Algorithms for Packing and Scheduling
Second Year Progress Report

Student: Mihai Burcea
Supervisors: Prudence W.H. Wong and Russell Martin
Advisors: Leszek A. Gąsieniec, Mingyu Guo, and Piotr Krysta

Department of Computer Science, University of Liverpool, UK
m.burcea@liverpool.ac.uk

Abstract. This is the Second Year Progress Report for the "Algorithms for Packing and Scheduling" Ph.D. project. In this report, we give the aims of the project, a summary of the current results obtained, we address the questions raised in the Postgraduate Workshop, and we conclude with possible future work and a timetabled research plan.
The project is concerned with the study, design, and analysis, of deterministic online and offline algorithms. We present the current results obtained for packing and scheduling algorithms. More specifically, for the online dynamic bin packing problem we first present a new lower bound of $8/3 \sim 2.666$ for the one-dimensional model. Secondly, we give algorithms for the two- and three-dimensional model when the input consists of unit fraction (lengths are $\frac{1}{k}$, for some integer $k \geq 1$) and power fraction (lengths are $\frac{1}{2^k}$, for some integer $k \geq 0$) items. For scheduling algorithms, we present a polynomial time optimal offline algorithm for minimizing the electricity cost in Smart Grid. We consider the model in which the power requirement and the duration a request needs service are both unit-size. A possible future direction is to consider a generalization of this model.

1 Aims of the project

This project is concerned with the study of algorithms for packing and scheduling. The aims of the project are the study of existing algorithms for packing and scheduling problems, design and analysis of improved algorithms for these problems, formulation of new problems for scheduling, and possibly experimental studies for some of the problems. In particular, for packing problems we look at the online dynamic bin packing problem, while for scheduling problems we consider an offline problem where we aim to minimize the total electricity cost in Smart Grid.

2 Summary of results

In this section, we introduce the two types of problems we consider, for online packing algorithms and for an offline scheduling algorithm. For both types, we give an introduction to the problem and summarize the results obtained.

2.1 The Bin Packing Problem

Bin packing is an NP-hard [15] classical combinatorial optimization problem that has been studied since the early 70's and continues to attract researchers' attention (see [9, 10, 13]). The problem was first studied in one dimension (1-D), and has been extended to multiple dimensions (d-D, where $d \geq 1$). In d-D packing, the bins have lengths all equal to 1, while items are of lengths in $(0, 1]$ in each dimension. The objective of the problem is to pack the items into a minimum number of unit-sized bins such that the items do not overlap and do not exceed the boundary of the bin. The items are oriented and cannot be rotated.

We look at the online dynamic setting, where the online setting means that items arrive over time, while their lengths and arrival times are not known until the items arrive. The dynamic setting, introduced by Coffman, Garey, and Johnson [18], means that the items may depart at arbitrary times. Note that the departure times are also not known until these occur. The items arrive over time, reside for some period of time, and may depart at arbitrary times. Each item must be packed to a bin from its arrival to its departure. Migration to another bin is not allowed, yet rearrangement of items within a bin is allowed. The objective is to minimize the maximum number of bins used over all time.

The performance of an online algorithm is measured using competitive analysis [4]. Consider any online algorithm $\mathcal{A}$. Given an input I, let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be c-competitive if there exists a constant b such that $\mathcal{A}(I) \leq c\,OPT(I) + b$ for all I.

An 8/3 Lower Bound for Online Dynamic Bin Packing

We study the one-dimensional online dynamic bin packing problem and improve the lower bound for any deterministic online algorithm from 2.5 [7] to $8/3 \sim 2.666$. Previously, the lower bounds were 2.388 [12], improved to 2.428 [6] and then 2.5 [7]. The new lower bound of $8/3 \sim 2.666$ makes a big step forward to close the gap with the upper bound, which currently stands at 2.788 [12]. The improvement stems from an adversarial sequence that forces an online algorithm $\mathcal{A}$ to open $2s$ bins with items having a total size of s only and this can be adapted appropriately regardless of the load of current bins opened by $\mathcal{A}$.

We design two operations that release items of slightly increasing sizes and items with complementary sizes. These operations make a more systematic approach to release items: the type of item sizes used in a later case is a superset of those used in an earlier case. This is in contrast to the previous 2.5 lower bound in [7] in which rather different sizes are used in different cases. We also show that the new operations defined lead to a much easier proof for a 2.5 lower bound.

The result is published in *Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC)*, 2012, pp. 44–53. This is the first attached paper as part of this report.

Online Multi-dimensional Dynamic Bin Packing of Unit-Fraction Items

While earlier studies on the online multi-dimensional dynamic bin packing problem had as input items with lengths real values in $(0, 1]$, we focus our study on unit fraction items and power fraction items. Unit fraction items are items with lengths of the form $\frac{1}{k}$, for some integer k. Bar-Noy et al. [2] initiated the study of the *unit fraction bin packing problem*, a restricted version where all sizes of items are of the form $\frac{1}{k}$, for some integer k. The problem was motivated by the window scheduling problem [1, 2]. We extend the study to two and three dimensions, providing algorithms for which the competitive ratio reduces from the general items case, i.e., items with lengths real values in $(0, 1]$. We also look at power fraction items, where sizes are of the form $\frac{1}{2^k}$, for some integer k.

For unit fraction items, we give a scheme that divides the items into classes and show that applying the first-fit algorithm to each class is 6.7850- and 21.6108-competitive for 2-D and 3-D, respectively, unit fraction items. This is in contrast to the 7.4842 and 22.4842 competitive ratios for 2-D and 3-D, respectively, that would be obtained using only existing results for unit fraction items. For power fraction items, we use the same approach and the competitive ratios reduce to 6.2455 and 20.0783 for 2-D and 3-D, respectively.

These results are published in *Proceedings of the 8th International Conference on Algorithms and Complexity (CIAC)*, 2013, pp. 85–96. This is the second attached paper as part of this report.

2.2 Scheduling for Electricity Cost in Smart Grid

We study an offline scheduling problem arising in "demand response management" in smart grid [16, 17, 21]. Consumers send in power requests with a flexible set of timeslots during which their requests can be served. The smart grid uses information and communication technologies in an automated fashion to improve the efficiency and reliability of production and distribution of electricity. The grid controller, upon receiving power requests, schedules each request within the specified duration. The electricity cost is measured by a convex function of the load in each timeslot. The objective of the problem is to schedule all requests with the minimum total electricity cost. As a first attempt, we consider a special case in which the power requirement and the duration a request needs service are both unit-size.

More formally, in this offline scheduling problem the input consists of a set of unit-sized jobs $\mathcal{J}$. The time is divided into integral timeslots $T = \{1, 2, 3, \ldots, n\}$ and each job $J_i \in \mathcal{J}$ is associated with a set of feasible timeslots $I_i \subseteq T$, in which it can be scheduled. In this model, each job J_i must be assigned to exactly one feasible timeslot from I_i. The *load* $\ell(t)$ of a timeslot t represents the total number of jobs assigned to the timeslot. We consider a general convex cost function f that measures the cost used in each timeslot t based on the load at t. The total cost used is the sum of cost over time. Over all timeslots this is $\sum_{t \in T} f(\ell(t))$.

The objective is to find an assignment of all jobs in $\mathcal{J}$ to feasible timeslots such that the total cost is minimized.

We propose a polynomial time offline algorithm and show that it is optimal. We show that the time complexity of the algorithm is $\mathcal{O}(|\mathcal{J}|n(|\mathcal{J}|+n))$, where $\mathcal{J}$ is the set of jobs and n is the number of timeslots. We further show that if the feasible timeslots for each job to be served forms a contiguous interval, we can improve the time complexity to $\mathcal{O}(|\mathcal{J}|n(\log|\mathcal{J}|+\log n))$ using dynamic range minimum query data structure.

The result is to be submitted. This is the third attached paper as part of this report.

3 Workshop feedback

We would like to thank the advisors for their feedback on the Postgraduate Workshop. We address the questions raised, specifically other metrics to define competitiveness of online algorithms, a note on the current lower bound for one-dimensional online dynamic bin packing, and LP-based techniques for analyis of online algorithms and for the offline algorithm.

Metrics for competitiveness. Usually, the performance of online algorithms is measured using competitive analysis [4]. Using competitive analysis, we provide a worst-case guarantee for the performance of an online algorithm. While this guarantees the algorithms' performance on any input, we recognize that other metrics are useful. For online *static* bin packing, where items are permanent, other benchmarks such as average case analysis have been studied [8, 11]. The difference from worst-case analysis is that average case analysis relies on input distribution and thus may not consider every possible input. Other ways to measure competitiveness are using random order analysis [19], relative worst order analysis [5], and advice complexity analysis [3, 14]. It may be an interesting problem of its own to introduce other metrics for online *dynamic* bin packing.

Lower bound for one-dimensional online dynamic bin packing. The new lower bound of $8/3 \sim 2.666$ is currently the best known for any deterministic online algorithm. At the moment, we have not formally explored how to prove that the technique used in the adversary cannot give a better lower bound. Informally, however, it seems that the items we chose for the adversary are the best for the technique we use. To illustrate this, suppose there are k existing bins each containing exactly one item of size ϵ, for a small ϵ. Assuming that we keep the maximum load (total size of items) to at most n, then we can release $n-\epsilon k$ items of sizes $1/4-i\delta$, for some integer i and small δ. Suppose we use items of these sizes to open new bins. Then any algorithm will open at least $k/4$ new bins as each existing of the k bins can pack at most three of these items. We depart items such that each of the k bins contains exactly one ϵ-item and each of the newly opened $k/4$ bins contains a $1/4-i\delta$-item. We can release items of complementary sizes $3/4+i\delta$ to open an additional $k/4$ new bins (these processes take place in phases and can be constructed using Op-Inc and Op-Comp from the paper). Thus, we have opened $k/4+k/4=k/2$ new bins

using $1/4-i\delta$ and $3/4+i\delta$-items. Contrastingly, if we use items of sizes $1/5-i\delta$ and $4/5+i\delta$ we can open at most $k/5+k/5 = 2k/5$ new bins. Thus, using smaller items we can only open fewer number of new bins. Using items of size $1/3-i\delta$ and $2/3+i\delta$ will open more bins initially, but will not allow for the maximum number of bins to be opened which is obtained by using $1/4-i\delta$, $3/4+i\delta$ and $1/2-i\delta$, $1/2+i\delta$-items. On the other hand, using $1/3-i\delta$, $2/3+i\delta$ and $1/2-i\delta$, $1/2+i\delta$- items would result in exceeding the maximum load of n, which would not be a feasible adversary. It would be worthwhile to have a formal proof that shows we can achieve the best lower bound with this technique if and only if we use the combination of $\epsilon, 1/4 - i\delta, 1/2 - i\delta, 1/2 + i\delta, 3/4 + i\delta$, and 1-items.

LP techniques for analysis. We have not yet considered LP techniques in order to analyse the performance of online algorithms or to prove the optimality of the offline scheduling algorithm. This may be a future direction.

4 Future work

In the Scheduling for Electricity Cost in Smart Grid paper, we have considered the model in which the power requirement and the duration a request needs service are both unit-size. One possible future direction is to generalize the offline scheduling problem to the models where jobs have arbitrary power requirements or arbitrary duration. When jobs have both arbitrary power requirements and duration the problem is NP-Hard [20].

Other possible directions of work include further improvement on online dynamic bin packing of unit and power fraction items and a bandwidth allocation problem in optical networks that aims to maximize the profit of a network owner that schedules transmissions (jobs) between two points. For completeness, we show the model we consider here. Note that this problem may be considered in both offline and online settings.

Bandwidth allocation. We are given a sequence of jobs or paths $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Each job J_i is associated with a start time s_i and end time t_i, where $s_i < t_i$ and the length of J_i is denoted by $len_i = t_i - s_i$. Without loss of generality, we can assume that s_i and t_i are positive integers. At any given time t there are a minimum and maximum number of jobs present in the system, denoted by l_{min} and l_{max}, respectively. In this setting and at any time t, we have some resources available to be allocated to jobs. The amount of resources to be allocated is represented by an integer $W > 1$ and we have that $l_{min} \leq l_{max} \leq W$. In other words, we have at least as many resources to be allocated to the jobs at any time as the number of jobs present.

The resources to be allocated (W) are represented by bandwidth. There are W different bandwidth wavelengths to be assigned and each job must receive at least one bandwidth wavelength, that is, for every job J_i, $b_i \geq 1$. The different wavelengths can be thought of as distinct colours from the set $\{1, \ldots, W\}$. At any time t, no two jobs can be assigned the same colour. Furthermore, once a colour has been allocated to a job J_i it cannot be changed, i.e., it remains the

same for the duration of $[s_i, t_i]$. However, another job J_j with $s_j = t_i$ can share the same colour with J_i.

There are two different models for the bandwidth allocation problem. In the first and less constrained model, for each job J_i that receives a bandwidth $b_i > 1$, it can be allocated any non-contiguous colours from the W available, as long as the wavelengths have not been allocated to any other job for the duration of $[s_i, t_i)$. Conversely, for the second model, the colours must be contiguous for any job J_i with $b_i > 1$. The objective of the problem is to maximize the profit obtained by allocating bandwidth to all jobs in $\mathcal{J}$. The profit of a job J_i is $p_i = b_i \cdot len_i$. Thus, we aim to maximize $\sum_{J_i \in \mathcal{J}} p_i$.

Plan

	Timeline	Description of activities
Year 1	Months 1-2	Background reading: online algorithms analysis, energy-efficient scheduling and routing algorithms.
	Months 3-10	Contribution to two papers: – P.W.H. Wong, F.C.C. Yung, and M. Burcea. An 8/3 Lower Bound for Online Dynamic Bin Packing. *ISAAC*, 2012, pp. 44–53; – M. Burcea, P.W.H. Wong, and F.C.C. Yung. Online Multi-dimensional Dynamic Bin Packing of Unit-Fraction Items. *CIAC*, 2013, pp. 85–96.
	Months 11-12	Attempt at the windows scheduling problem, further reading on energy-efficient scheduling and routing, and scheduling with a thermal threshold.
Year 2	Months 1-2	Observations on the bandwidth allocation problem, problem definition, and consider the offline and online models.
	Months 3-7	Contribution to one paper: – M. Burcea, W.-K. Hon, H.-H. Liu, P.W.H. Wong, and D.K.Y. Yau. Scheduling for Electricity Cost in Smart Grid. Submitted. 2013.
	Months 8-12	Complete formal requirements for Year 2. Consider the generalization of the problem of scheduling for electricity cost in Smart Grid.
Year 3	Months 1-6	Possible further work on the bandwidth allocation problem and improvement on the online dynamic bin packing problem of unit fraction items.
	Months 7-12	Conclusions and thesis write-up.

References

1. Amotz Bar-Noy and Richard E. Ladner. Windows scheduling problems for broadcast systems. *SIAM J. Comput.*, 32:1091–1113, April 2003.
2. Amotz Bar-Noy, Richard E. Ladner, and Tami Tamir. Windows scheduling as a restricted version of bin packing. *ACM Trans. Algorithms*, 3, August 2007.
3. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 331–340, Berlin, Heidelberg, 2009. Springer-Verlag.

4. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
5. Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Trans. Algorithms*, 3(2), May 2007.
6. Joseph Wun-Tat Chan, Tak Wah Lam, and Prudence W. H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(3):172–206, 2008.
7. Joseph Wun-Tat Chan, Prudence W. H. Wong, and Fencol C. C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis. *Algorithmica*, 53(2):172–206, 2009.
8. E. G. Coffman, Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, Part I: Perfect packing theorems and the average case behavior of optimal packings. *SIAM J. Discrete Math.*, 13:38–402, 2000.
9. E. G. Coffman Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, 1998.
10. E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publishing, 1996.
11. E. G. Coffman, Jr., D. S. Johnson, L. A. McGeoch, P. W. Shor, and R. R. Weber. Bin packing with discrete item sizes, Part III: Average case behavior of FFD and BFD. In preparation.
12. Edward G. Coffman, Jr., M. R. Garey, and David S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983.
13. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In A. Fiat and G. J. Woeginger, editors, *On-line Algorithms–The State of the Art*, pages 147–177. Springer, 1996.
14. Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theoretical Informatics and Applications*, 43:585–613, 6 2009.
15. Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
16. K. Hamilton and N. Gulhar. Taking demand response to the next level. *Power and Energy Magazine, IEEE*, 8(3):60–65, 2010.
17. A. Ipakchi and F. Albuyeh. Grid of the future. *IEEE Power and Energy Magazine*, 7(2):52–62, 2009.
18. Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983.
19. Claire Kenyon. Best-fit bin-packing with random order. In *In 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1997.
20. Iordanis Koutsopoulos and Leandros Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *Proc. e-Energy*, pages 41–50, 2011.
21. T.J. Lui, W. Stirling, and H.O. Marcy. Get smart. *IEEE Power and Energy Magazine*, 8(3):66–78, 2010.

