Temporal Bin Packing with Half-Capacity Jobs

Christopher Muir[1] Luke Marshall[2] Alejandro Toriello[1]

[1] School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia

[2] Microsoft Research
Redmond, Washington

May 2022

Abstract

Motivated by applications in cloud computing, we study a temporal bin packing problem with jobs that occupy half of a bin's capacity. An instance is given by a set of jobs, each with a start and end time during which it must be processed, i.e., assigned to a bin. A bin can accommodate two jobs simultaneously, and the objective is an assignment that minimizes the time-averaged number of open or active bins over the horizon; 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, which motivates us to introduce a novel combinatorial lower bound and an integer programming (IP) formulation, both based on an interpretation of the model as a series of connected matching problems. We theoretically compare the static bound, the new matching-based bounds, and various linear programming bounds. We perform a computational study using both synthetic and application-based instances, and show that our bounds offer significant improvement over existing methods, particularly for sparse instances.

1 Introduction

Temporal bin packing (TBP) is a 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. Even small relative improvements in server utilization can lead to large absolute gains; for example, [18] 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 6.7, is summarized graphically in Figure 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.

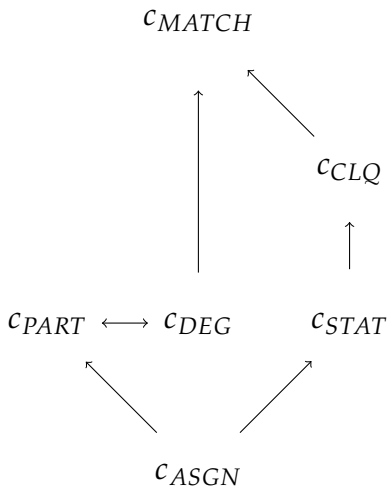

Figure 1: Graphical representation of Theorem 6.7. Arcs indicate that the bound at the tail is less than or equal to the bound at the head.

The outline of the paper is as follows. Section 2 summarizes relevant results from the literature. Next, in Section 3 we formulate TBP2 and present some preliminary results. In Section 4, we study clique instances and use them to derive a new lower bound. After that, Section 5 presents our IP formulation along with an additional formulation based on a set partitioning model. In Section 6, we provide our lower bound hierarchy, establishing the relative strengths of the discussed bounds. In Section 7, we present our computational study. We conclude in Section 8, while an appendix includes proofs omitted from the main body.

2 Literature Review

Static bin packing, or just bin packing, is among the most well-known NP-Complete problems [16]. 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 [22]. 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 [37]; conversely, if jobs aren't sorted the algorithms are tight 17/10-approximations [11]. Despite the negative approximability results in [37], [24] 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 [17]. More recently, [21] proposed an algorithm with a worst-case additive gap of $O(\log(OPT))$, using techniques from discrepancy theory and building on a previous result from [36].

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, [29] 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; [13] 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 [6, 9].

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 [8], the authors present a branch-and-price algorithm along with various methods for efficiently obtaining upper and lower bounds; [15] provide a matheuristic based on column generation methods. Other algorithms come from the vector bin packing literature and include branch-and-price [20], arc-flow approaches [3], and heuristics [26, 31, 34]. This TBP model has also been applied to problems in cloud scheduling. For instance, [38] use a variant to model a problem in virtual machine consolidation; [2] 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 [30] 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 [39], the authors show that TBP with uniform jobs is NP-Hard, even

when $g = 2$, which corresponds to our model TBP2. The authors of [14] prove that a First Fit algorithm is a 4-approximation; [5] demonstrate how prior results on fiber minimization and line system design in [1, 27] yield two 2-approximations. In [32], 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 [25] provide a 5-approximation and extend the result to a setting with flexible start times, while [35] describe a 4-approximation based on *dual coloring*. Most recently, [4] 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 [28]. In the context of cloud computing, [7] use an IP formulation to solve a variant in which all jobs have the same starting time.

