solution is optimal by producing a corresponding dual bound. Consider

the dual of the linear relaxation of (A.1), with edge costs $2t_{\max}\hat{c}_{i,j}$,

$$\max_{y \geq 0} \sum_{i \in J} y_i \tag{A.2a}$$

$$\text{s.t. } y_i + y_j \leq \sum_{t \in T} \mathbb{1}_{\{|\{i,j\} \cap J(t)|=1\}} \qquad \forall i, j \in J. \tag{A.2b}$$

Construct a solution in which $y_{(t_1,t_2)} = 1$ if $t_1 + t_2$ is even, and $y_{(t_1,t_2)} = 0$ otherwise. The value of this solution is $n/2$, as half of the y variables are set to one.

We now argue the solution's dual feasibility. Given two jobs $(t_1, t_2), (t_1', t_2')$, the left-hand side of constraint (A.2b) has $0 \leq y_{(t_1,t_2)} + y_{(t_1',t_2')} \leq 2$, as the y variables are binary. Furthermore, we have $\mathbb{1}_{\{|\{i,j\} \cap J(t)|=1\}} = |t_1 - t_1'| + |t_2 - t_2'| \geq 1$. Therefore, the constraints hold when $y_{(t_1,t_2)} + y_{(t_1',t_2')} \leq 1$, occurring when at most one of $t_1 + t_2$, $t_1' + t_2'$ is even. It remains to consider the case in which both are even; assume for contradiction that $|t_1 - t_1'| + |t_2 - t_2'| = 1$; this means that either $t_1 = t_1'$ or $t_2 = t_2'$. Without loss of generality, assume that $t_1 = t_1'$ and $t_2 < t_2'$. Then $t_2 + 1 = t_2'$, but this implies that exactly one of $t_1 + t_2$, $t_1' + t_2'$ is even. We thus conclude that $|t_1 - t_1'| + |t_2 - t_2'| \geq 2$, and the constraints are satisfied.

By scaling this dual solution down by a factor of $2t_{\max}$, we obtain a feasible dual solution for the LP relaxation of (A.1) with objective value $n/4t_{\max}$, implying our primal solution is optimal. Therefore,

$$c^* - c_{STAT} = \frac{n}{4t_{\max}} = \frac{(t_{\max}+1)^2}{16t_{\max}} \geq \frac{t_{\max}}{16},$$

and the result follows as $t_{\max}$ can be made arbitrarily large.

A.3 Proof of Proposition 4.6.6

We prove the equivalent statement $z_{PART} = z_{DEG}$, splitting the result into two parts.

Lemma A.3.1. $z_{PART} \leq z_{DEG}$.

Proof. We argue than any fractional solution to the maximization version of (4.10) implies a corresponding fractional solution to (4.8) with the same objective value. Given a fractional solution η to the partition formulation, we construct a solution ρ by setting $\rho_{i,j} = \sum_{S \in \mathbb{S}, S \ni i,j} \eta_S$, for job pairs $i, j \in O$.

To see that ρ is feasible, suppose it violates constraint (4.8b) for some clique C and job i; this implies $1 < \sum_{j \in C | j \neq i} \rho_{i,j} \leq \sum_{S \in \mathbb{S}(i)} \eta_S$, but this contradicts the feasibility of η for the LP relaxation of (4.10). Furthermore, the objective coefficient of a set S in maximization terms is $\omega_S = \sum_{i,j \in O} \mathbb{1}_{\{i,j \in S\}} o_{i,j}$; this implies

$$\sum_{i,j \in O} o_{i,j} \rho_{i,j} = \sum_{i,j \in O} o_{i,j} \left[\sum_{S \in \mathbb{S}} \mathbb{1}_{\{i,j \in S\}} \eta_S \right] = \sum_{S \in \mathbb{S}} \left[\sum_{i,j \in O} \mathbb{1}_{\{i,j \in S\}} o_{i,j} \right] \eta_S = \sum_{S \in \mathbb{S}} \omega_S \eta_S. \quad \square$$

Lemma A.3.2. $z_{DEG} \leq z_{PART}$.