An 8/3 Lower Bound for Online Dynamic Bin Packing

Prudence W.H. Wong, Fencol C.C. Yung, and Mihai Burcea*

Department of Computer Science, University of Liverpool, UK
{pwong,m.burcea}@liverpool.ac.uk, ccyung@graduate.hku.hk

Abstract. We study the dynamic bin packing problem introduced by Coffman, Garey and Johnson. This problem is a generalization of the bin packing problem in which items may arrive and depart dynamically. The objective is to minimize the maximum number of bins used over all time. The main result is a lower bound of $8/3 \sim 2.666$ on the achievable competitive ratio, improving the best known 2.5 lower bound. The previous lower bounds were 2.388, 2.428, and 2.5. This moves a big step forward to close the gap between the lower bound and the upper bound, which currently stands at 2.788. The gap is reduced by about 60% from 0.288 to 0.122. The improvement stems from an adversarial sequence that forces an online algorithm $\mathcal{A}$ to open $2s$ bins with items having a total size of s only and this can be adapted appropriately regardless of the current load of other bins that have already been opened by $\mathcal{A}$. Comparing with the previous 2.5 lower bound, this basic step gives a better way to derive the complete adversary and a better use of items of slightly different sizes leading to a tighter lower bound. Furthermore, we show that the 2.5-lower bound can be obtained using this basic step in a much simpler way without case analysis.

1 Introduction

Bin packing is a classical combinatorial optimization problem [6, 8, 9]. The objective is to pack a set of items into a minimum number of unit-size bins such that the total size of the items in a bin does not exceed the bin capacity. The problem has been studied extensively both in the offline and online settings. It is well-known that the problem is NP-hard [11]. In the online setting [14, 15], items may arrive at arbitrary time; item arrival time and item size are only known when an item arrives. The performance of an online algorithm is measured using competitive analysis [3]. Consider any online algorithm $\mathcal{A}$. Given an input I, let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be c-competitive if there exists a constant b such that $\mathcal{A}(I) \leq c\,OPT(I) + b$ for all I.

Online dynamic bin packing. Most existing work focuses on "static" bin packing in the sense that items do not depart. In some potential applications like

* Supported by EPSRC Studentship.

warehouse storage, a more realistic model takes into consideration of dynamic arrival and departures of items. In this natural generalization, known as dynamic bin packing [7], items arrive over time, reside for some period of time, and may depart at arbitrary time. Each item has to be assigned to a bin from the time it arrives until it departs. The objective is to minimize the maximum number of bins used over all time. Note that migration to another bin is not allowed. In the online setting, the size and arrival time is only known when an item arrives and the departure time is only known when the item departs.

In this paper, we focus on online dynamic bin packing. It is shown in [7] that First-Fit has a competitive ratio between 2.75 and 2.897, and a modified first-fit algorithm is 2.788-competitive. A lower bound of 2.388 is given for any deterministic online algorithm. This lower bound has later been improved to 2.428 [4] and then 2.5 [5]. The problem has also been studied in two- and three-dimension as well as higher dimension [10, 16]. Other work on dynamic bin packing considered a restricted type of items, namely unit-fraction items [2, 4, 12]. Furthermore, Ivkovic and Lloyd [13] studied the fully dynamic bin packing problem, which allows repacking of items for each item arrival or departure and they gave a 1.25-competitive online algorithm for this problem. Balogh et al. [1] studied the problem when a limited amount of repacking is allowed.

Our contribution. We improve the lower bound of online dynamic bin packing for any deterministic online algorithm from 2.5 to $8/3 \sim 2.666$. This makes a big step forward to close the gap with the upper bound, which currently stands at 2.788 [7]. The improvement stems from an adversarial sequence that forces an online algorithm $\mathcal{A}$ to open $2s$ bins with items having a total size of s only and this can be adapted appropriately regardless of the load of current bins opened by $\mathcal{A}$. Comparing with the previous 2.5 lower bound, this basic step gives a better use of items of slightly different sizes leading to a tighter lower bound. Furthermore, we show in Section 3.3 that the 2.5-lower bound can be obtained using this basic step in a much simpler way without case analysis. It is worth mentioning that we consider optimal packing without migration at any time.

The adversarial sequence is composed of two operations, namely Op-Inc and Op-Comp. Roughly speaking, Op-Inc uses a load of at most s to make $\mathcal{A}$ open s bins, this is followed by some item departure such that each bin is left with only one item and the size is increasing across the bins. Op-Comp then releases items of complementary size such that for each item of size x, items of size $1 - x$ are released. The complementary size ensures that the optimal offline algorithm $\mathcal{O}$ is able to pack all these items using s bins while the sequence of arrival ensures that $\mathcal{A}$ has to pack these complementary items into separate bins.

2 Preliminaries

In dynamic bin packing, items arrive and depart at arbitrary time. Each item comes with a size. We denote by *s-item* an item of size s. When an item arrives, it must be assigned to a unit-sized bin immediately without exceeding the bin capacity. At any time, the *load* of a bin is the total size of items currently

assigned to that bin that have not yet departed. We denote by ℓ-*bin* a bin of load ℓ. Migration is not allowed, i.e., once an item is assigned to a bin, it cannot be moved to another bin. This also applies to the optimal offline algorithm. The objective is to minimize the maximum number of bins used over all time.

When we discuss how items are packed, we use the following notations:

- Item configuration ψ: y_{*z} describes a load y with $\frac{y}{z}$ items of size z, e.g., $\frac{1}{2}_{*\epsilon}$ means a load $\frac{1}{2}$ with $\frac{1}{2\epsilon}$ items of size ϵ. We skip the subscript when $y = z$.
- Bin configuration π: $(\psi_1, \psi_2, \cdots)$, e.g., $(\frac{1}{3}, \frac{1}{2}_{*\epsilon})$ means a bin has a load of $\frac{5}{6}$, with a $\frac{1}{3}$-item and an addition load $\frac{1}{2}$ with ϵ-items. In some cases, it is clearer to state the bin configuration in other ways, e.g., $(\frac{1}{2}, \frac{1}{2})$, instead of $1_{*\frac{1}{2}}$. Similarly, we will use $6 \times \frac{1}{6}$ instead of $1_{*\frac{1}{6}}$.
- Packing configuration ρ: $\{x_1{:}\pi_1, x_2{:}\pi_2, \cdots\}$ a packing where there are x_1 bins with bin configuration π_1, x_2 bins with π_2, and so on. E.g., $\{2k{:}1_{*\epsilon}, k{:}(\frac{1}{3}, \frac{1}{2}_{*\epsilon})\}$ means $2k$ bins are each packed with load 1 with ϵ-items and another k bins are each packed with a $\frac{1}{3}$-item and an addition load $\frac{1}{2}$ with ϵ-items.
- It is sometimes more convenient to describe a packing as $x{:}f(i)$, for $1 \le i \le x$, which means that there are x bins with different load, one bin with load $f(i)$ for each i. E.g., $k{:}\frac{1}{2}{-}i\delta$, for $1 \le i \le k$, means that there are k bins and one bin with load $\frac{1}{2}{-}i\delta$ for each i.

3 Op-Inc and Op-Comp

In this section, we discuss a process that the adversary uses to force an online algorithm $\mathcal{A}$ to open new bins. The adversary releases items of slightly different sizes in each stage and uses items of complementary sizes in different stages. Two operations are designed, namely, Op-Inc and Op-Comp. Op-Inc forces $\mathcal{A}$ to open some bins each with one item (of size $< \frac{1}{2}$) and the size of items is strictly increasing. Op-Comp then bases on the bins opened by Op-Inc and releases items of complementary size. This is to ensure that an item released in Op-Inc can be packed with a corresponding item released in Op-Comp into the same bin by an optimal offline algorithm. In the adversary, a stage of Op-Inc is associated with a corresponding stage of Op-Comp, but not necessarily consecutive, e.g., in one of the cases, Op-Inc is in Stage 1 and the corresponding Op-Comp is in Stage 4.

3.1 Operation Op-Inc

The aim of Op-Inc is to make $\mathcal{A}$ open at least s more bins, for some $s > 0$, such that each new bin contains one item with item size increasing over the s bins.
Pre-condition. Consider any value $0 < x < \frac{1}{2}$. Let h be the number of x-items that can be packed in existing bins.
Items to be involved. The items to be released have size in the range $[x, x+\epsilon]$, for some small ϵ, such that $x+\epsilon < \frac{1}{2}$. A total of $h + \lfloor \frac{s}{x} \rfloor$ items are to be released.
Outcome. $\mathcal{A}$ opens at least s more new bins with increasing load in each new bin and the load of current bins remains unchanged.

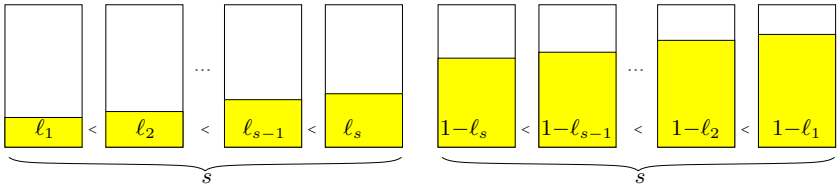


Fig. 1. Op-Comp: Assuming $k = 0$. The s bins on the left are bins created by Op-Inc. The s new bins on the right are due to Op-Comp. Note that each existing item has a complementary new item such that the sum of size is 1.

The adversary. The adversary releases items of size x, $x + \frac{\epsilon}{s}$, $x + \frac{2\epsilon}{s}$, $\cdots$. Let $z_i = x + \frac{i\epsilon}{s}$. In each step i, the adversary releases z_i-items until $\mathcal{A}$ opens a new bin. We stop releasing items when $h + \lfloor \frac{s}{x} \rfloor$ items have been released in total. By the definition of h, s and x, $\mathcal{A}$ would have opened at least s new bins. We then let z_i-items depart except exactly one item of size z_i, for $0 \leq i < s$, in the i-th new bin opened by $\mathcal{A}$.

Using Op-Inc. When we use Op-Inc later, we simply describe it as Op-Inc releasing $h + \lfloor \frac{s}{x} \rfloor$ items with the understanding that it works in phases and that items depart at the end.

3.2 Operation Op-Comp

Op-Comp is designed to work with Op-Inc and assumes that there are s existing bins each with load in the range $[x, y]$ where $x < y < \frac{1}{2}$. The outcome of Op-Comp is that $\mathcal{A}$ opens s more bins. Figure 1 gives an illustration.

Pre-condition. Consider two values $x < y < \frac{1}{2}$. Suppose $\mathcal{A}$ uses s bins with load $x = \ell_1 < \ell_2 < \cdots < \ell_s = y$. Let $\ell = \sum_{1 \leq i \leq s} \ell_i$. Furthermore, suppose there are some additional bins with load smaller than x. Let h be the number of $(1-y)$-items that can be packed in other existing bins with load less than x.

Items to be involved. The items to be released have size in the range $[1-y, 1-x]$. Note that $1 - x > 1 - y > \frac{1}{2}$. In each step i, for $1 \leq i \leq s$, the number of $(1 - \ell_{s+1-i})$-items released is at most $h + s + 2 - i$.

Outcome. $\mathcal{A}$ opens s more bins, each with an item $1 - \ell_{s+1-i}$, for $1 \leq i \leq s$.

The adversary. Starting from the largest load ℓ_s, we release items of size $1-\ell_s$ until $\mathcal{A}$ opens a new bin. At most $h + s + 1$ items are needed. Then we let all $(1-\ell_s)$-items depart except the one packed in the new bin. In general, in Step i, for $1 \leq i \leq s$, we release items of size $1-\ell_{s+1-i}$ until $\mathcal{A}$ opens a new bin. Note that such items can only be packed in the first $s + 1 - i$ bins and so at most $h + s + 2 - i$ items are required to force $\mathcal{A}$ to open another bin. We then let all $(1-\ell_{s+1-i})$-items depart except the one packed in the new bin.

Using Op-Comp. Similar to Op-Inc, when we use Op-Comp later, we describe it as Op-Comp with h and s and the understanding is that it works in phases and there are items released and departure in between. Note that the ℓ_i- and $(1 - \ell_i)$-items are complementary and the optimal offline algorithm would pack each pair of complementary items in the same bin.

3.3 A 2.5 lower bound using Op-Inc and Op-Comp

We demonstrate how to use Op-Inc and Op-Comp by showing that we can obtain a 2.5 lower bound as in [5] using the two operations in a much simpler way.

Let k be some large even integer, $\epsilon = \frac{1}{k}$, and $\delta = \frac{\epsilon}{k+1}$. The adversary works in stages. In Stage 1, we release $\frac{k}{\epsilon}$ items of size ϵ. Any online algorithm $\mathcal{A}$ uses at least k bins. Let items depart until the configuration is $\{k{:}\epsilon\}$. In Stage 2, we aim to force $\mathcal{A}$ to use $\frac{k}{2}$ new bins. We use Op-Inc to release at most $2k$ items of size in $[\frac{1}{2}-\frac{k}{2}\delta, \frac{1}{2}-\delta]$. For each existing ϵ-bin, at most one such new items can be packed because $1-k\delta+\epsilon > 1$. The parameters for Op-Inc are therefore $x = \frac{1}{2}-\frac{k}{2}\delta$, $h = k$ and $s = \frac{k}{2}$. The configuration of $\mathcal{A}$ becomes $\{k{:}\epsilon, \frac{k}{2}{:}\frac{1}{2}-i\delta\}$, for $1 \leq i \leq \frac{k}{2}$. In Stage 3, we aim to force $\mathcal{A}$ to use $\frac{k}{2}$ new bins. We use Op-Comp to release items of size in the range $x = \frac{1}{2}+\delta$ to $y = \frac{1}{2}+\frac{k}{2}\delta$. At most one such item can be packed in the bins with an ϵ-item, i.e., $h = k$. The second $\frac{k}{2}$ bins contains items of complementary size to the items released in Stage 3, i.e., $s = \frac{k}{2}$. Note that at any time during Op-Comp, at most $\frac{3k}{2}+1$ items are released. $\mathcal{A}$ needs to open at least $\frac{k}{2}$ new bins with the configuration $\{k{:}\epsilon, \frac{k}{2}{:}\frac{1}{2}-i\delta, \frac{k}{2}{:}\frac{1}{2}+i\delta\}$, for $1 \leq i \leq \frac{k}{2}$. In the final stage, we release $\frac{k}{2}$ items of size 1 and $\mathcal{A}$ needs a new bin for each of these items. The total number of bins used by $\mathcal{A}$ becomes $\frac{5k}{2}$.

On the other hand, the optimal algorithm $\mathcal{O}$ can use $k+2$ bins to pack all items as follows and hence the competitive ratio is at least 2.5. In Stage 1, all the ϵ-items that never depart are packed in one bin and the rest in $k-1$ bins. In Stage 2, the new items are packed in k bins, with the $\frac{k}{2}$ bins with size $\frac{1}{2}-i\delta$, for $1 \leq i \leq \frac{k}{2}$, that never depart each packed in one bin, and the remaining $\frac{3k}{2}$ items in the remaining space. At the end of the stage, only one item is left in each of the first $\frac{k}{2}$ bins and the second $\frac{k}{2}$ bins are freed for Stage 3. In Stage 3, the complementary items that do not depart are packed in the corresponding $\frac{k}{2}$ bins, and the remaining in at most $\frac{k}{2}+1$ bins. Finally in Stage 4, the 1-items are packed in the $\frac{k}{2}$ bins freed in Stage 3.

4 The 8/3 Lower Bound

We give an adversary such that at any time, the total load of items released and not departed is at most $6k + O(1)$, for some large integer k. We prove that any online algorithm $\mathcal{A}$ uses $16k$ bins, while the optimal offline algorithm $\mathcal{O}$ uses at most $6k + O(1)$ bins. Then, the competitive ratio of $\mathcal{A}$ is at least $\frac{8}{3}$. The adversary works in stages and uses Op-Inc and Op-Comp in pairs. Let n_i be the number of new bins used by $\mathcal{A}$ in Stage i. Let $\epsilon = \frac{1}{6k}$ and $\delta = \frac{\epsilon}{16k}$.