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 [39]. In the remainder of the paper, 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., [5]. Consider the increasing sequence of distinct start and end times, i.e., let $\mathcal{T} = \{s_i : i \in J\} \cup \{e_i : i \in J\}$, and let $\mathcal{T}_{(t)}$ indicate the t-th value in ascending order for $t \in T = \{1, \ldots, |\mathcal{T}|\}$. Then, the interval $I_t := \left[\mathcal{T}_{(t-1)}, \mathcal{T}_{(t)}\right)$ is the t-th period, with weight $w_t = \mathcal{T}_{(t)} - \mathcal{T}_{(t-1)}$. 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 [23]. 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

$$\min \sum_{b \in \mathbb{B}} \sum_{t \in T} w_t y_{b,t} \tag{1a}$$

subject to

$$\sum_{b \in \mathbb{B}} x_{i,b} = 1 \qquad \forall i \in J \tag{1b}$$

$$\sum_{i \in J(t)} x_{i,b} \leq 2y_{b,t} \qquad \forall b \in \mathbb{B}, t \in T \tag{1c}$$

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

The x variables denote job-to-bin assignments, and the y variables track when a bin is active. IP (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 [32]. 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

$$c^* = \sum_{i \in J} (e_i - s_i) - z^*; \tag{2}$$

the cost and savings of any solution are related in the same way.

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 (1),

$$\min \sum_{b \in \mathbb{B}} \sum_{t \in T} w_t y_{b,t} \tag{3a}$$

subject to

$$\sum_{b \in \mathbb{B}} x_{i,b,t} = 1 \qquad \forall i \in J, t \in T \tag{3b}$$

$$\sum_{i \in J(t)} x_{i,b,t} \leq 2y_{b,t} \qquad \forall b \in \mathbb{B}, t \in T \tag{3c}$$

$$x_{i,b,t-1} = x_{i,b,t} \qquad \forall i \in J, b \in \mathbb{B}, t \in T \setminus \{1\} \tag{3d}$$

$$x_{i,b,t}, y_{b,t} \in \{0,1\}. \tag{3e}$$

IPs (1) and (3) are equivalent; the only difference is the expansion of the decision variables $x_{i,b}$ into $|T|$ copies and the addition of temporal linking constraints (3d). A natural relaxation of (3) is to remove constraints (3d). If these constraints are removed, the problem

decomposes into $|T|$ 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 (1). In our case, the sub-problems admit an analytic solution, and we obtain the bound

$$c_{STAT} = \sum_{t \in T} w_t \lceil |J(t)|/2 \rceil. \tag{4}$$

As (4) is obtained by a relaxation, $c_{STAT} \leq c^*$; we refer to c_{STAT} as the *static bound*. Using this bound, it follows that any solution has a cost of at most twice the optimum: the solution that assigns every job to a separate bin has a cost no larger than $2c_{STAT}$, and any other solution has lower cost.

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 [29]. It has also been previously called the *demand profile* bound in [5]. The static bound can be computed in $O(n \log(n))$ time.

3.2 Three-Period Instance

In this section we focus on the special case $|T| = 3$, the simplest case where $c_{STAT} < c^*$ is possible. Consider the following example, adapted from [27] and depicted in Figure 2: $J = \{1, 2, 3\}$, $s_1 = s_2 = 0, s_3 = (1-\epsilon)/2$, 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 $\epsilon \to 0$.

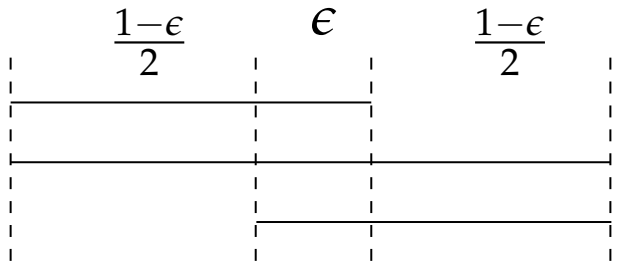

Figure 2: Three-period example.