Proof. Take a solution ρ to the linear relaxation of (4.8). We make use of the following LP to obtain a minimally infeasible projection of the solution ρ into the space of η variables:

$$\min_{\eta, \nu \geq 0} \sum_{i \in J} \nu_i \tag{A.3a}$$

$$\text{s.t.} \sum_{S \in \mathbb{S}(i)} \eta_S \leq 1 + \nu_i \qquad \forall i \in J \tag{A.3b}$$

$$\sum_{S \in \mathbb{S}, S \ni i,j} \eta_S = \rho_{i,j} \qquad \forall i, j \in J. \tag{A.3c}$$

This LP attempts to find a fractional solution η that corresponds to the solution ρ, while minimizing the violation of the partition constraints. We only need to consider over-coverage of a job, as under-coverage can be resolved by appropriately assigning weight to η variables corresponding to single jobs. If the optimal objective to LP (A.3) is zero, there is a feasible projection and the result follows; assume otherwise that $\sum_{i \in J} \nu_i > 0$, and let i be some job with $\nu_i > 0$.

Let $\mathbb{S}'(i)$ be the set of feasible sets containing i with $\eta_S > 0$. We assume the sub-graph

induced by each set is connected and that, for each $S \in \mathbb{S}'(i)$, every job in S overlaps job i. We may assume this without loss of generality, as we can group all jobs up to and including the first job overlapping i into a single job, and similarly for jobs after i. If the sets induce disconnected graphs, they can be instead split into connected components with different η variables.

Construct a graph as follows. For each $j \in S \setminus \{i\}$, $S \in \mathbb{S}'(i)$, create a node (j, S); let the set of these nodes be N. Two nodes $(j_1, S_1), (j_2, S_2) \in N$ are adjacent if $j_1 = j_2$ or jobs j_1, j_2 overlap. This graph is a conflict graph of the jobs that overlap i in the current solution, potentially copied if a job j is in multiple sets in $\mathbb{S}'(i)$. We associate each node $(j, S) \in N$ with a weight $\zeta_{(j,S)} = \eta_S$, representing the amount of coverage that S provides in the constraints of (A.3).

Consider two cases. First, assume $|\mathbb{S}'(i)|$ equals the coloring number of this graph. If this is the case, since the graph is the conflict graph of interval jobs, there must be some clique of size $|\mathbb{S}'(i)|$. This means there is some set of nodes $N' \subseteq N$ with $|N'| = |\mathbb{S}'(i)|$, with all nodes adjacent; let $N'(j)$ be the set of these nodes containing j. As the nodes N' are all adjacent, either the corresponding jobs overlap or are duplicates of the same job. This means that there is some maximal clique $C \in \mathbb{C}$ in the original instance containing all of the jobs covered by nodes in N'; however, $\rho_{i,j} \geq \sum_{(j,S) \in N'(j)} \zeta_{(j,S)}$ implies

$$\sum_{j \in C/\{i\}} \rho_{i,j} \geq \sum_{j \in C/\{i\}} \sum_{(j,S) \in N'(j)} \zeta_{(j,S)} = \sum_{S \in \mathbb{S}'(i)} \eta_S > 1,$$

contradicting the assumption that η is feasible. The equality follows as $|N'| = |\mathbb{S}'(i)|$ and no two nodes $(j_1, S_1), (j_2, S_2) \in N'$ have $S_1 = S_2$. The last inequality follows from the assumption that i is over-covered by η, i.e. that $\nu_i > 0$.

Now assume instead that $|\mathbb{S}'(i)|$ is greater than the graph's coloring number; we do not need to consider the case in which it is less, as the packings $S \in \mathbb{S}'(i)$ imply a coloring on this graph. Because the constraint matrix of (4.8) is rational, a solution to its LP relaxation

is rational; this implies that a solution to (A.3) is also rational. For each $S \in \mathbb{S}'(i)$ we can write $\eta_S = p_S/q_S$ for some $p_S, q_S \in \mathbb{Z}_{\geq 1}$. Let L be the least common multiple of q_S for $S \in \mathbb{S}'(i)$. Modify the previous graph by creating $p_S L/q_S$ copies of each node $(j, S) \in N$, yielding nodes $(j, S)_\iota$ for $\iota \in \{1, 2, ..., \frac{p_S L}{q_S}\}$; the modified graph has the same edges as before. Each node $(j, S)_\iota$ gets weight $\zeta_{(j,S)_\iota} = 1/L$.