In Stage 0, the adversary releases $\frac{6k}{\epsilon}$ items of size ϵ, with total load $6k$. It is clear that $\mathcal{A}$ needs at least $6k$ bins, i.e., $n_0 \geq 6k$. We distinguish between two cases: $n_0 \geq 8k$ and $8k > n_0 \geq 6k$. We leave the details of the easier first case in the full paper, and we consider only the complex second case in this paper.

Case 2: $6k \leq n_0 < 8k$.

This case involves three subcases. We make two observations about the load of the n_0 bins. If less than $4k$ bins have load at least $\frac{1}{2}+\epsilon$, then the total load of all bins is at most $(4k-1)+4k/2 = 6k-1$, contradicting the fact that total load of items released is $6k$. Similarly, if less than $5k$ bins have load at least $\frac{1}{4}+\epsilon$, then the total load of all bins is at most $(5k-1)+3k/4 < 6k$, leading to a contradiction.

Observation 1 *At the end of Stage* 0 *of Case* 2*, (i) at least* $4k$ *bins have load at least* $\frac{1}{2}+\epsilon$*; (ii) at least* $5k$ *bins have load at least* $\frac{1}{4}+\epsilon$*.*

Stage 1. We aim at $n_1 \geq 2k$. We let ϵ-items depart until the configuration of $\mathcal{A}$ becomes

$$\{4k{:}(\frac{1}{2}{+}\epsilon)_{*\epsilon}, k{:}(\frac{1}{4}{+}\epsilon)_{*\epsilon}, k{:}\epsilon\} \ ,$$

with $6k$ bins and a total load of $9k/4+O(1)$. We then use Op-Inc with $x = \frac{1}{4}+\delta$, $h = 8k$, and $s = 2k$. The first $4k$ bins can pack at most one x-item, the next k bins at most two, and the last k bins at most three, i.e., $h = 9k$. Any new bin can pack at most three items, implying that Op-Inc releases $15k = h+3s$ items of increasing sizes, from $\frac{1}{4}+\delta$ to at most $\frac{1}{4}+15k\delta$. According to Op-Inc, $\mathcal{A}$ opens at least $2k$ bins, i.e., $n_1 \geq 2k$. We consider two subcases: $n_1 \geq 4k$ and $2k \leq n_1 < 4k$.

Case 2.1: $6k \leq n_0 < 8k$ and $n_1 \geq 4k$. In this case, we have $10k \leq n_0 + n_1$.
Stage 2. We aim at $n_2 \geq 4k$. The configuration after Op-Inc becomes

$$\{4k{:}(\frac{1}{4}{+}\epsilon)_{*\epsilon}, k{:}(\frac{1}{4}{+}\epsilon)_{*\epsilon}, k{:}\epsilon, 4k{:}\frac{1}{4}{+}i\delta\} \ , \ \text{for } 1 \leq i \leq 4k,$$

with $10k$ bins and a total load of $9k/4+O(1)$. Note that in the last $4k$ bins, the load increases by δ from $\frac{1}{4}+\delta$ to $\frac{1}{4}+4k\delta$. We now use Op-Comp with $x = \frac{1}{4}+\delta$, $y = \frac{1}{4}+4k\delta$, $h = k$, and $s = 4k$. I.e., Op-Comp releases items of sizes from $\frac{3}{4}-4k\delta$ to $\frac{3}{4}-\delta$ and at any time, at most $5k+1$ items are needed. None of these items can be packed in the first $5k$ bins, and only one can be packed in the next k bins, i.e., $h = k$ as said. According to Op-Comp, $\mathcal{A}$ requires $4k$ new bins.
Stage 3. We aim at $n_3 = 2k$. We let items depart until the configuration becomes

$$\{4k{:}\epsilon, k{:}\epsilon, k{:}\epsilon, 4k{:}\frac{1}{4}{+}i\delta, 4k{:}\frac{3}{4}{-}i\delta\} \ , \ \text{for } 1 \leq i \leq 4k,$$

with $14k$ bins and a load of $4k+O(1)$. We further release $2k$ items of size 1. $\mathcal{A}$ needs to open $2k$ new bins. In total, $\mathcal{A}$ uses $6k+4k+4k+2k = 16k$ bins.

We note that each item with size $\frac{1}{4}+i\delta$ has a corresponding item $\frac{3}{4}-i\delta$ such that the sum of sizes is 1. This allows the optimal offline algorithm to have a better packing. The details will be given in the full paper.

Lemma 1. *If* $\mathcal{A}$ *uses* $[6k, 8k)$ *bins in Stage* 0 *and at least* $4k$ *bins in Stage* 1*, then* $\mathcal{A}$ *uses* $16k$ *bins at the end while* $\mathcal{O}$ *uses* $6k+4$ *bins.*

Case 2.2: $6k \le n_0 < 8k$ and $2k \le n_1 < 4k$. In this case, the Op-Inc in Stage 1 is paired with an Op-Comp in Stage 4 (not consecutively), and in between, there is another pair of Op-Inc and Op-Comp in Stages 2 and 3, respectively. Let m be the number of bins among the n_1 new bins that have been packed two items. We further distinguish two subcases: $m \ge 2k$ and $m < 2k$.

Case 2.2.1: $6k \le n_0 < 8k$, $2k \le n_1 < 4k$ and $m \ge 2k$. In this case, we have $8k \le n_0 + n_1 < 10k$ and $m \ge 2k$. We make an observation about the bins containing some ϵ-items. In particular, we claim that there are at least k bins that are packed with

- either one ϵ-item and at least two $(\frac{1}{4}+i\delta)$-items,
- or one $(\frac{1}{4}+i\delta)$-item plus at least a load of $(\frac{1}{4}+\epsilon)_{*\epsilon}$.

We note that in Stage 1, $15k$ items are released, at most three items can be packed in any of the $n_1 < 4k$ new bins, i.e., at most $12k$ items. So, at least $3k$ of them have to been packed in the first $6k$ bins. Let a and b be the number of bins in the first $5k$ bins (with load at least $\frac{1}{4}+\epsilon$) that are packed at least one $(\frac{1}{4}+i\delta)$-item; z_1, z_2, z_3 be the number of bins in the next k bins (with one ϵ-item) that are packed one, two, and three $(\frac{1}{4}+i\delta)$-items, respectively. Note that $z_1 + z_2 + z_3 = k$. Since $3k$ items have to be packed in these bins, we have $a + 2b + z_1 + 2z_2 + 3z_3 \ge 3k$, hence $a + 2b + z_2 + 2z_3 \ge 2k$. The last inequality implies that $a + b + z_2 + z_3 \ge k$ and the claim holds.

Observation 2 *At the end of Stage* 1 *of Case* 2.2.1*, at least k bins are packed with either one ϵ-item and at least two $(\frac{1}{4}+i\delta)$-items, or one $(\frac{1}{4}+i\delta)$-item plus at least a load of $(\frac{1}{4}+\epsilon)_{*\epsilon}$.*

Stage 2. We aim at $n_2 \ge 2k$. Let $z = z_2 + z_3$. We let items depart until the configuration becomes

$$\{3k{:}(\frac{1}{2}+\epsilon)_{*\epsilon}, k-z{:}((\frac{1}{4}+\epsilon)_{*\epsilon}, \frac{1}{4}+i\delta), z{:}(\epsilon, \frac{1}{4}+i\delta, \frac{1}{4}+i\delta), 2k{:}\epsilon, 2k{:}(\frac{1}{4}+i\delta, \frac{1}{4}+i\delta)\} \ ,$$

with $8k$ bins and a total load of $3k + O(1)$.

Let $\delta' = \frac{\delta}{16k}$. We use Op-Inc with $x = \frac{1}{2}-6k\delta'$, $h = 2k$, and $s = 2k$. The x-items can only be packed in the $2k$ bins with load ϵ, at most one item in one bin, i.e., $h = 2k$. Any new bin can pack at most two, implying that Op-Inc releases $6k = h + 2s$ items of increasing sizes, from $\frac{1}{2}-6k\delta'$ to at most $\frac{1}{2}-\delta'$. According to Op-Inc, $\mathcal{A}$ has to open at least $2k$ new bins, i.e., $n_2 \ge 2k$.

Stage 3. In this stage, we aim at $n_3 \ge 2k$. We use Op-Comp which corresponds to Op-Inc in Stage 2. We let items depart until the configuration becomes

$$\{3k{:}(\tfrac{1}{2}+\epsilon)_{*\epsilon}, k-z{:}((\tfrac{1}{4}+\epsilon)_{*\epsilon}, \tfrac{1}{4}+i\delta), z{:}(\epsilon, \tfrac{1}{4}+i\delta, \tfrac{1}{4}+i\delta), 2k{:}\epsilon, 2k{:}(\tfrac{1}{4}+i\delta, \tfrac{1}{4}+i\delta), 2k{:}\tfrac{1}{2}-i\delta'\} \ ,$$

with $10k$ bins and a total load of $4k + O(1)$. We then use Op-Comp with $x = \frac{1}{2}-6k\delta'$, $y = \frac{1}{2}-5k\delta'$, $h = 2k$, and $s = 2k$. I.e., we release items of increasing size from $\frac{1}{2}+5k\delta'$ to $\frac{1}{2}+6k\delta'$, and at any time, at most $4k + 1$ items are needed. The $2k$ bins of load ϵ can pack one such item. Suppose there are w, out of $2k$,

ϵ-bins that are not packed with a $\frac{1}{2}+i\delta'$-item. According to Op-Comp, $\mathcal{A}$ has to open $2k+w$ new bins.

Stage 4. In this stage, we aim at $n_4 \geq 2k-w$. We use Op-Comp which corresponds to Op-Inc in Stage 1. We let items depart until the configuration is

$$\{3k{:}(\tfrac{1}{4}+\epsilon)_{*\epsilon}, k-z{:}(\tfrac{1}{4}+\epsilon)_{*\epsilon}, z{:}(\epsilon, \tfrac{1}{4}+i\delta), 2k-w{:}(\epsilon, \tfrac{1}{2}+i\delta'), w{:}\epsilon, 2k{:}\tfrac{1}{4}+i\delta, 2k{:}\tfrac{1}{2}-i\delta', 2k+w{:}\tfrac{1}{2}+i\delta', \}\ ,$$

with $12k+w$ bins and a total load of $9k/2+O(1)$. We then use Op-Comp with $x=\frac{1}{4}+\delta$, $y=\frac{1}{4}+2k\delta$, $h=w$, and $s=2k-w$. I.e., we release items of sizes from $\frac{3}{4}-2k\delta$ to $\frac{3}{4}-\delta$ and at any time, at most $2k+1$ items are needed. Only w ϵ-bins can pack such item, i.e., $h=w$ as said. According to Op-Comp, $\mathcal{A}$ has to open $2k-w$ new bins.

Stage 5. In this final stage, we aim at $n_5=2k$. We let items depart until the configuration is

$$\{3k{:}\epsilon, k-z{:}\epsilon, z{:}\epsilon, 2k-w{:}\epsilon, w{:}\epsilon, 2k{:}\frac{1}{4}+i\delta, 2k{:}\frac{1}{2}-i\delta', 2k+w{:}\frac{1}{2}+i\delta', 2k-w{:}\frac{3}{4}-i\delta, \}\ ,$$

with $14k$ bins and a total load of $4k-\frac{w}{4}+O(1)$. Finally, we release $2k$ items of size 1 and $\mathcal{A}$ has to open $2k$ new bins. In total, $\mathcal{A}$ uses $3k+(k-z)+z+(2k-w)+w+2k+2k+(2k+w)+(2k-w)+2k=16k$. The packing of $\mathcal{O}$ will be given in the full paper.

Lemma 2. *If $\mathcal{A}$ uses $[6k,8k)$ bins in Stage 0, $[2k,4k)$ bins in Stage 1, and $m\geq 2k$, then $\mathcal{A}$ uses $16k$ bins at the end while $\mathcal{O}$ uses $6k+3$ bins.*

Case 2.2.2: $6k \leq n_0 < 8k$, $2k \leq n_1 < 4k$ and $m < 2k$. We recall that in Stage 1, $15k$ items of size $\frac{1}{4}+i\delta$ are released and $\mathcal{A}$ uses $[2k,4k)$ new bins for these items.

Observation 3 *(i) At most $8k$ items of size $\frac{1}{4}+i\delta$ can be packed to the n_1 new bins. (ii) At least k of the $\{k{:}\frac{1}{4}+i\delta, k{:}\epsilon\}$ bins have load more than $\frac{1}{2}$. (iii) At least $2k$ of the $\{4k{:}(\frac{1}{2}+\epsilon)_{*\epsilon}\}$ bins are packed with at least one $(\frac{1}{2}+i\delta)$-item.*

Let z_1 and z_2 be the number of new bins that are packed one and at least two, respectively, $(\frac{1}{4}+i\delta)$-items The following observation gives a bound on z.

Observation 4 *(i) At most $9k$ items of size $\frac{1}{4}+i\delta$ can be packed in existing bins. (ii) $z_2\geq k$. (iii) $z_1\geq 3(2k-z_2)$.*

Stage 2. We target $n_2\geq z_2$. We let items depart until the configuration becomes

- $2k{:}(\frac{1}{2}+\epsilon)_{*\epsilon}$,
- $2k{:}((\frac{1}{4}+\epsilon)_{*\epsilon}, \frac{1}{4}+i\delta)$, this is possible because of Observation 3(iii),
- $x{:}(\epsilon, \frac{1}{4}+i\delta, \frac{1}{4}+i\delta)$,
- $k-x{:}((\frac{1}{4}+\epsilon)_{*\epsilon}, \frac{1}{4}+i\delta)$, this is possible because of Observation 3(ii),
- $k{:}\epsilon$,

- z_2:$(\frac{1}{4}+i\delta, \frac{1}{4}+i\delta)$, this is possible because of Observation 4(ii),
- $2(2k-z_2)$:$(\frac{1}{4}+i\delta)$, this is possible because of Observation 4(iii),

with $10k - z_2$ bins and a total load of $7k/2 + O(1)$. We then use Op-Inc with $x = \frac{1}{2}-5k\delta'$, $h = 5k - 2z_2$ and $s = z_2$. The x-items can only be packed in k of ϵ-bins and $2(2k - z_2)$ of $(\frac{1}{4}+i\delta)$-bins, i.e., $h = k + 2(2k - z_2) = 5k - 2z_2$ as said. Any new bin can pack at most two, implying that Op-Inc releases $5k = h + 2s$ items of increasing sizes from $\frac{1}{2}-5k\delta'$ to $\frac{1}{2}-\delta'$. According to Op-Inc, $\mathcal{A}$ has to open at least z_2 bins, i.e., $n_2 \geq z_2$.

Stage 3. We target $n_3 \geq z_2$. We let items depart until the configuration becomes

$$\{2k{:}(\frac{1}{2}+\epsilon)_{*\epsilon}, 2k{:}((\frac{1}{4}+\epsilon)_{*\epsilon}, \frac{1}{4}+i\delta), x{:}(\epsilon, \frac{1}{4}+i\delta, \frac{1}{4}+i\delta), k-x{:}((\frac{1}{4}+\epsilon)_{*\epsilon}, \frac{1}{4}+i\delta), k{:}\epsilon,$$
$$z_2{:}(\frac{1}{4}+i\delta, \frac{1}{4}+i\delta), 2(2k-z_2){:}\frac{1}{4}+i\delta, z_2{:}\frac{1}{2}-i\delta'\} \ ,$$

with $10k$ bins and a total load of $7k/2 + z_2/2 + O(1)$. We use Op-Comp with $s = z_2$ to release items of increasing size from $\frac{1}{2}+\delta'$. These items can only be packed in ϵ-bins (k of them) and $(\frac{1}{4}+i\delta)$-bins ($2(2k - z_2)$ of them). At any time, at most $(5k - z_2) + 1$ items are needed. According to Op-Comp, $\mathcal{A}$ has to open z_2 bins, i.e., $n_3 \geq z_2$.

Stage 4. We target $n_4 \geq (4k - z_2)$. We let items depart until the configuration becomes

$$\{4k-x{:}(\frac{1}{4}+\epsilon)_{*\epsilon}, k+x{:}(\epsilon, \frac{1}{4}+i\delta), k{:}\epsilon, 4k-z_2{:}\frac{1}{4}+i\delta, z_2{:}\frac{1}{2}-i\delta', z_2{:}\frac{1}{2}+i\delta'\} \ ,$$

with $10k+z_2+O(1)$ bins and a total load of $9k/4+3z_2/4$. We then use Op-Comp with $s = 4k-z_2$ and items of increasing size $\frac{3}{4}-i\delta$. Using similar ideas as before, $\mathcal{A}$ has to open $(4k - z_2)$ new bins.