We establish that this is the worst the static bound can do for any three-period instance with the following proposition, proved in Appendix A.1.

Proposition 3.1. *In a three-period instance,* $c^* - c_{STAT} \leq 1/2$.

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 5. Specifically, for a fixed $|T|$, 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 $(t_1, t_2) \in \{1, 2, .., \hat{t}\} \times \{\hat{t}, \hat{t}+1, ..., |T|\}$. An example with $|T| = 7, \hat{t} = 4$ is shown in Figure 3. Given $|T|$ and $\hat{t}$, the total number of jobs is $n = \hat{t}(1 + |T| - \hat{t})$. We use this family

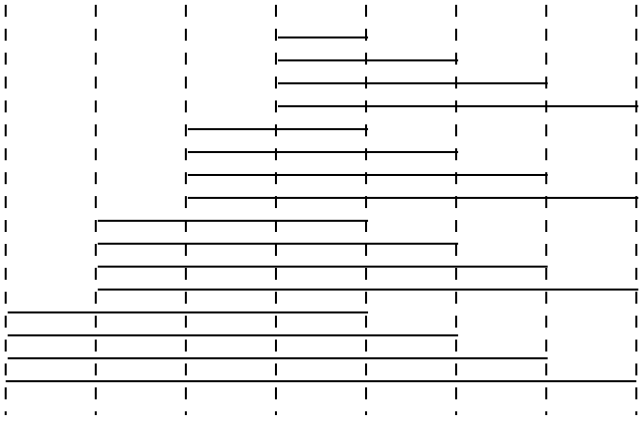

Figure 3: Example of instance from family with unbounded additive gap, where $|T| = 7, \hat{t} = 4$. This particular instance has worst case additive gap of $2/3$, with $w_{\hat{t}} = 0$ and $w_t = 1/6$ for $t \in T \setminus \{\hat{t}\}$.

of instances to establish the following proposition, proved in Appendix A.2.

Proposition 3.2. *There is no constant $\beta \geq 0$ such that $c^* - c_{STAT} \leq \beta$ for all instances.*

4 Clique Instances

The examples in Sections 3.2 and 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., [32].

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 [12].

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 the 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

$$c_{CLQ}(\gamma) = \sum_{C_i \in \mathbb{C}} c^*(C_i, \gamma_{i-1}, \gamma_i). \tag{5}$$

Proposition 4.1. *$c_{STAT} \leq c_{CLQ}(\gamma)$ for any feasible choice of γ.*

Proof. 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. 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 4.1 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 by the 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). Let c_μ be the weighted cost of the matching truncated to the interval $[s_{C_i}, e_{C_i}] \subseteq [\gamma_{i-1}, \gamma_i]$. 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

$$\max \sum_{i=1}^{m} \sigma_i \tag{6a}$$

subject to

$$\begin{aligned}
\sigma_1 &\leq c_\mu + (\gamma_1 - e_{C_1})\phi_\mu^+ && \forall \mu \in M_1 && (6b)\\
\sigma_m &\leq c_\mu + (s_{C_{|C|}} - \gamma_{m-1})\phi_\mu^- && \forall \mu \in M_m && (6c)\\
\sigma_i &\leq c_\mu + (\gamma_i - e_{C_i})\phi_\mu^+ + (s_{C_i} - \gamma_{i-1})\phi_\mu^- && \forall i \in \{2, ..., m-1\}, \mu \in M_i && (6d)\\
e_{C_i} &\leq \gamma_i \leq s_{C_{i+1}} && \forall i \in \{1, 2, ..., m-1\} && (6e)\\
\sigma, \gamma &\geq 0. && && (6f)
\end{aligned}$$

For each clique C_i, M_i may contain exponentially many perfect matchings; consequently, (6) may contain exponentially many constraints. Nevertheless, we can optimize (6) efficiently using constraint generation, as described in the following proposition.