In this new graph, compute a minimum coloring. Let the set of color classes be $\mathbb{K}$. We use $\mathbb{K}(j)$ to denote the subset of those colors that contain nodes covering job j. We now modify the η variables. First, set $\eta_S = 0$ for $S \in \mathbb{S}'(i)$ and leave the remainder unchanged. For each color class $K \in \mathbb{K}$, create a packing $S_K = \{j : \forall (j, S)_\iota \in K\} \cup \{i\}$ and set $\eta_{S_K} = 1/L$. If there are $K_1, K_2 \in K$ with $S_{K_1} = S_{K_2}$, the weights can instead be aggregated, but for simplicity we assume that duplicate packings are allowed as they can be added to (4.10) without altering the objective. Let $\mathbb{S}''(i) = \{S_K : K \in \mathbb{K}\}$. This new solution does not change either the coverage of jobs $j \in J/\{i\}$ or $\rho_{i,j}$, as

$$\sum_{K \in \mathbb{K}(j)} \eta_{S_K} = \sum_{S \in \mathbb{S}'(i), S \ni j} \sum_{\iota=1}^{p_S L/q_S} \zeta_{(j,S)_\iota} = \sum_{S \in \mathbb{S}'(i), S \ni j} \zeta_{(j,S)} = \rho_{i,j};$$

this follows as each node covering j must belong to a different class K and the η_S with $i \notin S$ are unchanged. Lastly, as we have computed a minimum coloring we know that the clique number equals $|K| = |\mathbb{S}''(i)|$, and this new solution returns to the first case.

Finally, we conclude that an optimal solution to LP (A.3) has $\sum_{i \in J} \nu_i = 0$, implying that we can always find a fractional solution to (4.10) matching the given fractional solution to (4.8). □

REFERENCES

[1] *Grand view research: Cloud computing market size report, 2022-2030.*

[2] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[3] C. Muir and A. Toriello, "Dynamic node packing," *Mathematical Programming*, Forthcoming.

[4] M. Y. Kovalyov, C. Ng, and T. E. Cheng, "Fixed interval scheduling: Models, applications, computational complexity and algorithms," *European journal of operational research*, vol. 178, no. 2, pp. 331–342, 2007.

[5] H. Waterer, E. L. Johnson, P. Nobili, and M. W. Savelsbergh, "The relation of time indexed formulations of single machine scheduling problems to the node packing problem," *Mathematical Programming*, vol. 93, no. 3, pp. 477–494, 2002.

[6] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco, "An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation," *Management science*, vol. 44, no. 5, pp. 714–729, 1998.

[7] S. Basagni, "Finding a maximal weighted independent set in wireless networks," *Telecommunication Systems*, vol. 18, no. 1-3, pp. 155–168, 2001.

[8] B.-H. Liu, N.-T. Nguyen, V.-T. Pham, and Y.-H. Yeh, "A maximum-weight-independent-set-based algorithm for reader-coverage collision avoidance arrangement in rfid networks," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1342–1350, 2015.

[9] W. Brendel, M. Amer, and S. Todorovic, "Multiobject tracking as maximum weight independent set," in *CVPR 2011*, IEEE, 2011, pp. 1273–1280.

[10] P. J. Zwaneveld, L. G. Kroon, and S. P. Van Hoesel, "Routing trains through a railway station based on a node packing model," *European Journal of Operational Research*, vol. 128, no. 1, pp. 14–33, 2001.

[11] N. J. Sloane, "On single-deletion-correcting codes," *Codes and designs*, vol. 10, pp. 273–291, 2000.

[12] O. Kwon, K. Lee, and S. Park, "Targeting and scheduling problem for field artillery," *Computers & industrial engineering*, vol. 33, no. 3-4, pp. 693–696, 1997.

[13] P. M. Pardalos and J. Xue, "The maximum clique problem," *Journal of Global Optimization*, vol. 4, no. 3, pp. 301–328, Apr. 1994.

[14] R. R. Vemuganti, "Applications of set covering, set packing and set partitioning models: A survey," in *Handbook of Combinatorial Optimization: Volume1–3*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 573–746.

[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer US, 1972, pp. 85–103.

[16] J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," *Acta Math.*, vol. 182, no. 1, pp. 105–142, 1999.

[17] E. Maslov, M. Batsyn, and P. M. Pardalos, "Speeding up branch and bound algorithms for solving the maximum clique problem," *Journal of Global Optimization*, vol. 59, no. 1, pp. 1–21, 2014.