Stage 5. We target $n_5 = 2k$. We let items depart until the configuration becomes

$$\{4k-x{:}\epsilon, k+x{:}\epsilon, k{:}\epsilon, 4k-z_2{:}\frac{1}{4}+i\delta, z_2{:}\frac{1}{2}-i\delta', z_2{:}\frac{1}{2}+i\delta', 4k-z_2{:}\frac{3}{4}-i\delta,\} \ ,$$

with $14k$ bins and a total load of $4k + O(1)$. We finally release $2k$ items of size 1 and $\mathcal{A}$ has to open $2k$ new bins. In total $\mathcal{A}$ uses $6k + 8k + 2k = 16k$ bins. The packing of $\mathcal{O}$ will be given in the full paper.

Lemma 3. *If $\mathcal{A}$ uses $[6k, 8k)$ bins in Stage 0, $[2k, 4k)$ bins in Stage 1, and $m < 2k$, then $\mathcal{A}$ uses $16k$ bins at the end while $\mathcal{O}$ uses $6k + 5$ bins.*

Theorem 5. *No online algorithm can be better than $8/3$-competitive.*

5 Conclusion

We have derived a $8/3 \ \sim 2.666$ lower bound on the competitive ratio for dynamic bin packing, improving the best known 2.5 lower bound [5]. We designed

two operations that release items of slightly increasing sizes and items with complementary sizes. These operations make a more systematic approach to release items: the type of item sizes used in a later case is a superset of those used in an earlier case. This is in contrast to the previous 2.5 lower bound in [5] in which rather different sizes are used in different cases. Furthermore, in each case, we use one or two pairs of Op-Inc and Op-Comp, which makes the structure clearer and the proof easier to understand. We also show that the new operations defined lead to a much easier proof for a 2.5 lower bound. An obvious open problem is to close the gap between the upper and lower bounds.

References

1. J. Balogh, J. Békési, G. Galambos, and G. Reinelt. Lower bound for the online bin packing problem with restricted repacking. *SIAM J. Comput.*, 38:398–410, 2008.
2. A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. In J. I. Munro, editor, *SODA*, pages 224–233. SIAM, 2004.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
4. J. W.-T. Chan, T. W. Lam, and P. W. H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(3):172–206, 2008.
5. J. W.-T. Chan, P. W. H. Wong, and F. C. C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis. *Algorithmica*, 53(2):172–206, 2009.
6. E. G. Coffman, Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, 1998.
7. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983.
8. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Bin packing approximation algorithms: A survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS, 1996.
9. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In A. Fiat and G. J. Woeginger, editors, *On-line Algorithms—The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 147–177. Springer, 1996.
10. L. Epstein and M. Levy. Dynamic multi-dimensional bin packing. *Journal of Discrete Algorithms*, 8:356–372, 2010.
11. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.
12. X. Han, C. Peng, D. Ye, D. Zhang, and Y. Lan. Dynamic bin packing with unit fraction items revisited. *Information Processing Letters*, 110:1049–1054, 2010.
13. Z. Ivkovic and E. L. Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Inf. Process. Lett.*, 59(4):229–232, 1996.
14. S. S. Seiden. On the online bin packing problem. *J. ACM*, 49(5):640–671, 2002.
15. A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.
16. P. W. H. Wong and F. C. C. Yung. Competitive multi-dimensional dynamic bin packing via L-shape bin packing. In *Proc. of Workshop on Approximation and Online Algorithms (WAOA)*, pages 242–254, 2009.

Online Multi-dimensional Dynamic Bin Packing of Unit-Fraction Items

Mihai Burcea*, Prudence W.H. Wong, and Fencol C.C. Yung

Department of Computer Science, University of Liverpool, UK
{m.burcea,pwong}@liverpool.ac.uk, ccyung@graduate.hku.hk

Abstract. We study the 2-D and 3-D dynamic bin packing problem, in which items arrive and depart at arbitrary times. The 1-D problem was first studied by Coffman, Garey, and Johnson motivated by the dynamic storage problem. Bar-Noy et al. have studied packing of unit fraction items (i.e., items with length $1/k$ for some integer $k \geq 1$), motivated by the window scheduling problem. In this paper, we extend the study of 2-D and 3-D dynamic bin packing problem to unit fractions items. The objective is to pack the items into unit-sized bins such that the maximum number of bins ever used over all time is minimized. We give a scheme that divides the items into classes and show that applying the First-Fit algorithm to each class is 6.7850- and 21.6108-competitive for 2-D and 3-D, respectively, unit fraction items. This is in contrast to the 7.4842 and 22.4842 competitive ratios for 2-D and 3-D, respectively, that would be obtained using only existing results for unit fraction items.

1 Introduction

Bin packing is a classical combinatorial optimization problem that has been studied since the early 70's and different variants continue to attract researchers' attention (see [7, 10, 12]). It is well known that the problem is NP-hard [14]. The problem was first studied in one dimension (1-D), and has been extended to multiple dimensions (d-D, where $d \geq 1$). In d-D packing, the bins have lengths all equal to 1, while items are of lengths in $(0, 1]$ in each dimension. The objective of the problem is to pack the items into a minimum number of unit-sized bins such that the items do not overlap and do not exceed the boundary of the bin. The items are oriented and cannot be rotated.

Extensive work (see [7, 10, 12]) has been done in the offline and online settings. In the offline setting, all the items and their sizes are known in advance. In the online setting, items arrive at unpredictable time and the size is only known when the item arrives. The performance of an online algorithm is measured using competitive analysis [3]. Consider any online algorithm $\mathcal{A}$ with an input I. Let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be c-competitive if there exists a constant b such that $\mathcal{A}(I) \leq c \cdot OPT(I) + b$ for all I.

* Supported by EPSRC Studentship.

In some real applications, item size is not represented by arbitrary real numbers in $(0, 1]$. Bar-Noy et al. [2] initiated the study of the *unit fraction bin packing problem*, a restricted version where all sizes of items are of the form $\frac{1}{k}$, for some integer k. The problem was motivated by the window scheduling problem [1, 2]. Another related problem is for *power fraction* items, where sizes are of the form $\frac{1}{2^k}$, for some integer k. Bin packing with other restricted form of item sizes includes divisible item sizes [8] (where each possible item size can be divided by the next smaller item size) and discrete item sizes [6] (where possible item sizes are $\{1/k, 2/k, \cdots, j/k\}$ for some $1 \leq j \leq k$). For d-D packing, items of restricted form have been considered, e.g., [16] considered strip packing ([19]) of items with one of the dimensions having discrete sizes and [17] considered bin packing of items where the lengths of each dimension are at most $1/m$, for some integer m. The study of these problems is motivated by applications in job scheduling. As far as we know, unit or power fraction items have not been considered in multi-dimensional packing.

Dynamic Bin Packing. Earlier work concentrated on "static" bin packing, where items do not depart. In potential applications, like warehouse storage, a more realistic setting is the dynamic model, where items arrive and depart dynamically. This natural generalization, known as dynamic bin packing problem, was introduced by Coffman, Garey, and Johnson [9]. The items arrive over time, reside for some period of time, and may depart at arbitrary times. Each item must be packed to a bin from its arrival to its departure. Again, migration to another bin is not allowed, yet rearrangement of items within a bin is allowed. The objective is to minimize the maximum number of bins used over all time. In the offline setting, the sizes, and arrival and departure times of items are known in advance, while in the online setting the sizes and arrival times of items are only known when items arrive, and the departure times are known only when items depart.

Previous Work. The dynamic bin packing problem was first studied in 1-D for general size items by Coffman, Garey and Johnson [9], showing that the First-Fit (FF) algorithm has a competitive ratio lying between 2.75 and 2.897, and a modified First-Fit algorithm is 2.788-competitive. They gave a formula of the competitive ratio of FF when the item size is at most $\frac{1}{k}$. When $k = 2$ and 3, the ratios are 1.7877 and 1.459, respectively. They also gave a lower bound of 2.388 for any deterministic online algorithm, which was improved to 2.5 [5] and then to 2.666 [21]. For unit fraction items, Chan et al. [4] obtained a competitive ratio of 2.4942, which was recently improved by Han et al. to 2.4842 [15], while the lower bound was proven to be 2.428 [4]. Multi-dimensional dynamic bin packing of general size items has been studied by Epstein and Levy [13], who showed that the competitive ratios are 8.5754, 35.346 and $2 \cdot 3.5^d$ for 2-D, 3-D and d-D, respectively. The ratios are then improved to 7.788, 22.788, and 3^d, correspondingly [20]. For 2-D and 3-D general size items, the lower bounds are 3.70301 and 4.85383 [13], respectively. In this case, the lower bounds apply even to unit fraction items.

Table 1. Competitive ratios for general size, unit fraction, and power fraction items. Results obtained in this paper are marked with "[*]".

	1-D	2-D	3-D
General size	2.788 [9]	7.788 [20]	22.788 [20]
Unit fraction	2.4842 [15]	6.7850 [*]	21.6108 [*]
Power fraction	2.4842 [15]	6.2455 [*]	20.0783 [*]

Our Contribution. In this paper, we extend the study of 2-D and 3-D online dynamic bin packing problem to unit and power fraction items. We observe that using the 1-D results on unit fraction items [15], the competitive ratio of 7.788 for 2-D [20] naturally becomes 7.4842, while the competitive ratio of 22.788 for 3-D [20] becomes 22.4842. An immediate question arising is whether we can have an even smaller competitive ratio. We answer the questions affirmatively as follows (see Table 1 for a summary).

- For 2-D, we obtain competitive ratios of 6.7850 and 6.2455 for unit and power fraction items, respectively; and
- For 3-D, we obtain competitive ratios of 21.6108 and 20.0783 for unit and power fraction items, respectively.

We adopt the typical approach of dividing items into classes and analyzing each class individually. We propose several natural classes and define different packing schemes based on the classes[1]. In particular, we show that two schemes lead to better results. We show that one scheme is better than the other for unit fraction items, and vice versa for power fraction items. Our approach gives a systematic way to explore different combinations of classes. One observation we have made is that dividing 2-D items into three classes gives comparable results but dividing into four classes would lead to much higher competitive ratios.

As an attempt to justify the approach of classifying items, we show that, when classification is not used, the performance of the family of any-fit algorithms is unbounded for 2-D general size items. This is in contrast to the case of 1-D packing, where the First-Fit algorithm (without classification) is $O(1)$-competitive [9].

2 Preliminaries

Notations and Definitions. We consider the online dynamic bin packing problem, in which 2-D and 3-D items must be packed into 2-D and 3-D unit-sized bins, respectively, without overflowing. The items arrive over time, reside for some period of time, and may depart at arbitrary times. Each item must be packed into a bin from its arrival to its departure. Migration to another bin is not allowed and the items are oriented and cannot be rotated. Yet, repacking

[1] The proposed classes are not necessarily disjoint while a packing scheme is a collection of disjoint classes that cover all types of items.

Table 2. Types of unit fraction items considered

$\mathrm{T}(1,1)$	$\mathrm{T}(1,\frac{1}{2})$	$\mathrm{T}(\frac{1}{2},1)$	$\mathrm{T}(\frac{1}{2},\frac{1}{2})$	$\mathrm{T}(1,\leq\frac{1}{3})$	$\mathrm{T}(\frac{1}{2},\leq\frac{1}{3})$	$\mathrm{T}(\leq\frac{1}{3},\leq 1)$

Table 3. The 2-D results of [20] for unit-fraction items

Scheme in [20]		
Classes	Types of items	Competitive ratios
Class A	$\mathrm{T}(\leq\frac{1}{3},\leq 1)$	3 [20]
Class B	$\mathrm{T}(\frac{1}{2},1), \mathrm{T}(\frac{1}{2},\frac{1}{2}), \mathrm{T}(\frac{1}{2},\leq\frac{1}{3})$	2 [20]
Class C	$\mathrm{T}(1,1), \mathrm{T}(1,\frac{1}{2}), \mathrm{T}(1,\leq\frac{1}{3})$	2.4842 [15]
Overall	All items	7.4842

of items within the same bin is permitted[2]. The *load* refers to the total area or volume of a set of 2-D or 3-D items, respectively. The objective of the problem is to minimize the total number of bins used over all time. For both 2-D and 3-D, we consider two types of input: unit fraction and power fraction items.

A *general size item* is an item such that the length in each dimension is in $(0, 1]$. A *unit fraction (UF) item* is an item with lengths of the form $\frac{1}{k}$, where $k \geq 1$ is an integer. A *power fraction (PF) item* has lengths of the form $\frac{1}{2^k}$, where $k \geq 0$ is an integer.

A packing is said to be *feasible* if all items do not overlap and the packing in each bin does not exceed the boundary of the bin; otherwise, the packing is said to *overflow* and is *infeasible*.

Some of the algorithms discussed in this paper repack existing items (and possibly include a new item) in a bin to check if the new item can be packed into this bin. If the repacking is infeasible, it is understood that we would restore the packing to the original configuration.

For 2-D items, we use the notation $\mathrm{T}(w, h)$ to refer to the type of items with width w and height h. We use '$*$' to mean that the length can take any value at most 1, e.g., $\mathrm{T}(*, *)$ refers to all items. The parameters w (and h) may take an expression $\leq x$ meaning that the width is at most x. For example, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{2})$ refers to the items with width $\frac{1}{2}$ and height at most $\frac{1}{2}$. In the following discussion, we divide the items into seven disjoint types as showed in Table 2.

The bin assignment algorithm that we use for all types of 2-D and 3-D unit and power fraction items is the First-Fit (FF) algorithm. When a new item arrives, if there are occupied bins in which the item can be repacked, FF assigns the new item to the bin which has been occupied for the longest time.

Remark on Existing Result on Unit Fraction Items. Using this notation, the algorithm in [20] effectively classifies unit fraction items into the classes as shown in Table 3. Items in the same class are packed separately, independent of

[2] If rearrangement within a bin is not allowed, one can show that there is no constant competitive deterministic online algorithm.

other classes. The overall competitive ratio is the sum of the competitive ratios of all classes. By the result in [15], the competitive ratio for Class C reduces from 2.788 [9] to 2.4842 [15] and the overall competitive ratio immediately reduces from 7.778 to 7.4842.

Corollary 1. *The 2-D packing algorithm in* [20] *is* 7.4842*-competitive for UF items.*

Remarks on Using Classification of Items. To motivate our usage of classification of items, let us first consider algorithms that do not use classification. In the full paper, we show that the family of any-fit algorithms is unbounded for 2-D general size items (Lemma 1). When a new item R arrives, if there are occupied bins in which R can be packed (allowing repacking for existing items), the algorithms assign R to one of these bins as follows: First-Fit (FF) assigns R to the bin which has been occupied for the longest time; Best-Fit (BF) assigns R to the heaviest loaded bin with ties broken arbitrarily; Worst-Fit (WF) assigns R to the lightest loaded bin with ties broken arbitrarily; Any-Fit (AF) assigns R to any of the bins arbitrarily.

Lemma 1. *The competitive ratio of the any-fit family of algorithms (*First-Fit*,* Best-Fit*,* Worst-Fit*, and* Any-Fit*) for the online dynamic bin packing problem of 2-D general size items with no classification of items is unbounded.*

When the items are unit fraction and no classification is used, we can show that FF is not c-competitive for any $c < 5.4375$, BF is not c-competitive for any $c < 9$, and WF is not c-competitive for any $c < 5.75$. The results hold even for power fraction items. These results are in contrast to the lower bound of 3.70301 of unit fraction items for any algorithm [13].

Repacking. To determine if an item can be packed into an existing bin, we will need some repacking. Here we make some simple observations about the load of items if repacking is not feasible. We first note the following lemma which is implied by Theorem 1.1 in [18].

Lemma 2 ([18]). *Given a bin with width u and height v, if all items have width at most $\frac{u}{2}$ and height at most v, then any set of these items with total area at most $\frac{uv}{2}$ can fit into the same bin by using Steinberg's algorithm.*

The implication of Lemma 2 is that if packing a new item of width $w \leq \frac{u}{2}$ and height h into a bin results in infeasible packing, then the total load of the existing items is at least $\frac{uv}{2} - wh$.

Lemma 3. *Consider packing of two types of items* $\mathrm{T}(\frac{1}{2}, \leq h)$ *and* $\mathrm{T}(1, *)$*, for some $0 < h < 1$. If we have an item of type* $\mathrm{T}(1, h')$ *that cannot be packed to an existing bin, then the current load of the bin is at least* $1 - \frac{h}{2} - h'$.