Proposition 4.2. *The separation problem for* (6) *can be solved by computing* $m = O(n)$ *perfect matchings, and thus* (6) *can be solved in polynomial time.*

Proof. Assume we are given a candidate solution (σ^*, γ^*) to (6); 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 (6), 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 (6) 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.

In addition to providing a bound, partitioning the instance into cliques in this fashion also leads to a simple approximation algorithm, which we describe in the following proposition.

Proposition 4.3. *Let* m *be the number of maximal cliques. Then*

$$c^* \leq (2 - 1/m)c_{CLQ}(\gamma)$$

for any feasible choice of γ*.*

Proof. Take the clique $C_i \in \mathbb{C}$ with the highest sub-problem cost $c^*(C_i, \gamma_{i-1}, \gamma_i)$. Next, fix the packing decisions in the optimal solution to C_i's sub-problem; pack any remain-

ing jobs arbitrarily. Letting $c_{ALG}(\gamma_{j-1}, \gamma_j)$ be the cost of this solution over the interval $[\gamma_{j-1}, \gamma_j]$, the algorithm obtains a solution with cost

$$\begin{aligned}\sum_{j=1}^{m} c_{ALG}(\gamma_{j-1}, \gamma_j) &\leq 2\sum_{j\neq i} c^*(C_j, \gamma_{j-1}, \gamma_j) + c^*(C_i, \gamma_{i-1}, \gamma_i) \\ &= 2c_{CLQ}(\gamma) - c^*(C_i, \gamma_{i-1}, \gamma_i) \leq (2 - 1/m)c_{CLQ}(\gamma).\end{aligned}$$

For the first inequality, note that $2c_{STAT}$ is an upper bound on the cost of any feasible solution and $c_{STAT} \leq c_{CLQ}(\gamma)$ by Proposition 4.1. By construction, $c_{ALG}(\gamma_{i-1}, \gamma_i) = c^*(C_i, \gamma_{i-1}, \gamma_i)$. For the last inequality, C_i is the clique with the largest contribution to $c_{CLQ}(\gamma)$, implying $c^*(C_i, \gamma_{i-1}, \gamma_i) \geq c_{CLQ}(\gamma)/m$. □

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

$$\max \sum_{i,j\in O} o_{i,j}\rho_{i,j} \tag{7a}$$

subject to

$$\sum_{j\in C|j\neq i} \rho_{i,j} \leq 1 \qquad \forall C \in \mathbb{C}, i \in C \tag{7b}$$

$$\rho_{i,j} \in \{0, 1\} \qquad \forall i, j \in O \tag{7c}$$

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 (7b) 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 5.1. *IP (7) 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 (2) is equivalent to minimizing the

assignment's cost. It then suffices to argue that a solution of (7) 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 (7); constraint (7b) 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 (7). 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 (7b) 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 (7) is similar in size to IP (1), having $O(n^2)$ variables and $O(nm)$ constraints compared to $O(nb_{\max})$ variables and $O(\max(n, |T|b_{\max}))$ constraints; however, (7) is typically somewhat larger given a reasonable choice of $b_{\max}$. One advantage of (7) over (1) is that it does not exhibit the same level of symmetry. Any feasible solution x, y to (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 6 and empirically in Section 7.

As with (1), we obtain a bound from the linear relaxation of (7). We define z_{DEG} as the optimal objective value for the linear relaxation of IP (7) and c_{DEG} as the equivalent value converted to time-average bins via (2); we refer to the latter as the *degree* bound.