[18] J. Walteros and A. Buchanan, "Why is maximum clique often easy in practice?" *Operations Research*, 2019, Forthcoming.

[19] G. L. Nemhauser and L. E. Trotter, "Vertex packings: Structural properties and algorithms," *Mathematical Programming*, vol. 8, no. 1, pp. 232–248, 1975.

[20] ——, "Properties of vertex packing and independence system polyhedra," *Mathematical Programming*, vol. 6, no. 1, pp. 48–61, 1974.

[21] M. W. Padberg, "On the facial structure of set packing polyhedra," *Mathematical programming*, vol. 5, no. 1, pp. 199–215, 1973.

[22] L. Cánovas, M. Landete, and A. Marín, "New facets for the set packing polytope," *Operations Research Letters*, vol. 27, no. 4, pp. 153–161, 2000.

[23] L. E. Trotter Jr, "A class of facet producing graphs for vertex packing polyhedra," *Discrete Mathematics*, vol. 12, no. 4, pp. 373–388, 1975.

[24] E. Cheng and W. H. Cunningham, "Wheel inequalities for stable set polytopes," *Mathematical programming*, vol. 77, no. 2, pp. 389–421, 1997.

[25] L. Lovász, "On the shannon capacity of a graph," *IEEE Transactions on Information theory*, vol. 25, no. 1, pp. 1–7, 1979.

[26] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization*. Springer Science & Business Media, 2012, vol. 2.

[27] G. J. Minty, "On maximal independent sets of vertices in claw-free graphs," *Journal of Combinatorial Theory, Series B*, vol. 28, no. 3, pp. 284–304, 1980.

[28] P. Prosser, "Exact algorithms for maximum clique: A computational study," *Algorithms*, vol. 5, Jul. 2012.

[29] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003.

[30] Q. Wu and J.-K. Hao, "A review on algorithms for maximum clique problems," *European Journal of Operational Research*, vol. 242, no. 3, pp. 693–709, 2015.

[31] M. Pinedo, "Stochastic scheduling with release dates and due dates," *Operations Research*, vol. 31, no. 3, pp. 559–572, 1983.

[32] C. Derman, G. J. Lieberman, and S. M. Ross, "A renewal decision problem," *Management Science*, vol. 24, no. 5, pp. 554–561, 1978.

[33] D. Blado, W. Hu, and A. Toriello, "Semi-infinite relaxations for the dynamic knapsack problem with stochastic item sizes," *SIAM Journal on Optimization*, vol. 26, no. 3, pp. 1625–1648, 2016.

[34] D. Blado and A. Toriello, "Relaxation analysis for the dynamic knapsack problem with stochastic item sizes," *SIAM Journal on Optimization*, vol. 29, no. 1, pp. 1–30, 2019.

[35] ——, "A column and constraint generation algorithm for the dynamic knapsack problem with stochastic item sizes," *Mathematical Programming Computation*, 2020, Forthcoming.

[36] B. C. Dean, M. X. Goemans, and J. Vondrák, "Approximating the stochastic knapsack problem: The benefit of adaptivity," *Mathematics of Operations Research*, vol. 33, no. 4, pp. 945–964, 2008.

[37] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[38] A. Toriello, W. B. Haskell, and M. Poremba, "A dynamic traveling salesman problem with stochastic arc costs," *Operations Research*, vol. 62, no. 5, pp. 1107–1125, 2014.

[39] B. C. Dean, M. X. Goemans, and J. Vondrák, "Adaptivity and approximation for stochastic packing problems," in *Proceedings of the sixteenth annual ACM-SIAM*

symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2005, pp. 395–404.

[40] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[41] D. Bertsimas and J. Niño-Mora, "Conservation laws, extended polymatroids and multiarmed bandit problems; a polyhedral approach to indexable systems," *Mathematics of Operations Research*, vol. 21, no. 2, pp. 257–306, 1996. eprint: https://doi.org/10.1287/moor.21.2.257.

[42] E. G. Coffman Jr and I. Mitrani, "A characterization of waiting time performance realizable by single-server queues," *Operations Research*, vol. 28, no. 3-part-ii, pp. 810–821, 1980.