Proof. We first pack all items with width 1, including the new type $\mathrm{T}(1, h')$ item, one by one on top of the previous one. For the remaining space, we divide it into two equal halves each with width $\frac{1}{2}$. We then try to pack the $\mathrm{T}(\frac{1}{2}, \leq h)$

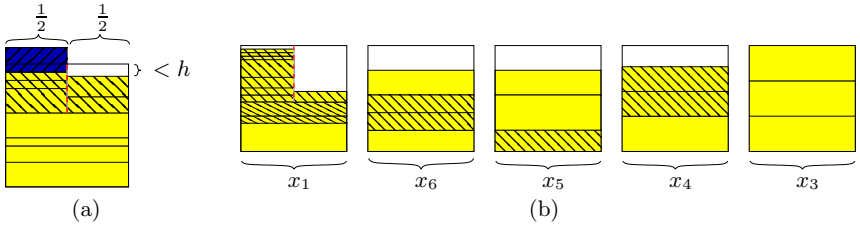


Fig. 1. (a) Infeasible repacking of existing items of types $\mathrm{T}(1, \leq\frac{1}{3})$ and $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$ and a new item of type $\mathrm{T}(1, *)$. The empty space has width $\frac{1}{2}$ and height less than h. (b) Illustration of the proof of Lemma 8. In each set of bins, the shaded items are the item types that do not appear in subsequent bins. For example, items of type $\mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$ in the first x_1 bins do not appear in the subsequent bins.

Table 4. Values of $\beta\langle x, y\rangle$ for $3 \leq x \leq 6$ and $3 \leq y \leq 6$

$\beta\langle x, y\rangle$	$y = 3$	4	5	6
$x = 3$	1	1	1	1
4	$\frac{3}{4}$	$\frac{5}{6} = \frac{1}{3} + \frac{1}{4} + \frac{1}{4}$	$\frac{5}{6}$	$\frac{11}{12} = \frac{2}{3} + \frac{1}{4}$
5	$\frac{7}{10} = \frac{1}{4} + \frac{1}{4} + \frac{1}{5}$	$\frac{47}{60} = \frac{1}{3} + \frac{1}{4} + \frac{1}{5}$	$\frac{5}{6}$	$\frac{17}{20} = \frac{1}{4} + \frac{3}{5}$
6	$\frac{7}{10}$	$\frac{23}{30} = \frac{1}{6} + \frac{3}{5}$	$\frac{49}{60} = \frac{1}{4} + \frac{1}{6} + \frac{2}{5}$	$\frac{17}{20}$

items into one compartment until it overflows, and then continue packing into the other compartment. The space left in the second compartment has a height less than h, otherwise, the overflow item can be packed there (see Figure 1(a)). As a result, the total load of items is at least $1 - \frac{h}{2}$. Since the new item has a load of h', the total load of existing items is at least $1 - \frac{h}{2} - h'$ as claimed. □

In the case of 1-D packing, Chan et al. [4] have defined the following notion. Let x and y be positive integers. Suppose that a 1-D bin is already packed with some items whose sizes are chosen from the set $\{1, \frac{1}{2}, \ldots, \frac{1}{x}\}$. They defined the notion of the minimum load of such a bin that an additional item of size $\frac{1}{y}$ cannot fit into the bin. We modify this notion such that the set in concern becomes $\{\frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{x}\}$. We define $\beta\langle x, y\rangle$ to be the minimum load of this bin containing items with length at least $\frac{1}{x}$ and at most $\frac{1}{3}$ such that an item of size $\frac{1}{y}$ cannot be packed into this bin. Precisely,

$$\beta\langle x, y\rangle = \min_{3 \leq j \leq x \text{ and } n_j \geq 0} \{\frac{n_3}{3} + \frac{n_4}{4} + \ldots + \frac{n_x}{x} \mid \frac{n_3}{3} + \frac{n_4}{4} + \ldots + \frac{n_x}{x} > 1 - \frac{1}{y}\}.$$

Table 4 shows the values of this function for $3 \leq x \leq 6$ and $3 \leq y \leq 6$.

Table 5. Classifications of 2-D unit fraction items and their competitive ratios

Classes	Types of items	Competitive ratios
Class 1	$\mathrm{T}(\leq\frac{1}{3}, \leq 1)$	2.8258
Class 2	$\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$	1.7804
Class 3	$\mathrm{T}(1,1)$, $\mathrm{T}(1, \frac{1}{2})$, $\mathrm{T}(\frac{1}{2}, 1)$, $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$	2.25
Class 4	$\mathrm{T}(1, \frac{1}{2})$, $\mathrm{T}(1, \leq\frac{1}{3})$, $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$, $\mathrm{T}(\frac{1}{2}, \leq\frac{1}{3})$	2.4593
Class 5	$\mathrm{T}(1,1)$, $\mathrm{T}(\frac{1}{2}, 1)$	1.5

3 Classification of 2-D Unit Fraction Items

Following the idea in [20], we also divide the type of items into classes. In Table 5, we list the different classes we considered in this paper. We propose two packing schemes, each of which makes use of a subset of the classes that are disjoint. The competitive ratio of a packing scheme is the sum of the competitive ratio we can achieve for each of the classes in the scheme. In this section, we focus on individual classes and in the next section, we discuss the two packing schemes. For each class, we use FF (First-Fit) to determine which bin to assign an item. For each bin, we check if the new item can be packed together with the existing items in the bin; this is done by some repacking procedures and the repacking is different for different classes.

Class 5: $\mathbf{T(1,1), T(\frac{1}{2}, 1)}$

This is a simple case and we skip the details.

Lemma 4. *FF is* 1.5*-competitive for UF items of types* $\mathrm{T}(1,1)$ *and* $\mathrm{T}(\frac{1}{2}, 1)$.

Class 3: $\mathbf{T(1,1), T(1, \frac{1}{2}), T(\frac{1}{2}, 1), T(\frac{1}{2}, \frac{1}{2})}$

We now consider Class 3 for which both the width and height are at least $\frac{1}{2}$.

Lemma 5. *FF is* 2.25*-competitive for UF items of types* $\mathrm{T}(1,1), \mathrm{T}(1, \frac{1}{2}), \mathrm{T}(\frac{1}{2}, 1)$, $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$.

Proof. Suppose the maximum load at any time is n. Then OPT uses at least n bins. Let x_1 be the last bin that FF ever packs a $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$-item, $x_1 + x_2$ for $\mathrm{T}(1, \frac{1}{2})$ and $\mathrm{T}(\frac{1}{2}, 1)$, and $x_1 + x_2 + x_3$ for $\mathrm{T}(1,1)$. When FF packs a $\mathrm{T}(\frac{1}{2}, \frac{1}{2})$-item to bin-$x_1$, all the $x_1 - 1$ before that must have a load of 1. Therefore, $(x_1 - 1) + \frac{1}{4} \leq n$. When FF packs a $\mathrm{T}(1, \frac{1}{2})$ or $\mathrm{T}(\frac{1}{2}, 1)$-item to bin-$(x_1 + x_2)$, all the bins before that must have a load of $\frac{1}{2}$. Hence, $\frac{x_1 + x_2}{2} \leq n$. When FF packs a $\mathrm{T}(1,1)$-item to bin-$(x_1 + x_2 + x_3)$, the first x_1 bins must have a load of at least $\frac{1}{4}$, the next x_2 bins must have a load of at least $\frac{1}{2}$, and the last $x_3 - 1$ bins must have a load of 1. Therefore, $\frac{x_1}{4} + \frac{x_2}{2} + (x_3 - 1) + 1 \leq n$. The maximum value of $x_1 + x_2 + x_3$ is obtained by setting $x_1 = x_2 = n$ and $x_3 = \frac{n}{4}$. Then, $x_1 + x_2 + x_3 = 2.25n \leq 2.25OPT$. □

Class 2: $\mathbf{T(1, \leq \frac{1}{3}), T(\frac{1}{2}, \leq \frac{1}{3})}$

We now consider items whose width is at least $\frac{1}{2}$ and height is at most $\frac{1}{3}$. For this class, the repack when a new item arrives is done according to the description in the proof of Lemma 3. We are going to show that FF is 1.7804-competitive for Class 2.

Suppose the maximum load at any time is n. Let x_1 be the last bin that FF ever packs a $\mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$-item. Using the analysis in [9] for 1-D items with size at most $\frac{1}{3}$, one can show that $x_1 \leq 1.4590n$.

Lemma 6 ([9]). *Suppose we are packing UF items of types* $\mathrm{T}(1, \leq \frac{1}{3}), \mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$ *and the maximum load over time is* n*. We have* $x_1 \leq 1.4590n$*, where* x_1 *is the last bin that FF ever packs a* $\mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$*-item.*

Lemma 6 implies that FF only packs items of $\mathrm{T}(1, \leq \frac{1}{3})$ in bin-y for $y > 1.459n$. The following lemma further asserts that the height of these items is at least $\frac{1}{6}$.

Lemma 7. *Suppose we are packing UF items of types* $\mathrm{T}(1, \leq \frac{1}{3}), \mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$ *and the maximum load over time is* n*. Any item that is packed by FF to bin-*y*, for* $y > 1.459n$*, must be of type* $\mathrm{T}(1, h)$*, where* $\frac{1}{6} \leq h \leq \frac{1}{3}$*.*

Proof. Suppose on the contrary that FF packs a $\mathrm{T}(1, \leq \frac{1}{7})$-item in bin-$y$ for $y > 1.459n$. This means that packing the item in any of the first $1.459n$ bins results in an infeasible packing. By Lemma 3, with $h = \frac{1}{3}$ and $h' = \frac{1}{7}$, the load of each of the first $1.459n$ bins is at least $1 - \frac{1}{6} - \frac{1}{7} = 0.69$. Then the total is at least $1.459n \times 0.69 > 1.0067n$, contradicting that the maximum load at any time is n. Therefore, the lemma follows. □

Lemma 8. *FF is* 1.7804*-competitive for UF items of types* $\mathrm{T}(1, \leq \frac{1}{3})$*,* $\mathrm{T}(\frac{1}{2}, \leq \frac{1}{3})$*.*

Proof. Figure 1(b) gives an illustration. Let $(x_1 + x_6), (x_1 + x_6 + x_5), (x_1 + x_6 + x_5 + x_4)$, and $(x_1 + x_6 + x_5 + x_4 + x_3)$ be the last bin that FF ever packs a $\mathrm{T}(1, \frac{1}{6})$-, $\mathrm{T}(1, \frac{1}{5})$-, $\mathrm{T}(1, \frac{1}{4})$-, and $\mathrm{T}(1, \frac{1}{3})$- item, respectively. When FF packs a $\mathrm{T}(1, \frac{1}{6})$-item to bin-$(x_1 + x_6)$, the load of the first x_1 is at least $1 - \frac{1}{6} - \frac{1}{6} = \frac{2}{3}$, by Lemma 3. By Lemma 7, only type $\mathrm{T}(1, k)$-item, for $\frac{1}{6} \leq k \leq \frac{1}{3}$, could be packed in the x_6 bins. These items all have width 1 and thus can be considered as 1-D case. Therefore, when we cannot pack a $\mathrm{T}(1, \frac{1}{6})$-item, the current load must be at least $\beta\langle 6, 6\rangle$. Then we have $x_1(\frac{2}{3}) + x_6\,\beta\langle 6, 6\rangle \leq n$. Similarly, we have

1. $x_1(\frac{2}{3}) + x_6\,\beta\langle 6, 6\rangle \leq n$,
2. $x_1(1 - \frac{1}{6} - \frac{1}{5}) + x_6\,\beta\langle 6, 5\rangle + x_5\,\beta\langle 5, 5\rangle \leq n$,
3. $x_1(1 - \frac{1}{6} - \frac{1}{4}) + x_6\,\beta\langle 6, 4\rangle + x_5\,\beta\langle 5, 4\rangle + x_4\,\beta\langle 4, 4\rangle \leq n$,
4. $x_1(1 - \frac{1}{6} - \frac{1}{3}) + x_6\,\beta\langle 6, 3\rangle + x_5\,\beta\langle 5, 3\rangle + x_4\,\beta\langle 4, 3\rangle + x_3\,\beta\langle 3, 3\rangle \leq n$.

We note that for each inequality, the coefficients are increasing, e.g., for (1), we have $\frac{2}{3} \leq \beta\langle 6, 6\rangle = \frac{17}{20}$, by Table 4. Therefore, the maximum value of $x_1 + x_6 + x_5 + x_4 + x_3$ is obtained by setting the maximum possible value of x_6

Table 6. Competitive ratios for 2-D unit fraction items

2DDynamicPackUFS1		
Classes	Types of items	Competitive ratios
Class 1	T($\leq\frac{1}{3}, \leq 1$)	2.8258
Class 4	T($1, \frac{1}{2}$), T($1, \leq\frac{1}{3}$), T($\frac{1}{2}, \frac{1}{2}$), T($\frac{1}{2}, \leq\frac{1}{3}$)	2.4593
Class 5	T($1, 1$), T($\frac{1}{2}, 1$)	1.5
Overall	All of the above	6.7850

satisfying (1), and then the maximum possible value of x_5 satisfying (2), and so on. Using Table 4, we compute the corresponding values as $1.4590n$, $0.0322n$, $0.0597n$, $0.0931n$ and $0.1365n$, respectively. As a result, $x_1 + x_6 + x_5 + x_4 + x_3 \leq 1.7804n \leq 1.7804OPT$. □

Class 1: T($\leq\frac{1}{3}, \leq 1$)

Items of type T($\leq\frac{1}{3}, \leq 1$) are further divided into three subtypes: T($\leq\frac{1}{3}, \leq\frac{1}{3}$), T($\leq\frac{1}{3}, \frac{1}{2}$), and T($\leq\frac{1}{3}, 1$). We describe how to repack these items and leave the analysis in the full paper.

1. When the new item is T($\leq\frac{1}{3}, \leq\frac{1}{3}$), we use Steinberg's algorithm [18] to repack the new and existing items. Note that the item width satisfies the criteria of Lemma 2.
2. When the new item is T($\leq\frac{1}{3}, \frac{1}{2}$) or T($\leq\frac{1}{3}, 1$) and the bin contains T($\leq\frac{1}{3}, \leq\frac{1}{3}$)-item, we divide the bin into two compartments, one with width $\frac{1}{3}$ and the other $\frac{2}{3}$ and both with height 1. We reserve the small compartment for the new item and try to repack the existing items in the large compartment using Steinberg's algorithm. This idea originates from [20].
3. When the new item is T($\leq\frac{1}{3}, \frac{1}{2}$) or T($\leq\frac{1}{3}, 1$) and the bin does not contain T($\leq\frac{1}{3}, \leq\frac{1}{3}$)-item, we use the repacking method as in Lemma 3 but with the width becoming the height and vice versa. Note that this implies that Lemma 8 applies for these items.

Lemma 9. *FF is* 2.8258*-competitive for UF items of type* T($\leq\frac{1}{3}, \leq 1$).

Class 4: T($1, \frac{1}{2}$), T($1, \leq\frac{1}{3}$), T($\frac{1}{2}, \frac{1}{2}$), T($\frac{1}{2}, \leq\frac{1}{3}$)

The analysis of Class 4 follows a similar framework as in Class 2. We state the result (Lemma 10) and leave the proof in the full paper.

Lemma 10. *FF is* 2.4593*-competitive for UF items of types* T($1, \frac{1}{2}$), T($1, \leq\frac{1}{3}$), T($\frac{1}{2}, \frac{1}{2}$), T($\frac{1}{2}, \leq\frac{1}{3}$).

4 Packing of 2-D Unit Fraction Items

Our algorithm, named as 2DDynamicPackUF, classifies items into classes and then pack items in each class independent of other classes. In each class, FF is

Table 7. Competitive ratios for 2-D power fraction items. Marked with [*] are the competitive ratios that are reduced as compared to unit fraction items.

2DDynamicPackPF		
Class	Types of items	Competitive ratios
Class 1	$T(\leq\frac{1}{4}, \leq 1)$	2.4995 [*]
Class 2	$T(1, \leq\frac{1}{4})$, $T(\frac{1}{2}, \leq\frac{1}{4})$	1.496025 [*]
Class 3	$T(1,1)$, $T(1, \frac{1}{2})$, $T(\frac{1}{2}, 1)$, $T(\frac{1}{2}, \frac{1}{2})$	2.25
Overall	All items	6.2455

used to pack the items as described in Section 3. In this section, we present two schemes and show their competitive ratios.