The formulation (7) can be interpreted as a sequence of linked matching problems. Within each clique $C \in \mathbb{C}$, the constraints (7b) 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 [12]:

$$\sum_{i,j \in S} \rho_{i,j} \leq \frac{|S| - 1}{2} \quad \forall S \subseteq C, \quad |S| \text{ odd}. \tag{8}$$

While there are exponentially many blossom inequalities, we can separate over the collection for a single clique in polynomial time [33], and there are $O(n)$ maximal cliques. We use z_{MATCH} to refer to the optimal fractional solution to (7) including blossom inequalities, and c_{MATCH} as the equivalent time-average number of bins via (2). We call the latter

of these the *matching* bound.

5.1 Partition Formulation

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

$$\begin{aligned} & \min \sum_{S \in \mathbb{S}} \ell_S \eta_S && && (9a) \\ \text{subject to} & \\ & \sum_{S \in \mathbb{S}(i)} \eta_S = 1 && \forall i \in J && (9b) \\ & \eta_S \in \{0,1\} && \forall S \in \mathbb{S} && (9c) \end{aligned}$$

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 (9) is potentially exponentially large, with $O(2^n)$ variables; solving it requires special tools such as branch-and-price; the linear relaxation of (9) can be solved using column generation. We use c_{PART} to denote the objective of the linear relaxation of (9) and z_{PART} to be the equivalent optimal time-averaged savings via (2).

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 6.1. $c_{ASGN} \leq c_{STAT}$.

Proof. Consider a feasible, potentially fractional assignment of the x variables in IP (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 $w_t \sum_{b \in \mathbb{B}} y_{t,b} = w_t \sum_{b \in \mathbb{B}} \sum_{i \in J(t)} x_{i,b}/2 = w_t |J(t)|/2 \leq w_t \lceil |J(t)|/2 \rceil$. □

Proposition 6.2. $c_{ASGN} \leq c_{PART}$.

Proof. We show this by arguing that any solution to the linear relaxation of IP (9) implies an equivalent fractional solution to IP (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 (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 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 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 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. In this example, $c_{DEG} = c^* = 11/7$ and $c_{CLQ} = 9/7$. □

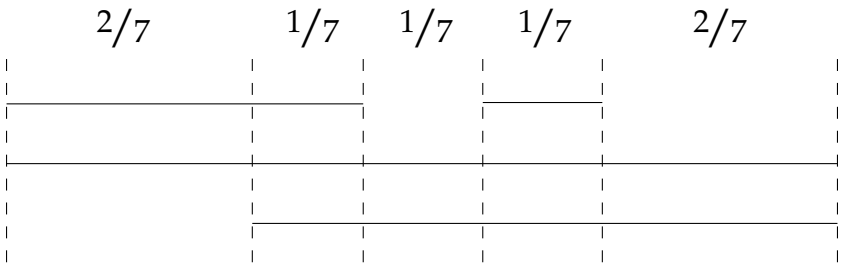

Figure 4: Example with $c_{DEG} > c_{CLQ}$.

Next, we demonstrate that the clique bound is weaker than the linear relaxation of (7) strengthened with blossom inequalities.

Proposition 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

$$\max \sum_{C \in \mathbb{C}} \sum_{i,j \in C} o_{i,j}^{C} \rho_{i,j}^{C} \tag{10a}$$

subject to

$$\sum_{j \in C | j \neq i} \rho_{i,j}^{C} \leq 1 \qquad \forall C \in \mathbb{C}, i \in C \tag{10b}$$

$$\sum_{i,j \in S} \rho_{i,j}^{C} \leq \frac{|S|-1}{2} \qquad \forall C \in \mathbb{C}, S \subseteq C, |S| \text{ odd} \tag{10c}$$

$$\rho_{i,j}^{C} = \rho_{i,j}^{D} \qquad \forall C, D \in \mathbb{C}, i, j \in C \cap D \tag{10d}$$

$$\rho \geq 0 \tag{10e}$$

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 (10) has optimal objective z_{MATCH}. Now consider the relaxation in which constraints (10d) 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 (10) 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 6.6. $c_{PART} = c_{DEG}$.