[43] A. Torrico, S. Ahmed, and A. Toriello, "A polyhedral approach to online bipartite matching," *Mathematical Programming*, vol. 172, no. 1-2, pp. 443–465, 2018.

[44] D. Adelman, "Price-directed replenishment of subsets: Methodology and its application to inventory routing," *Manufacturing & Service Operations Management*, vol. 5, no. 4, pp. 348–371, 2003.

[45] D. P. De Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[46] M. M. Halldórsson, K. Iwama, S. Miyazaki, and S. Taketomi, "Online independent sets," *Theoretical Computer Science*, vol. 289, no. 2, pp. 953–962, 2002, Computing and Combinatorics.

[47] O. Göbel, M. Hoefer, T. Kesselheim, T. Schleiden, and B. Vöcking, "Online independent set beyond the worst-case: Secretaries, prophets, and periods," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2014, pp. 508–519.

[48] J. Boyar, L. M. Favrholdt, C. Kudahl, and J. W. Mikkelsen, "Advice complexity for a class of online problems," in *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[49] S. Dobrev, R. Královič, and R. Královič, "Advice complexity of maximum independent set in sparse and bipartite graphs," *Theory of Computing Systems*, vol. 56, no. 1, pp. 197–219, 2015.

[50] ——, "Independent set with advice: The impact of graph knowledge," in *International Workshop on Approximation and Online Algorithms*, Springer, 2012, pp. 2–15.

[51] N. Alon and J. H. Spencer, *The probabilistic method*. John Wiley & Sons, 2004.

[52] F. Juhász, "The asymptotic behaviour of Lovász' ϑ function for random graphs," *Combinatorica*, vol. 2, no. 2, pp. 153–155, Jun. 1982.

[53] U. Feige and R. Krauthgamer, "The probable value of the Lovász-Schrijver relaxations for maximum independent set," *SIAM Journal on Computing*, vol. 32, p. 2003, 2003.

[54] S. Ahmed and A. Atamtürk, "Maximizing a class of submodular utility functions," *Mathematical Programming*, vol. 128, no. 1, pp. 149–169, Jun. 2011.

[55] I. Kra and S. R. Simanca, "On circulant matrices," *Notices of the AMS*, vol. 59, no. 3, pp. 368–377, 2012.

[56] Gupta, Lee, and Leung, "An optimal solution for the channel-assignment problem," *IEEE Transactions on Computers*, vol. C-28, no. 11, pp. 807–810, 1979.

[57] S. Lee, J. Turner, M. S. Daskin, T. Homem-de-Mello, and K. Smilowitz, "Improving fleet utilization for carriers by interval scheduling," *European Journal of Operational Research*, vol. 218, no. 1, pp. 261–269, 2012.

[58] S. Martello and P. Toth, "A heuristic approach to the bus driver scheduling problem," *European Journal of Operational Research*, vol. 24, no. 1, pp. 106–117, 1986, OR and Microcomputers Miscellaneous OR Applications.

[59] V. Gabrel, "Scheduling jobs within time windows on identical parallel machines: New model and algorithms," *European Journal of Operational Research*, vol. 83, no. 2, pp. 320–329, 1995.

[60] S. Rojanasoonthon, J. F. Bard, and S. D. Reddy, "Algorithms for parallel machine scheduling: A case study of the tracking and data relay satellite system," *Journal of the Operational Research Society*, vol. 54, no. 8, pp. 806–821, 2003.

[61] F. C. Spieksma, "On the approximability of an interval scheduling problem," *Journal of Scheduling*, vol. 2, no. 5, pp. 215–227, 1999.

[62] M. Dell'Amico, F. Furini, and M. Iori, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, p. 104 825, 2020.

[63] F. Furini and X. Shen, "Matheuristics for the temporal bin packing problem," in *Recent Developments in Metaheuristics*, L. Amodeo, E.-G. Talbi, and F. Yalaoui, Eds., Cham: Springer International Publishing, 2018, pp. 333–345, ISBN: 978-3-319-58253-5.

[64] N. Aydın, İ. Muter, and Ş. İlker Birbil, "Multi-objective temporal bin packing problem: An application in cloud computing," *Computers & Operations Research*, vol. 121, p. 104 959, 2020.

[65] S. Olariu, "An optimal greedy heuristic to color interval graphs," *Information Processing Letters*, vol. 37, no. 1, pp. 21–25, 1991.