Table 6 shows the classification and associated competitive ratios for 2D-DynamicPackUFS1. This scheme contains Classes 1, 4, and 5, covering all items.

Theorem 1. 2DDynamicPackUFS1 *is* 6.7850*-competitive for* 2*-D UF items.*

Scheme 2DDynamicPackUFS2 has a higher competitive ratio than Scheme 2DDynamicPackUFS1, nevertheless, Scheme 2DDynamicPackUFS2 has a smaller competitive ratio for power fraction items to be discussed in the next section. 2DDynamicPackUFS2 contains Classes 1, 2, and 3, covering all items.

Lemma 11. 2DDynamicPackUFS2 *is* 6.8561*-competitive for* 2*-D UF items.*

5 Adaptations to Other Scenarios

In this section we extend our results to other scenarios.

2-D Power Fraction Items. Table 7 shows a scheme based on 2DDynamic-PackUFS2 for unit fraction items and the competitive ratio is reduced to 6.2455.

Theorem 2. 2DDynamicPackPF *is* 6.2455*-competitive for* 2*-D PF items.*

3-D Unit and Power Fraction Items. The algorithm in [20] effectively classifies the unit fraction items as shown in Table 8(a). The overall competitive ratio reduces from 22.788 to 21.6108. For power fraction items we slightly modify the classification for 3-D items, such that boundary values of $\frac{1}{3}$ are replaced by $\frac{1}{4}$. Table 8(b) details this classification. The overall competitive ratio reduces to 20.0783. We state the following theorem and leave the proof in the full paper.

Theorem 3. (1) *Algorithm* 3DDynamicPackUF *is* 21.6108*-competitive for UF items and* (2) *algorithm* 3DDynamicPackPF *is* 20.0783*-competitive for PF items.*

Table 8. (a) Competitive ratios for 3-D UF items. [*] This result uses Theorem 1. [**] This result uses Lemma 9. (b) Competitive ratios for 3-D PF items. [*] This result uses Theorem 2. [**] This result uses the competitive ratio of Class 1 2-D PF items.

(a)

3DDynamicPackUF [20]		
Classes	Types of items	Competitive ratios
Class 1	T($> \frac{1}{2}, *, *$)	6.7850 [*]
Class 2	T($\leq \frac{1}{2}, > \frac{1}{2}, *$)	4.8258 [**]
Class 3	T($\leq \frac{1}{2}, (\frac{1}{3}, \frac{1}{2}], *$)	4
Class 4	T($\leq \frac{1}{2}, \leq \frac{1}{3}, *$)	6
Overall	All items	21.6108

(b)

3DDynamicPackPF		
Classes	Types of items	Competitive ratios
Class 1	T($> \frac{1}{2}, *, *$)	6.2455 [*]
Class 2	T($\leq \frac{1}{2}, > \frac{1}{2}, *$)	4.4995 [**]
Class 3	T($\leq \frac{1}{2}, (\frac{1}{4}, \frac{1}{2}], *$)	4
Class 4	T($\leq \frac{1}{2}, \leq \frac{1}{4}, *$)	5.334
Overall	All items	20.0783

6 Conclusion

We have extended the study of 2-D and 3-D dynamic bin packing problem to unit and power fraction items. We have improved the competitive ratios that would be obtained using only existing results for unit fraction items from 7.4842 to 6.7850 for 2-D, and from 22.4842 to 21.6108 for 3-D. For power fraction items, the competitive ratios are further reduced to 6.2455 and 20.0783 for 2-D and 3-D, respectively. Our approach is to divide items into classes and analyzing each class individually. We have proposed several classes and defined different packing schemes based on the classes. This approach gives a systematic way to explore different combinations of classes.

An open problem is to further improve the competitive ratios for various types of items. The gap between the upper and lower bounds could also be reduced by improving the lower bounds. Another problem is to consider multi-dimensional bin packing. For d-dimensional static and dynamic bin packing, for $d \geq 2$, the competitive ratio grows exponentially with d. Yet there is no matching lower bound that also grows exponentially with d. It is believed that this is the case [11] and any such lower bound would be of great interest.

Another direction is to consider the packing of unit fraction and power fraction squares, where all sides of an item are the same length. We note that the competitive ratio for the packing of 2-D unit fraction square items would reduce to 3.9654 compared to the competitive ratio of 2-D general size square items of 4.2154 [13]. For 3-D unit fraction squares, this would reduce to 5.24537 compared to 5.37037 for 3-D general size squares [13].

References

1. A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM J. Comput.*, 32:1091–1113, April 2003.

2. A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. *ACM Trans. Algorithms*, 3, August 2007.
3. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.
4. J. W.-T. Chan, T.-W. Lam, and P. W. H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(3):521 – 529, 2008.
5. J. W.-T. Chan, P. W. H. Wong, and F. C. C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis. *Algorithmica*, 53:172–206, February 2009.
6. E. G. Coffman Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, P. W. Shor, R. R. Weber, and M. Yannakakis. Bin packing with discrete item sizes, Part I: Perfect packing theorems and the average case behavior of optimal packings. *SIAM J. Discrete Math.*, 13:38–402, 2000.
7. E. G. Coffman Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, 1998.
8. E. G. Coffman Jr., M. R. Garey, and D. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3:405–428, 1987.
9. E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983.
10. E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publishing, 1996.
11. D. Coppersmith and P. Raghavan. Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Operations Research Letters*, 8(1):17–20, 1989.
12. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In A. Fiat and G. J. Woeginger, editors, *On-line Algorithms–The State of the Art*, pages 147–177. Springer, 1996.
13. L. Epstein and M. Levy. Dynamic multi-dimensional bin packing. *J. of Discrete Algorithms*, 8:356–372, December 2010.
14. M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
15. X. Han, C. Peng, D. Ye, D. Zhang, and Y. Lan. Dynamic bin packing with unit fraction items revisited. *Inf. Process. Lett.*, 110:1049–1054, November 2010.
16. K. Jansen and R. Thöle. Approximation algorithms for scheduling parallel jobs: Breaking the approximation ratio of 2. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008*, volume 5125 of *LNCS*, pages 234–245. Springer Berlin Heidelberg, 2008.
17. F. Miyazawa and Y. Wakabayashi. Two- and three-dimensional parametric packing. *Computers & Operations Research*, 34(9):2589 – 2603, 2007.
18. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM J. Comput.*, 26(2):401–409, 1997.
19. R. van Stee. Combinatorial algorithms for packing and scheduling problems. Habilitation thesis, Universität Karlsruhe, June 2008. Available at `http://www.mpi-inf.mpg.de/~vanstee/habil.pdf`. Accessed November 2012.
20. P. W. H. Wong and F. C. C. Yung. Competitive multi-dimensional dynamic bin packing via L-shape bin packing. In E. Bampis and K. Jansen, editors, *WAOA 2009*, volume 5893 of *LNCS*, pages 242–254. Springer Berlin Heidelberg, 2010.
21. P. W. H. Wong, F. C. C. Yung, and M. Burcea. An 8/3 lower bound for online dynamic bin packing. In K.-M. Chao, T.-S. Hsu, and D.-T. Lee, editors, *ISAAC 2012*, volume 7676 of *LNCS*, pages 44–53. Springer Berlin Heidelberg, 2012.

Scheduling for Electricity Cost in Smart Grid

Mihai Burcea[1], Wing-Kai Hon[2], Hsiang-Hsuan Liu[2],
Prudence W.H. Wong[1], and David K. Y. Yau[3]

[1] Department of Computer Science, University of Liverpool, UK
{m.burcea,pwong}@liverpool.ac.uk
[2] Department of Computer Science, National Tsing Hua University, Taiwan
{wkhon,hhliu}@cs.nthu.edu.tw
[3] Department of Computer Science, Purdue University, US.
yau@cs.purdue.edu

Abstract. We study an offline scheduling problem arising in demand response management in smart grid. Consumers send in power requests with a flexible set of timeslots during which their requests can be served. For example, a consumer may request the dishwasher to operate for 30 minutes during the periods 8am to 11am or 2pm to 4pm. The grid controller, upon receiving power requests, schedules each request within the specified duration. The electricity cost is measured by a convex function of the load in each timeslot. The objective of the problem is to schedule all requests with the minimum total electricity cost. As a first attempt, we consider a special case in which the power requirement and the duration a request needs service are both unit-size. For this problem, we present a polynomial time offline optimal algorithm and show that the time complexity can be further improved if the given set of timeslots is a contiguous interval.

1 Introduction

We study an offline scheduling problem arising in "demand response management" in smart grid [6, 7, 15]. The electrical smart grid is one of the major challenges in the 21st century [5, 22, 23]. The smart grid uses information and communication technologies in an automated fashion to improve the efficiency and reliability of production and distribution of electricity. Peak demand hours happen only for a short duration, yet makes existing electrical grid less efficient. It has been noted in [4] that in the US power grid, 10% of all generation assets and 25% of distribution infrastructure are required for less than 400 hours per year, roughly 5% of the time [23]. *Demand response management* attempts to overcome this problem by shifting users' demand to off-peak hours in order to reduce peak load [3, 9, 14, 17, 18, 20]. This is enabled technologically by the advances in smart meters [10] and integrated communication. Research

initiatives in the area include GridWise [8], the SeeLoad^{TM} system [13], EnviroGrid^{TM} [19], peak demand [21], etc.

The smart grid operator and consumers communicate through smart metering devices. We assume that time is divided into integral timeslots. A consumer sends in a power request with the power requirement, required duration of service, and the time intervals that this request can be served (giving some flexibility). For example, a consumer may want the dishwasher to operate for 30 minutes during the periods from 8am to 11am or 2pm to 4pm. The grid operator upon receiving all requests has to schedule them in their respective time intervals using the minimum energy cost. The *load* of the grid at each timeslot is the sum of the power requirements of all requests allocated to that timeslot. The *energy cost* is modeled by a convex function on the load. As a first attempt to the problem, we consider in this paper the case that the power requirement and the duration of service requested are both unit-size, a request can specify several intervals during which the request can be served, and the power cost function is any convex function.

Previous work. Koutsopoulos and Tassiulas [9] has formulated a similar problem to our problem where the cost function is piecewise linear and presented a fractional solution. They also consider online algorithms as well. Salinas et al. [20] considered a multi-objective problem to minimize energy consumption cost and maximize some utility. A closely related problem is to manage the load by changing the price of electricity over time, which has been considered in a game theoretic manner [3, 17, 18]. Heuristics have also been developed for demand side management [14]. Other aspects of smart grid have also been considered, e.g., communication [4, 11, 12], security [16]. Reviews of smart grid can be found in [6, 7, 15].

The combinatorial problem we defined in this paper has analogy to the traditional load balancing problem [2] in which the machines are like our timeslots and the jobs are like our power requests. The main difference is that the aim of load balancing is usually to minimize the maximum load of the machines. Another related problem is deadline scheduling with speed scaling [1, 24] in which the cost function is also a convex function, nevertheless a job can be served using varying speed of the processor.

Our contributions. In this paper we study an optimization problem in demand response management in which requests have unit power requirement, unit duration, arbitrary timeslots that the jobs can be served, and the cost function is a general convex function. We propose a polynomial time offline algorithm and show that it is optimal. We show that

the time complexity of the algorithm is $\mathcal{O}(|\mathcal{J}|n(|\mathcal{J}|+n))$, where $\mathcal{J}$ is the set of jobs and n is the number of timeslots. We further show that if the feasible timeslots for each job to be served forms a contiguous interval, we can improve the time complexity to $\mathcal{O}(|\mathcal{J}|n(\log|\mathcal{J}|+\log n))$ using dynamic range minimum query data structure.

Technically speaking, we use a notion of "feasible graph" to represent alernative assignments. After scheduling a job, we can look for improvement via this feasible graph. We show that we can maintain optimality each time a job is scheduled. For the analysis, we compare our schedule with an optimal schedule via the notion of "agreement graph", which captures the difference of our schedule and an optimal schedule. We then show that we can transform our schedule stepwise to improve the agreement with the optimal schedule, without increasing the cost, thus proving the optimality of our algorithm.

2 Preliminaries

We consider an offline scheduling problem where the input consists of a set of unit-sized jobs $\mathcal{J}$. The time is divided into integral timeslots $T = \{1, 2, 3, \ldots, n\}$ and each job $J_i \in \mathcal{J}$ is associated with a set of feasible timeslots $I_i \subseteq T$, in which it can be scheduled. In this model, each job J_i must be assigned to exactly one feasible timeslot from I_i. The *load* $\ell(t)$ of a timeslot t represents the total number of jobs assigned to the timeslot. We consider a general convex cost function f that measures the cost used in each timeslot t based on the load at t. The total cost used is the sum of cost over time. Over all timeslots this is $\sum_{t\in T} f(\ell(t))$. The objective is to find an assignment of all jobs in $\mathcal{J}$ to feasible timeslots such that the total cost is minimized. We first describe the notions required for discussion.

Feasible graph. Given a particular job assignment A, we define a *feasible graph* G which is a directed multi-graph that shows the potential allocation of each job in alternative assignments. In G each timeslot is represented by a vertex. If job J_i is assigned to timeslot r in A, then for all $w \in I_i \backslash \{r\}$ we add a directed edge (r, w) with J_i as its label.

Legal-path in a feasible graph. A path (t, t') in a feasible graph G is a *legal-path* if and only if the load of the starting point t is at least 2 more than the load of the ending point t', i.e., $\ell(t) - \ell(t') \geq 2$.

Agreement graph. We define an *agreement graph* $G_a(A, A^*)$ which is a directed multi-graph that measures the difference between a job assignment solution A and an optimal assignment A^*. In $G_a(A, A^*)$ each timeslot is represented by a vertex. For each job J_i such that J_i is assigned

to different timeslots in A and A^*, we add an arc from t to t', where t and t' are the timeslots that J_i is assigned to by A and A^*, respectively. The arc (t, t') is labelled by the tuple $(J_i, +/-/=)$. The second value in the tuple is "+" or "−" if moving job J_i from timeslot t to timeslot t' causes the total cost of assignment A to increase or decrease, respectively. The value is "=" if moving the job does not cause any change in the total cost of assignment A.

Observation 1. *By moving J_i from t_1 to t_2 the overall energy cost* **(i)** *decreases if $\ell(t_1) > \ell(t_2) + 1$,* **(ii)** *remains the same if $\ell(t_1) = \ell(t_2) + 1$, and* **(iii)** *increases if $\ell(t_1) < \ell(t_2) + 1$.*

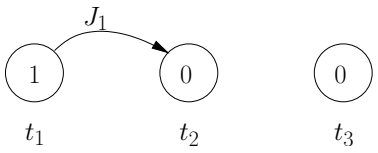
3 The Optimal Offline Algorithm

Description of the Algorithm. We propose an optimal offline algorithm that minimizes the total cost. The algorithm arranges the jobs in $\mathcal{J}$ in arbitrary order, and runs in steps. At Step i, we assign job J_i to a feasible timeslot with minimum load, breaking ties arbitrarily. Suppose job J_i is assigned to timeslot t. We update the feasible graph G to reflect this assignment in the following way. If applicable, we add arcs from t labelled by J_i to any other feasible timeslots (vertices) of J_i. Finally, the algorithm checks if there exists any directed legal-path in G from t to any other vertex t'. Recall that in this legal-path the difference between the load of the vertices t and t' is greater than 1, i.e., $\ell(t) - \ell(t') > 1$. If such a legal-path exists, the algorithm executes a *shift* whereby for each job corresponding to an arc on this path, it is moved from the original assigned timeslot to the timeslot determined by the corresponding arc. More precisely, if the path contains an arc (r, w) with J as its label, then job J is moved from r to w. At the end, the algorithm updates the feasible graph G to reflect this shift. Figure 1 shows an example of the algorithm.

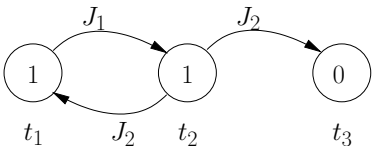
To ease the discussion, in the remainder of the paper, we use $\ell'_i(t)$ to represent the load of timeslot t after adding J_i (but before the shift), $\ell_i(t)$ to represent the load of timeslot t at the end of Step i, and $\ell'_i(s, t)$ and $\ell_i(s, t)$ to represent $\ell'_i(s) - \ell'_i(t)$ and $\ell_i(s) - \ell_i(t)$, respectively.

4 Correctness

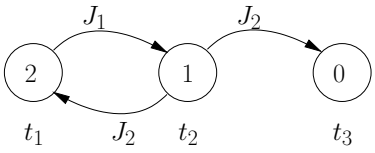
In this section we prove that our algorithm is optimal. We show in Lemma 3 that there is no legal-path after shifting in the algorithm and in Lemma 6 that this implies that the schedule is optimal.