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 (7) 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 1.

Theorem 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 4.1, 6.1, 6.2, 6.3, 6.4, 6.5, and 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.

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 (7) against the more standard (1). Second, we assess the empirical performance of the various bounds we compared theoretically in Section 6, with a focus on bound quality and scalability. We do not include c_{PART} and c_{ASGN} in this comparison; by Theorem 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 [19, 40].

7.1 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.

7.2 Heuristics

Next, we briefly discuss the heuristics we used to obtain upper bounds in our experiments. The first of these methods comes from IP (7). 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, i.e. 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

$$\max \sum_{i \in J} \tilde{w}_i x_i \tag{11a}$$

subject to

$$\sum_{i \in C} x_i \leq 2 \qquad \forall C \in \mathbb{C} \tag{11b}$$

$$x_i \in \{0,1\} \tag{11c}$$

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 (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.

7.3 IP Formulation Comparison

In the first experiment, we compare the performance of (1) and (7). 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 (7) 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 8.

		IP (7)			IP (1)		
$n = s_{\max}$	λ	UB	LB	% Gap	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 8: IP (1), IP (7) averaged results on random instances with a 600s time limit.

In each row of the table, we highlight the best average upper and lower bounds. On average, (7) obtains both better upper and lower bounds compared to (1), for all instance classes. Consequently, (7) also leads to significantly improved gaps compared to (1). For

all 200- and 400-job instances, (7) has average gaps under 1% while (1) has gaps in the range of 5% to 8%. Even for the worst parameter combination for (7), 600 jobs with expected lifetime of 80, the average gap is only 1.53%. Additionally, for (7) gaps tend to increase with density, while for (1) the gaps are fairly consistent across both size and density.

7.4 Lower Bound Comparison

We now report on two experiments comparing the performance of the bounds discussed in Section 6. First, we compare the bounds on instances with average job lengths $\lambda \in \{5, 10, 20\}$, which are relatively sparse. We use $n = s_{\max} \in \{800, 1600, 2400\}$, 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 (7) with blossom inequalities as described in Section 7.2, 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 9.

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 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 $n = s_{\max} \in \{800, 900, 1000\}$ 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 7.2, with the local search running until convergence; this takes less than 1,600 seconds per realization. We summarize results in Table 10.

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 9: 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 10: Comparison of c_{STAT}, c_{CLQ}, c_{DEG}, and $\hat{c}_{MATCH}$ on dense instances.

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 7.1. For each of these instances, we solve (7) 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 11 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 (7), c_{STAT}, c_{CLQ}, and corresponding running times. We highlight upper and lower bounds when they are provably optimal.

			IP (7)			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 11: Evaluation of IP (7), 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 (7) 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, (7) 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 (7) takes over an hour to solve, and c_{CLQ} also takes significantly longer.

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 [10]. We compute upper bounds using the previously described methods; see Section 7.2.

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 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 12.

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 improves on them for the denser instances. In terms of scaling, using the al-

		c_{STAT}		$c_{CLQ}(\hat{\gamma})$		c_{DEG}	
$n = s_{\max}$	λ	Gap	Time	Gap	Time	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 12: Comparison of c_{STAT}, $c_{CLQ}(\hat{\gamma})$, and c_{DEG} on very large instances.

ternate 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.

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.

References

[1] Mansoor Alicherry and Randeep Bhatia. Line system design and a generalized coloring problem. In *European Symposium on Algorithms*, pages 19–30. Springer, 2003.

[2] Nurşen Aydın, İbrahim Muter, and Ş İlker Birbil. Multi-objective temporal bin packing problem: An application in cloud computing. *Computers & Operations Research*, 121:104959, 2020.

[3] Filipe Brandao and Joao Pedro Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56–67, 2016.