[66] M. C. Carlisle and E. L. Lloyd, "On the k-coloring of intervals," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 225–235, 1995.

[67] E. M. Arkin and E. B. Silverberg, "Scheduling jobs with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1–8, 1987.

[68] M. Y. Kovalyov, C. Ng, and T. E. Cheng, "Fixed interval scheduling: Models, applications, computational complexity and algorithms," *European Journal of Operational Research*, vol. 178, no. 2, pp. 331–342, 2007.

[69] P. Brucker and L. Nordmann, "The k-track assignment problem," *Computing*, vol. 52, no. 2, pp. 97–122, 1994.

[70] C. T. Ng, T. C. E. Cheng, A. M. Bandalouski, M. Y. Kovalyov, and S. S. Lam, "A graph-theoretic approach to interval scheduling on dedicated unrelated parallel machines," *Journal of the Operational Research Society*, vol. 65, no. 10, pp. 1571–1579, 2014.

[71] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira, "Scheduling split intervals," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 1–15, 2006.

[72] J. R. Correa and N. Megow, "Clique partitioning with value-monotone submodular cost," *Discrete Optimization*, vol. 15, pp. 26–36, 2015.

[73] D. Gijswijt, V. Jost, and M. Queyranne, "Clique partitioning of interval graphs with submodular costs on the cliques," *RAIRO-Operations Research*, vol. 41, no. 3, pp. 275–287, 2007.

[74] T. Fukunaga, M. M. Halldórsson, and H. Nagamochi, "Robust cost colorings.," in *SODA*, vol. 8, 2008, pp. 1204–1212.

[75] J. Cardinal, S. Fiorini, and G. Joret, "Minimum entropy coloring," in *International Symposium on Algorithms and Computation*, Springer, 2005, pp. 819–828.

[76] B. Escoffier, J. Monnot, and V. T. Paschos, "Weighted coloring: Further complexity and approximability results," *Information Processing Letters*, vol. 97, no. 3, pp. 98–103, 2006.

[77] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006, vol. 5.

[78] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.

[79] A. Mehrotra and M. A. Trick, "A column generation approach for graph coloring," *informs Journal on Computing*, vol. 8, no. 4, pp. 344–354, 1996.

[80] F. Furini and E. Malaguti, "Exact weighted vertex coloring via branch-and-price," *Discrete Optimization*, vol. 9, no. 2, pp. 130–136, 2012.

[81] D. Delle Donne, F. Furini, E. Malaguti, and R. W. Calvo, "A branch-and-price algorithm for the minimum sum coloring problem," *Discrete Applied Mathematics*, 2020.

[82] S. Gualandi and F. Malucelli, "Exact solution of graph coloring problems via constraint programming and column generation," *INFORMS Journal on Computing*, vol. 24, no. 1, pp. 81–100, 2012.

[83] F. Furini, E. Malaguti, and A. Santini, "An exact algorithm for the partition coloring problem," *Computers & Operations Research*, vol. 92, pp. 170–181, 2018.

[84] X. Ji and J. E. Mitchell, "Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement," *Discrete Optimization*, vol. 4, no. 1, pp. 87–102, 2007.

[85] E. Angelelli, N. Bianchessi, and C. Filippi, "Optimal interval scheduling with a resource constraint," *Computers & operations research*, vol. 51, pp. 268–281, 2014.

[86] W. Ben-Ameur and J. Neto, "Acceleration of cutting-plane and column generation algorithms: Applications to network design," *Networks: An International Journal*, vol. 49, no. 1, pp. 3–17, 2007.

[87] O. Hadary *et al.*, "Protean:{vm} allocation service at scale," in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, pp. 845–861.

[88] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[89] D. S. Johnson, "Near-optimal bin packing algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1973.

[90] D. Simchi-Levi, "New worst-case results for the bin-packing problem," *Naval Research Logistics (NRL)*, vol. 41, no. 4, pp. 579–585, 1994.

[91] G. Dósa and J. Sgall, "First fit bin packing: A tight analysis," in *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[92] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, IEEE, 1982, pp. 312–320.

[93] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting-stock problem," *Operations research*, vol. 9, no. 6, pp. 849–859, 1961.