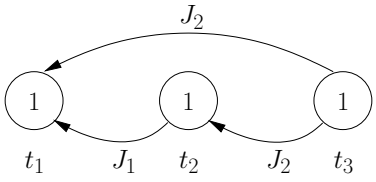


(a) The feasible graph G after adding job J_1 to t_1.



(b) The feasible graph G after adding job J_2 to t_2.





(c) Left: The feasible graph immediately after J_3 is added to slot t_1. The path (t_1, t_3) is a legal-path and we shift by moving J_1 to t_2 and J_2 to t_3. Right: The feasible graph after the shift, with no more legal-paths.

Fig. 1: Let $\mathcal{J} = \{J_1, J_2, J_3\}$, $T = \{t_1, t_2, t_3\}$, $I_1 = \{t_1, t_2\}$, $I_2 = \{t_1, t_2, t_3\}$, and $I_3 = \{t_1\}$. The number inside the vertices denotes their load. Suppose the algorithm schedules the jobs in order of their indices. (a) and (b) Jobs J_1 and J_2 are arbitrarily assigned their feasible minimum load slots. (c) A legal-path and the corresponding shift after assigning J_3.

Lemma 1. *Suppose that before adding job J_i to timeslot r the feasible graph G has no legal-path. If there is any legal-path after adding J_i, there is at least one legal-path starting from r.*

Proof. Assume that there is a legal-path (s, t) after assigning J_i to timeslot r, so that $\ell'_i(s, t) \geq 2$. If $r = s$, we have obtained a desired legal-path. Otherwise, $r \neq s$, there are two cases:

Case 1. G contains an (s, t) path before adding J_i. Since $r \neq s$, $\ell_{i-1}(s) = \ell'_i(s)$ and $\ell_{i-1}(t) \leq \ell'_i(t)$ (the latter inequality comes from the fact that r may be equal to t). This implies $\ell_{i-1}(s, t) \geq \ell'_i(s, t) \geq 2$, which contradicts the precondition that there is no legal-path before adding J_i. Thus, Case 1 cannot occur.

Case 2. G does not contain any (s, t) path before adding J_i. Since (s, t) becomes a legal-path after adding J_i, it must be the case that assigning J_i to timeslot r adds some new edge (r, w) (with J_i as its label) to G, which connects an existing (s, r) path and an existing (w, t) path. We know that $\ell_{i-1}(s) - \ell_{i-1}(r) \leq 1$ because there is no legal-path before adding J_i. Also, $\ell'_i(s) = \ell_{i-1}(s)$ and $\ell'_i(r) = \ell_{i-1}(r) + 1$ because the new job J_i is assigned to r, with $r \neq s$. Hence, $\ell'_i(r, t) \geq \ell'_i(s, t)$, so that the (r, t) subpath is also a legal-path. □

Lemma 2. *If before adding a job the feasible graph G does not have a legal-path, then after adding one more job there will be no legal-paths where the load of the starting point is at least* 3 *more than the load of the ending point. In other words, the load difference corresponding to any new legal-path, if it exists, is exactly* 2*.*

Proof. Suppose that before adding a job there is no legal-path in the feasible graph G. That is, $\ell_{i-1}(s,t) \leq 1$ for any path (s,t) in G with starting point s and ending point t. Now assume on the contrary that there is a legal-path (s,t) with $\ell'_i(s,t) \geq 3$. There are two cases:
Case 1. G contains an (s,t) path before adding job J_i. Adding a job at timeslot r increases by one on the load difference of any path starting from r. On the other hand, the load difference of any path ending at r is decreased by one. Recall that $\ell'_i(s,t) \geq 3$. There are three situations: **(i)** Job J_i is assigned to timeslot s, implying $\ell_{i-1}(s,t) \geq 2$; **(ii)** Job J_i is assigned to timeslot t, which implies $\ell_{i-1}(s,t) \geq 4$; **(iii)** Job J_i is not assigned to timeslot s or time t, which implies $\ell_{i-1}(s,t) \geq 3$. Each of these cases contradicts the fact that G has no legal-path before adding J_i.
Case 2. G does not contain any (s,t) path before adding job J_i. That is, adding job J_i create a new legal-path (s,t) with $\ell'_i(s,t) \geq 3$. There are two sub-cases (see Figure 2 for an illustration):
Case 2-1. Job J_i is assigned to timeslot s. Since (s,t) becomes a new legal-path after adding job J_i, it must be the case that there is some new edge (s,w) added in G that connects s with an existing (w,t) path. The edge (s,w) means that job J_i can be assigned to either timeslot s or w. Then, we have $\ell_{i-1}(s) = \ell'_i(s) - 1$, $\ell_{i-1}(t) = \ell'_i(t)$, and according to our assumption, $\ell'_i(s,t) \geq 3$. Since there is no legal-path before adding job J_i, $\ell_{i-1}(w,t) \leq 1$. Hence, $\ell_{i-1}(s) - \ell_{i-1}(w) \geq 1$, which contradicts the fact that J_i is assigned to a feasible timeslot with minimum load.
Case 2-2. The job J_i is assigned to timeslot r with $r \neq s$. Since (s,t) becomes a new legal-path after adding job J_i, it must be the case that there is some new edge (r,w) added in G that connects an existing (s,r) path with an existing (w,t) path. Because there is no legal-path before adding job J_i, $\ell_{i-1}(s,r) \leq 1$ and $\ell_{i-1}(w,t) \leq 1$. According to our assumption, $\ell'_i(s,t) \geq 3$; this implies $\ell_{i-1}(s,t) \geq 3$, so that $\ell_{i-1}(r) - \ell_{i-1}(w) \geq 1$. The latter inequality contradicts the fact that J_i is assigned to a feasible timeslot with minimum load. □

Based on Lemmas 1 and 2, we are ready to prove a key lemma for the correctness of our proposed algorithm.

Lemma 3. *Suppose that G is a feasible graph with no legal-paths. Then after adding a job and executing the corresponding shift by the algorithm, the resulting feasible graph has no legal-paths.*

Proof. Suppose that there were no legal-paths in G after Step $i-1$, but there is a new legal-path in G after assigning J_i. By Lemma 1, there must be one such legal-path (s,t) where s is the timeslot assigned to J_i, and without loss of generality, let it be the one that is selected by our algorithm to perform the corresponding shift. Let the ordering of the vertices in the path be $[s, v_1, v_2, \ldots, v_k, t]$, and P denote the set of these vertices.

We define $In(r)$ to be the set of vertices w such that a (w,r) path exists before adding J_i, and $Out(r)$ to be the set of vertices w such that an (r,w) path exists before adding J_i. We assume that $r \in In(r)$ and $r \in Out(r)$ for the ease of later discussion. Similarly, we define $In''(r)$ to be the set of vertices w such that a (w,r) path exists after shifting, and we define $Out''(r)$ analogously. Given a set R of vertices, let $IN(R) = \bigcup_{r\in R} In(r)$ and $OUT(R) = \bigcup_{r\in R} Out(r)$. The notation $IN''(R)$ and $OUT''(R)$ are defined analogously.

Briefly speaking, we upper bound the load of a vertex in $IN''(P)$, and lower bound the load of a vertex in $OUT''(P)$, as any legal-path that may exist after the shift must start from a vertex in $IN''(P)$ and end at a vertex in $OUT''(P)$. Based on the bounds, we shall argue that there are no legal-paths as the load difference of any path after the shift will be at most 1. Note that after the shift, only the load of t is increased by one, whereas the load of any other vertex remains unchanged. Now, concerning the legal-path (s,t), there are two cases:

Case 1. There was an arc from s to v_1 in the feasible graph G before adding J_i. In this case, it is easy to check that $IN''(P) \subseteq IN(P)$,[§] and $OUT''(P) \subseteq OUT(P) \cup OUT(I_i)$.[‡]

Suppose that $\ell_{i-1}(s) = x$. Then, $\ell_{i-1}(t) = x - 1$ because there is no legal-path before adding J_i but there is one after adding J_i. This implies

[§] Otherwise, let z be a vertex in $IN''(P)$ but not in $IN(P)$. Take the shortest path from z to some vertex in P after the shift. Then all the intermediate vertices of such a path are not from P. However, the jobs assigned to those intermediate vertices are unchanged, so that such a path also exists before the shift, and z is in $IN(P)$. A contradiction occurs.

[‡] Otherwise, let z be a vertex in $OUT''(P)$ but not in $OUT(P) \cup OUT(I_i)$. Take the shortest path that goes to z starting from some vertex in P after the shift. Then all the intermediate vertices of such a path are not from P. If such a path does not involve vertices from I_i, then this path must exist before the shift, so that z is in $OUT(P)$. Else, z is in $OUT(I_i)$. A contradiction occurs.

$\ell_{i-1}(v_h) \leq x$ for any $h \in [1,k]$, or there was a legal-path (v_h, t) before adding J_i. The load of any vertex in $IN(P)$ is at most x or there was a legal-path entering t before adding J_i. The load of any vertex in $OUT(P)$ is at least $x-1$ or there was a legal-path leaving s before adding J_i. For any vertex r in I_i, $\ell_{i-1}(r) \geq x$, since $s \in I_i$ has the minimum load. This implies that the load for any vertex in $OUT(I_i)$ is at least $x-1$, or there was a legal-path leaving a vertex in I_i before adding J_i. Thus, after the shift, the load of any vertex in $IN''(P)$ is at most x, and the load of any vertex in $OUT''(P)$ is at least $x-1$, so no legal-paths will exist.

Case 2. There were no arcs from s to v_1 in the feasible graph G before adding J_i. In this case, J_i must be involved in the shift, so that the jobs assigned to s after the shift will be the same as if J_i was not added. Consequently, if there is still a legal-path after the shift, the starting vertex must be from $IN''(P\backslash\{s\})$, while the ending vertex must be from $OUT''(P\backslash\{s\})$. Similar to Case 1, it is easy to check that $IN''(P\backslash\{s\}) \subseteq IN(P\backslash\{s\})$ and $OUT''(P\backslash\{s\}) \subseteq OUT(P\backslash\{s\}) \cup OUT(I_i)$. Suppose that $\ell_{i-1}(s) = x$, so that $\ell'_i(s) = x+1$. Because adding J_i creates a new legal-path (s,t), by Lemma 2, $\ell'_i(t) = \ell_{i-1}(t) = x-1$. Thus, the load of any vertex in $IN(P\backslash\{s\})$ is at most x, since there was no legal-path entering t before adding J_i. On the other hand, $\ell_{i-1}(v_1) \geq x$ otherwise job J_i would be assigned to v_1. However, $\ell_{i-1}(v_1) \leq x$ or there is a legal-path (v_1, t). Hence, $\ell_{i-1}(v_1) = x$. This implies that the load of any vertex in $OUT(P\backslash\{s\})$ is at least $x-1$, since there was no legal-path leaving v_1 before adding J_i. As for the vertices in $OUT(I_i)$, we can use a similar argument as in Case 1 to show that their load is at least $x-1$. Thus, after the shift, the load of any vertex in $IN''(P\backslash\{s\})$ is at most x, and the load of any vertex in $OUT''(P\backslash\{s\})$ is at least $x-1$, so no legal-path will exist. □

We now prove in Lemma 6 (the remaining key lemma for the correctness) that if the assignment is optimal before a certain step of the algorithm, it remains optimal after carrying out a shift in the algorithm. To prove Lemma 6, we need to prove the following two lemmas.

Lemma 4. *There exists an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic.*

Proof (Sketch). Consider an optimal assignment A^{**} such that $G_a(A, A^{**})$ contains directed cycles. We show that the assignment A^{**} can be transformed into an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic. For every cycle (s,t) such that $s = t$ in $G_a(A, A^{**})$, one can show that

the load of any vertex does not change after executing all the moves in the cycle. This implies that the total cost of A^{**} remains the same after removing all cycles from $G_a(A, A^{**})$. See a full proof in Appendix A.

Lemma 5. *Suppose A is not optimal and A^* is an optimal assignment such that $G_a(A, A^*)$ is acyclic. Then we can have a sequence of agreement graphs $G_a(A_1, A^*), G_a(A_2, A^*), \ldots, G_a(A_k, A^*)$ such that $A_1 = A$, $A_k = A^*$, and the cost is non-increasing every step.*

Proof. Consider the agreement graph $G_a(A_i, A^*)$, for $i \geq 1$, starting from $A_1 = A$. In each step, from $G_a(A_i, A^*)$ to $G_a(A_{i+1}, A^*)$, one arc is removed. In $G_a(A_i, A^*)$, for $i \geq 1$, we consider any arc labelled with either a "$-$" or an "$=$" and we execute the move corresponding to this arc. Through this move, we remove one arc, and thus we do not introduce any new arcs. However, the $+/-/=$ label of other arcs may change. If the resulting graph $G_a(A_{i+1}, A^*)$ does not contain any more "$-$" or "$=$" arcs, we stop. Otherwise, we repeat the process.

Note that the cost is non-increasing in every step. By the time we stop, if the resulting graph, say, $G_a(A_h, A^*)$, does not contain any more arcs, we have obtained the desired sequence of agreement graphs. Otherwise, we are left only with "$+$" labelled arcs in $G_a(A_h, A^*)$; however, in the following, we shall show that such a case cannot happen, thus completing the proof of the lemma.

Firstly, $cost(A_h) \geq cost(A^*)$ since A^* is an optimal assignment. Next, by Lemma 4, $G_a(A_1, A^*)$ is acyclic and the resulting graph $G_a(A_h, A^*)$ by removing all "$-$" and"$=$" labelled arcs is also acyclic. Thus, in $G_a(A_h, A^*)$, there must exist at least one vertex with in-degree 0 and one vertex with out-degree 0. We look at all such (v_1, v_i) paths in $G_a(A_h, A^*)$, where v_1 has in-degree 0 and v_i has out-degree 0. For any such (v_1, v_i) path, we show that by executing all moves of the path (i) the overall cost is increasing, and (ii) the labels of all arcs not contained in the (v_1, v_i) path remain "$+$". After executing all moves of the path, all arcs of the (v_1, v_i) path are removed.

(i) Suppose the vertices of the path are $[v_1, v_2, \ldots, v_i]$ and $\ell(v_1) = x$. As all arcs in (v_1, v_i) are labelled with "$+$" (i.e., the cost is increasing), $\ell(v_j) \geq x$, for $j > 1$. By executing all moves in the path, $\ell(v_1) = x - 1$, $\ell(v_j)$ is unchanged, for $1 < j < i$, and $\ell(v_i)$ is increased by one. Thus, the overall cost is increasing.

(ii) We show that the labels of all arcs not contained in the (v_1, v_i) path remain "$+$". There may be out-going arcs from v_1 to other vertices not in the (v_1, v_i) path initially labelled by "$+$". Before executing all the

moves in the (v_1, v_i) path, the load of all other vertices is at least x as we assume $\ell(v_1) = x$. After the move, $\ell(v_1) = x - 1$ and out-going arcs from v_1 point to vertices with load at least x. Thus, an arc from v_1 to any other vertex denotes a further increase in the cost and the labels of the arcs do not change. For vertices v_j, for $1 < j < i$, the load of v_j remains unchanged and thus the labels of the arcs incoming to or outgoing from v_j remain the same. For v_i, there may be incoming arcs. Suppose $\ell(v_i) = y$ before executing all the moves in the (v_1, v_i) path. Then the load of all other vertices pointing to v_i is at most y and the arcs are labelled by "+". After executing all the moves in the (v_1, v_i) path, $\ell(v_i) = y + 1$, and thus any subsequent moves from vertices pointing to v_i cause further increases in the cost, i.e., the labels do not change.

Thus, the overall cost is increasing. We repeat this process until there are no more such (v_1, v_i) paths. We end up with $cost(A_k) > cost(A^*)$, which contradicts the fact that $cost(A_k) = cost(A^*)$ as $A_k = A^*$. Thus, the case where we are left only with "+" labelled arcs in $G_a(A_h, A^*)$ cannot happen, and the lemma follows. □

Lemma 6. *If there is no legal-path in the feasible graph G, the corresponding assignment is optimal.*

Proof. Suppose by contradiction there is no legal-path in the feasible graph G, but the corresponding assignment A is not optimal. Let A^*, $A_1 = A, A_2, \ldots A_k = A^*$ be the assignments as defined in Lemmas 4 and 5. Note that each arc in the agreement graph $G_a(A_1, A^*)$ corresponds to an arc in the feasible graph G (since G captures all possible moves). Because the sequence of agreement graphs in Lemma 5 only involves removing arcs, each arc in all of $G_a(A_i, A^*)$ corresponds to an arc in G.