[4] Niv Buchbinder, Yaron Fairstein, Konstantina Mellou, Ishai Menache, and Joseph (Seffi) Naor. Online virtual machine allocation with predictions. arxiv.org/abs/2011.06250, 2020.

[5] Jessica Chang, Samir Khuller, and Koyel Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. *Journal of Scheduling*, 20(6):657–680, 2017.

[6] Edward G Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin Packing Approximation Algorithms: Survey and Classification. In *Handbook of combinatorial optimization*, pages 455–531. Springer, New York, NY, 2013.

[7] Milan De Cauwer, Deepak Mehta, and Barry 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)*, pages 157–164. IEEE, 2016.

[8] Mauro Dell'Amico, Fabio Furini, and Manuel Iori. A branch-and-price algorithm for the temporal bin packing problem. *Computers & Operations Research*, 114:104825, 2020.

[9] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[10] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON–an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[11] György Dósa and Jirí 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.

[12] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[13] Sándor P Fekete and Jörg Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical programming*, 91(1):11–31, 2001.

[14] Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Hadas Shachnai, Mordechai Shalom, Tami Tamir, and Shmuel Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *Theoretical Computer Science*, 411(40-42):3553–3562, 2010.

[15] Fabio Furini and Xueying Shen. Matheuristics for the temporal bin packing problem. In *Recent Developments in Metaheuristics*, pages 333–345. Springer, 2018.

[16] Michael R Garey and David S Johnson. *Computers and intractability*. Freeman, San Francisco, California, 1979.

[17] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[18] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean:{VM} allocation service at scale. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 845–861, 2020.

[19] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[20] Katrin Heßler, Timo Gschwind, and Stefan Irnich. Stabilized branch-and-price algorithms for vector packing problems. *European Journal of Operational Research*, 271(2):401–419, 2018.

[21] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017.

[22] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[23] Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

[24] Narendra Karmarkar and Richard M Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 312–320. IEEE, 1982.

[25] Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami 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.

[26] Lawrence T. Kou and George Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and development*, 21(5):443–448, 1977.

[27] Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 152–163. Springer, 2005.

[28] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

[29] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

[30] J Martinovic, N Strasdat, and M Selch. Compact integer linear programming formulations for the temporal bin packing problem with fire-ups. *Computers & Operations Research*, 132:105288, 2021.

[31] K Maruyama, SK Chang, and DT Tang. A general packing algorithm for multidimensional resource requirements. *International Journal of Computer & Information Sciences*, 6(2):131–149, 1977.

[32] George B Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence WH Wong, and Shmuel Zaks. Optimizing busy time parallel machines. *Theoretical Computer Science*, 562:524–541, 2015.

[33] Manfred W Padberg and M Ram Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[34] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. Technical report, Microsoft Research, 2011. https://www.microsoft.com/en-us/research/wp-content/uploads/2011/01/VBPackingESA11.pdf.

[35] Runtian Ren and Xueyan 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*, pages 227–237, 2016.

[36] Thomas Rothvoss. Approximating bin packing within $O(\log \mathrm{OPT} * \log \log \mathrm{OPT})$ bins. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 20–29, 2013.

[37] David Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL)*, 41(4):579–585, 1994.

[38] Benjamin Speitkamp and Martin Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Transactions on services computing*, 3(4):266–278, 2010.

[39] Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–831, 2003.

[40] Fan You, 2018. github.com/FanYouCN/BlossomSeparationPadbergRao.

A Remaining Proofs

A.1 Proof of Proposition 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. *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 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.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. 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 $\epsilon \to 0$. □

The proposition follows by combining Lemmas A.2 and A.1.

A.2 Proof of Proposition 3.2

We prove this result by by showing that a sequence of the instances described in Section 3.3 has an increasing additive gap $c^* - c_{STAT}$. Consider instances with $|T| = 4k-1$ for some positive integer k, $\hat{t} = (|T|+1)/2$, and uniform period weights. These parameters imply $n = (|T|+1)^2/4$, which is even given our choice of $|T|$. 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 $(|T|+1)/2$. As we have an even number of jobs in each period, $c_{STAT} = \sum_{t \in T} |J(t)|/2|T|$.