[94] R. Hoberg and T. Rothvoss, "A logarithmic additive integrality gap for bin packing," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2017, pp. 2616–2625.

[95] S. Martello and P. Toth, "Lower bounds and reduction procedures for the bin packing problem," *Discrete applied mathematics*, vol. 28, no. 1, pp. 59–70, 1990.

[96] S. P. Fekete and J. Schepers, "New classes of fast lower bounds for bin packing problems," *Mathematical programming*, vol. 91, no. 1, pp. 11–31, 2001.

[97] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Survey and classification," in *Handbook of combinatorial optimization*, 2013, pp. 455–531.

[98] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[99] M. Dell'Amico, F. Furini, and M. Iori, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, p. 104 825, 2020.

[100] F. Furini and X. Shen, "Matheuristics for the temporal bin packing problem," in *Recent Developments in Metaheuristics*, Springer, 2018, pp. 333–345.

[101] K. Heßler, T. Gschwind, and S. Irnich, "Stabilized branch-and-price algorithms for vector packing problems," *European Journal of Operational Research*, vol. 271, no. 2, pp. 401–419, 2018.

[102] F. Brandao and J. P. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56–67, 2016.

[103] L. T. Kou and G. Markowsky, "Multidimensional bin packing algorithms," *IBM Journal of Research and development*, vol. 21, no. 5, pp. 443–448, 1977.

[104] K. Maruyama, S. Chang, and D. Tang, "A general packing algorithm for multidimensional resource requirements," *International Journal of Computer & Information Sciences*, vol. 6, no. 2, pp. 131–149, 1977.

[105] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *research. microsoft. com*, 2011.

[106] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on services computing*, vol. 3, no. 4, pp. 266–278, 2010.

[107] N. Aydın, İ. Muter, and Ş. İ. Birbil, "Multi-objective temporal bin packing problem: An application in cloud computing," *Computers & Operations Research*, vol. 121, p. 104 959, 2020.

[108] J. Martinovic, N. Strasdat, and M. Selch, "Compact integer linear programming formulations for the temporal bin packing problem with fire-ups," *Computers & Operations Research*, vol. 132, p. 105 288, 2021.

[109] P. Winkler and L. Zhang, "Wavelength assignment and generalized interval graph coloring," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 2003, pp. 830–831.

[110] M. Flammini *et al.*, "Minimizing total busy time in parallel scheduling with application to optical networks," *Theoretical Computer Science*, vol. 411, no. 40-42, pp. 3553–3562, 2010.

[111] J. Chang, S. Khuller, and K. Mukherjee, "Lp rounding and combinatorial algorithms for minimizing active and busy time," *Journal of Scheduling*, vol. 20, no. 6, pp. 657–680, 2017.

[112] M. Alicherry and R. Bhatia, "Line system design and a generalized coloring problem," in *European Symposium on Algorithms*, Springer, 2003, pp. 19–30.

[113] V. Kumar and A. Rudra, "Approximation algorithms for wavelength assignment," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer, 2005, pp. 152–163.

[114] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. Wong, and S. Zaks, "Optimizing busy time parallel machines," *Theoretical Computer Science*, vol. 562, pp. 524–541, 2015.

[115] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir, "Minimizing busy time in multiple machine real-time scheduling," in *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[116] R. Ren and X. Tang, "Clairvoyant dynamic bin packing for job scheduling with minimum server usage time," in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016, pp. 227–237.

[117] N. Buchbinder, Y. Fairstein, K. Mellou, I. Menache, *et al.*, "Online virtual machine allocation with predictions," *arXiv preprint arXiv:2011.06250*, 2020.

[118] C. C. Lee and D.-T. Lee, "A simple on-line bin-packing algorithm," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 562–572, 1985.

[119] M. De Cauwer, D. Mehta, and B. O'Sullivan, "The temporal bin packing problem: An application to workload management in data centres," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2016, pp. 157–164.

[120] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[121] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.

[122] M. W. Padberg and M. R. Rao, "Odd minimum cut-sets and b-matchings," *Mathematics of Operations Research*, vol. 7, no. 1, pp. 67–80, 1982.

[123] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[124] F. You, github.com/FanYouCN/BlossomSeparationPadbergRao, 2018.

[125] B. Dezső, A. Jüttner, and P. Kovács, "Lemon–an open source c++ graph template library," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23–45, 2011.