As this instance has a single maximal clique, we can use a matching formulation [32]. Consider the following formulation,

$$\min \sum_{i,j \in J} c_{i,j} x_{i,j} \tag{12a}$$

subject to

$$\sum_{j \in J | j \neq i} x_{i,j} \geq 1 \qquad \forall i \in J \tag{12b}$$

$$x_{i,j} \in \{0,1\} \qquad \forall i,j \in J \tag{12c}$$

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

Let $\hat{c}_{i,j} = \sum_{i,j \in J} \mathbb{1}_{\{|\{i,j\} \cap J(t)| = 1\}} / 2|T|$. 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 (12) as $\sum_{i,j} c_{i,j} x_{i,j} = c_{STAT} + \sum_{i,j} \hat{c}_{i,j} x_{i,j}$. The quantity $c_{STAT} = \sum_{t \in T} |J(t)|/2|T|$ does not depend on matching decisions, so we can equivalently optimize (12) with this new objective.

Next, we construct a solution for (12). 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 $2|T|\hat{c}_{i,j} = 1$. As n is even, the cost of this solution under the objective $\hat{c}$ is $(n/2) \times (1/2|T|) = n/4|T|$.

We show this solution is optimal by producing a corresponding dual bound. Consider the dual of the linear relaxation of (12), with edge costs $2|T|\hat{c}_{i,j}$,

$$\max \sum_{i \in J} y_i \tag{13a}$$

subject to

$$y_i + y_j \leq \sum_{t \in T} \mathbb{1}_{\{|\{i,j\} \cap J(t)|=1\}} \qquad \forall i,j \in J \tag{13b}$$

$$y_i \geq 0 \qquad \forall i \in J \tag{13c}$$

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 (13b) 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 $2|T|$, we obtain a feasible dual solution for the LP relaxation of (12) with objective value $n/4|T|$, implying our primal solution is optimal. Therefore,

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

and the result follows as $|T|$ can be made arbitrarily large.

A.3 Proof of Proposition 6.6

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

Lemma A.3. $z_{PART} \leq z_{DEG}$.

Proof. We argue than any fractional solution to the maximization version of (9) implies a corresponding fractional solution to (7) with the same objective value. Given a fractional solution η to the partition formulation, we construct a solution ρ by setting $\rho_{i,j} = \sum_{S \in \$, S \ni i,j} \eta_S$, for job pairs $i, j \in O$.

To see that ρ is feasible, suppose it violates constraint (7b) for some clique C and job i; this implies $1 < \sum_{j \in C | j \neq i} \rho_{i,j} \leq \sum_{S \in \$(i)} \eta_S$, but this contradicts the feasibility of η for the LP relaxation of (9). 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

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

Lemma A.4. $z_{DEG} \leq z_{PART}$.

Proof. Take a solution ρ to the linear relaxation of (7). We make use of the following LP to obtain a minimally infeasible projection of the solution ρ into the space of η variables:

$$\min \sum_{i \in J} \nu_i \tag{14a}$$

subject to

$$\sum_{S \in \mathbb{S}(i)} \eta_S \leq 1 + \nu_i \qquad \forall i \in J \tag{14b}$$

$$\sum_{S \in \mathbb{S}, S \ni i,j} \eta_S = \rho_{i,j} \qquad \forall i,j \in J \tag{14c}$$

$$\eta, \nu \geq 0 \tag{14d}$$

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 (14) 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 (14).

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 (7) is rational, a solution to its LP relaxation is rational; this implies that a solution to (14) 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 (9) 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 (14) has $\sum_{i \in J} \nu_i = 0$, implying that we can always find a fractional solution to (9) matching the given fractional solution to (7). □