Suppose $G_a(A_j, A^*)$ is the first agreement graph in which a "−" labelled arc is considered between some timeslots t_a and t_b. If there is no such arc, then A is already an optimal solution (since the sequence will be both non-increasing by Lemma 5 and non-decreasing as no "−" labelled arc is involved). Otherwise, if there is such an arc in $G_a(A_j, A^*)$, we show that there must have existed a legal-path in the feasible graph G, leading to a contradiction. We denote by $\ell(A_i, t)$ the load of timeslot t in the agreement graph $G_a(A_i, A^*)$. Suppose $\ell(A_j, t_a) = x$, then $\ell(A_j, t_b) \leq x-2$ as the overall energy cost would be decreasing by moving a job from t_a to t_b. If $\ell(A_1, t_a) = x$ and $\ell(A_1, t_b) \leq x - 2$ in the original assignment, then there is a legal-path in G, which is a contradiction. Otherwise, we claim that there are some timeslots u_{i_y} and v_{k_z} such that $\ell(A_1, u_{i_y}) \geq x$ and

$\ell(A_1, v_{k_z}) \leq x - 2$, and there is a path from u_{i_y} to v_{k_z} in G. This forms a legal-path in G, leading to a contradiction.

To prove the claim, we first consider finding u_{i_y}. We first set $i_0 = j$ and $u_{i_0} = t_a$. If $\ell(A_1, u_{i_0}) \geq x$, we are done. Else, since $\ell(A_j, u_{i_0}) = x$ and $\ell(A_1, u_{i_0}) < x$, there must be some job that is moved to u_{i_0} before A_j. Let $i_1 < i_0$ be the latest step such that a job is added to u_{i_0} and the job is moved from u_{i_1}. Note that since this move corresponds to an arc with label "=", $\ell(A_{i_1}, u_{i_1}) = x$ and $\ell(A_{i_1}, u_{i_0}) = x-1$. If $\ell(A_1, u_{i_1}) \geq x$, we are done. Otherwise, we can repeat the above argument to find u_{i_2} and so on. The process must stop at some step $i_y < i_0$ where $\ell(A_1, u_{i_y}) \geq x$. Similarly, we set $k_0 = j$ and $v_{k_0} = t_b$, so that we can find a step $k_z < k_0$ such that $\ell(A_1, v_{k_z}) \leq x - 2$. Recall that since each arc in $G_a(A_1, A^*)$ corresponds to an arc in the feasible graph G and in all subsequent agreement graphs we only remove arcs, there is a path from u_{i_y} and v_{k_z} in G. Therefore, we have found a legal-path from u_{i_y} to v_{k_z} in G. □

By Lemmas 3 and 6 we have the following theorem.

Theorem 1. *Our algorithm finds an optimal assignment.*

5 Time Complexity

The proofs of the following theorems are given in Appendix A.

Theorem 2. *We can find the optimal schedule in* $\mathcal{O}(|\mathcal{J}|n(|\mathcal{J}|+n))$ *time.*

We consider the special case where each job $J_i \in \mathcal{J}$ is associated with an interval of contiguous timeslots $I_i = [\rho_i, \delta_i]$, for positive integers $\rho_i \leq \delta_i$, and each job J_i must be assigned to exactly one feasible timeslot s_i, for $\rho_i \leq s_i \leq \delta_i$.

Theorem 3. *We can find the optimal schedule in* $\mathcal{O}(|\mathcal{J}|n(\log|\mathcal{J}|+\log n))$ *time for the special case where the feasible timeslots associated with each job are contiguous.*

References

1. S. Albers. Energy-efficient algorithms. *Communication ACM*, 53(5):86–96, 2010.
2. Y. Azar. On-line load balancing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms*, volume 1442 of *LNCS*, pages 178–195. Springer, 1998.
3. S. Caron and G. Kesidis. Incentive-based energy consumption scheduling algorithms for the smart grid. In *Proc. IEEE Smart Grid Comm.*, pages 391–396, 2010.

4. C. Chen, K. G. Nagananda, G. Xiong, S. Kishore, and L. V. Snyder. A communication-based appliance scheduling scheme for consumer-premise energy management systems. *IEEE Trans. Smart Grid*, 4(1):56–65, 2013.
5. European Commission. Europen smartgrids technology platform. `ftp://ftp.cordis.europa.eu/pub/fp7/energy/docs/smartgrids_en.pdf`, 2006.
6. K. Hamilton and N. Gulhar. Taking demand response to the next level. *Power and Energy Magazine, IEEE*, 8(3):60–65, 2010.
7. A. Ipakchi and F. Albuyeh. Grid of the future. *IEEE Power and Energy Magazine*, 7(2):52–62, 2009.
8. L. D. Kannberg, D. P. Chassin, J. G. DeSteese, S. G. Hauser, M. C. Kintner-Meyer, R. G. Pratt, L. A. Schienbein, and W. M. Warwick. GridWiseTM: The benefits of a transformed energy system. *CoRR*, nlin/0409035, Sept. 2004.
9. I. Koutsopoulos and L. Tassiulas. Control and optimization meet the smart power grid: Scheduling of power demands for optimal energy management. In *Proc. e-Energy*, pages 41–50, 2011.
10. R. Krishnan. Meters of tomorrow [in my view]. *IEEE Power and Energy Magazine*, 6(2):96–94, 2008.
11. H. Li and R. C. Qiu. Need-based communication for smart grid: When to inquire power price? *CoRR*, abs/1003.2138, 2010.
12. Z. Li and Q. Liang. Performance analysis of multiuser selection scheme in dynamic home area networks for smart grid communications. *IEEE Trans. Smart Grid*, 4(1):13–20, 2013.
13. Lockheed Martin. SEELoad^{TM}Solution. `http://www.lockheedmartin.co.uk/us/products/energy-solutions/seesuite/seeload.html`.
14. T. Logenthiran, D. Srinivasan, and T. Z. Shun. Demand side management in smart grid using heuristic optimization. *IEEE Trans. Smart Grid*, 3(3):1244–1252, 2012.
15. T. Lui, W. Stirling, and H. Marcy. Get smart. *IEEE Power and Energy Magazine*, 8(3):66–78, 2010.
16. C. Y. T. Ma, D. K. Y. Yau, and N. S. V. Rao. Scalable solutions of markov games for smart-grid infrastructure protection. *IEEE Trans. Smart Grid*, 4(1):47–55, 2013.
17. S. Maharjan, Q. Zhu, Y. Zhang, S. Gjessing, and T. Basar. Dependable demand response management in the smart grid: A stackelberg game approach. *IEEE Trans. Smart Grid*, 4(1):120–132, 2013.
18. A.-H. Mohsenian-Rad, V. Wong, J. Jatskevich, and R. Schober. Optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid. In *Innovative Smart Grid Technologies (ISGT)*, 2010.
19. REGEN Energy Inc. ENVIROGRID^{TM}SMART GRID BUNDLE. `http://www.regenenergy.com/press/announcing-the-envirogrid-smart-grid-bundle/`.
20. S. Salinas, M. Li, and P. Li. Multi-objective optimal energy consumption scheduling in smart grids. *IEEE Trans. Smart Grid*, 4(1):341–348, 2013.
21. Toronto Hydro Corporation. Peaksaver Program. `http://www.peaksaver.com/peaksaver_THESL.html`.
22. UK Department of Energy & Climate Change. Smart grid: A more energy-efficient electricity supply for the UK. `https://www.gov.uk/smart-grid-a-more-energy-efficient-electricity-supply-for-the-uk`, 2013.
23. US Department of Energy. The Smart Grid: An Introduction. `http://www.oe.energy.gov/SmartGridIntroduction.htm`, 2009.
24. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

A Proofs

A.1 Proof of Lemma 4

Proof. Consider an optimal assignment A^{**} such that $G_a(A, A^{**})$ contains directed cycles. We show that the assignment A^{**} can be transformed into an optimal assignment A^* such that $G_a(A, A^*)$ is acyclic. Recall that each timeslot is represented by a vertex in $G_a(A, A^{**})$ and an arc from vertex s to vertex t labelled by a tuple $(J_i, +/-/=)$ means that J_i is assigned to timeslot s in assignment A and timeslot t in A^{**}. For every cycle (s, t) such that $s = t$ in $G_a(A, A^{**})$, we show that the load of any vertex does not change after executing all the moves in the cycle. This implies that the total cost of A^{**} remains the same after removing all cycles from $G_a(A, A^{**})$.

We consider a cycle that contains the vertices $[s, v_1, v_2, \ldots, v_k, t]$, for $s = t$. There are arcs from s to v_1, v_1 to v_2, and so on, until the last arc from v_k to $t = s$. An arc denotes the moving of a distinct job each step. As we move one job from s to v_1, $\ell(s)$ decreases by one and $\ell(v_1)$ increases by one. However, $\ell(v_1)$ returns to the original value as we move the respective job from vertex v_1 to v_2. Thus, $\ell(v_i)$, for $1 \leq i < k$ remains unchanged. As we move the last job from vertex v_k to $t = s$, both $\ell(v_k)$ and $\ell(s)$ return to their original value. Obviously, the load of all vertices remains the same even for cycles of size 2. Thus, the cost of A^{**} remains the same after removing all cycles from $G_a(A, A^{**})$ and we denote the corresponding agreement graph by $G_a(A, A^*)$. □

A.2 Proof of Theorem 2

Proof. We add jobs one by one. Each round when we assign the job J_i to timeslot r, we add arcs (r, w) labelled by J_i for all vertices w that $w \in I_i$ in the feasible graph. By Lemma 1, there is a legal-path starting from r if there is a legal-path after assigning J_i to timeslot r. When J_i is assigned to r, we start breadth-first search at r. By Lemma 2, if there is a node w which can be reached by the search and the number of jobs assigned to w is two less than the number of jobs assigned to r, it means that there is a legal-path (r, w). Then we shift the jobs according to the (r, w) legal-path. After shifting there will be no legal-paths anymore by Lemma 3. Finally we update the edges of the vertices on the legal-path in the feasible graph.

Adding J_i to the feasible graph needs $\mathcal{O}(|I_i|)$ time. Because $|I_i|$ is at most the total number of time intervals in T, $|I_i| = \mathcal{O}(n)$ where n is

the number of timeslots. The BFS takes $\mathcal{O}(n + |\mathcal{J}|n)$ time because there are at most $|\mathcal{J}|n$ edges in the feasible graph. If a legal-path exists after adding J_i and its length is l, the shifting needs $\mathcal{O}(l)$ time, which is $\mathcal{O}(n)$ because there are at most n vertices in the legal-path. After the shift, at most n edges are updated for each shifted job in the feasible graph, taking a total of $\mathcal{O}(n^2)$ time. The total time for adding $|\mathcal{J}|$ jobs is thus bounded by $\mathcal{O}(|\mathcal{J}|n(|\mathcal{J}| + n))$. □

A.3 Proof of Theorem 3

Proof (Sketch). For the special case we use data structure techniques for the speed up. For each timeslot r, we use two balanced binary search trees to maintain the collection of feasible intervals for the jobs that are assigned to it. One of these trees keeps ρ_i for all J_is that are assigned to timeslot r, and the other keeps δ_i. When job J_i is shifted from r to w, ρ_i and δ_i will be deleted from the two binary search trees, respectively; at the same time, ρ_i and δ_i will be inserted into the two binary search trees corresponding to w, respectively. The binary search trees can report the minimum ρ_i and the maximum δ_i for all J_i assigned to r. The two numbers represent the set of timeslots z in which there is a job currently assigned to r but can be shifted to z instead. Because of the contiguous property of the feasible intervals, the set of timeslots is contiguous. We define this as the *directly reachable interval* of timeslot t_i, denoted by $[\alpha_i, \beta_i]$. The query time or updating time takes $\mathcal{O}(\log |\mathcal{J}| + \log n)$ time.

We use a simple balanced binary tree structure to support dynamic range minimum query (RMQ). For a sequence of n numbers, we store the numbers in the leaves. For each internal node, it maintains the minimum number in its subtree. The value of the numbers stored in the leaves may change, and the value stored in the internal nodes on the path from root to the changed leaf should be updated to reflect this change. The updating can be done in $\mathcal{O}(\log n)$ time. This data structure can report the minimum number in the interval $[i, j]$ in $\mathcal{O}(\log n)$ time. Using the same idea, we can have a data structure that supports dynamic range maximum query.

For each timeslot r we can know the set of timeslots w such that there exists a path (r, w) using two dynamic range minimum/maximum query data structures. We maintain a dynamic range minimum query data structure as described above whose leaves are α_i for $i \in [1, n]$, and another dynamic range maximum query data structure whose leaves are β_i. Using the data structures we can query the ending vertices (i.e., timeslots) of all the paths, with length at most 2, starting from t_i by merging the interval

$[\alpha_i, \beta_i]$ together with the intervals $[\alpha_j, \beta_j]$ for each $t_j \in [\alpha_i, \beta_i]$. We define the merged interval as $[\alpha_i^{(2)}, \beta_i^{(2)}]$ for timeslot t_i (Note that after merging, we must get a contiguous interval since each merged interval must overlap with $[\alpha_i, \beta_i]$). By repeating the process, we can get $[\alpha_i^{(n-1)}, \beta_i^{(n-1)}]$, which is the set of the ending vertices of all the paths of length at most $n-1$ that start from t_i. This set contains exactly the vertices that are reachable by t_i, and is called the *reachable interval* of t_i, which can be obtained in a total of $\mathcal{O}(n \log n)$ time.

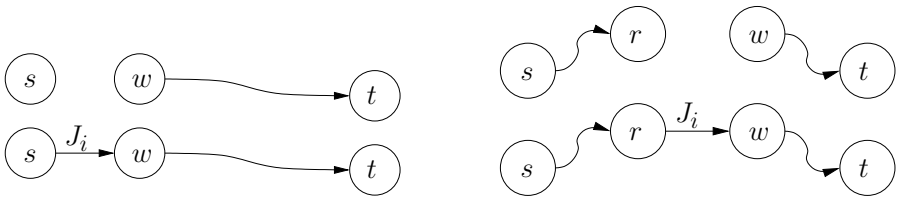
Using the dynamic range minimum query data structure we can maintain the minimum load in any interval of timeslots by storing the load of each timeslot. Each time when the shifting occurs on a legal-path (s, t), the loads of timeslot s and t change, and the load information in the data structure should be updated. The updating time and query time are both bounded by $\mathcal{O}(n(\log |\mathcal{J}| + \log n))$ time.

Now, we are ready to describe a modified algorithm for the job scheduling problem in the special case. For each job J_i, we assign it to the feasible timeslot t_i and update the binary search tree of t_i and the dynamic RMQ data structures for the directly reachable interval $[\alpha_i, \beta_i]$ and for the load. Then, we find the reachable interval $[\alpha_i^{(n-1)}, \beta_i^{(n-1)}]$ of t_i. Within this interval, we find the timeslot t with minimum load using the dynamic RMQ data structure for loads. In case the load of t is 2 fewer than the load of t_i, we can reconstruct one of the legal-paths (t_i, t) by using the dynamic RMQ data structure as follows: Suppose that $t > t_i$. Starting at t_i, we check if $t \leq \beta_i$. If so, we use the binary search tree for J_i to choose a job which can be shifted to timeslot t; else, we choose a job which can be shifted to β_i, and repeat the process to either obtain another job chosen from β_i that can be shifted to t (if $t \leq \beta_i^{(2)}$) or $\beta_i^{(2)}$. This process establishes a sequence of job shifting along a path from t_i to t. Then, we perform the shift accordingly, and update the binary search trees of timeslot r in which r is on the (t_i, t) path, the dynamic RMQ data structures for directly reachable intervals of r, and the dynamic RMQ data structures for loads of t_i and t, to reflect this shift.

The time needed for adding J_i into the feasible graph is $\mathcal{O}(\log |\mathcal{J}| + \log n)$. The time for updating information in the dynamic RMQ data structures is $\mathcal{O}(\log n)$. Finding the reachable interval takes $\mathcal{O}(n \log n)$ time, whereas finding the minimum load t from this interval takes a further $\mathcal{O}(\log n)$ time. Reconstructing a legal-path needs $\mathcal{O}(n \log n)$ time. For shifting according to the legal-path it takes $\mathcal{O}(n \log |\mathcal{J}|)$ time to update the binary search trees and $\mathcal{O}(n \log n)$ time to update the dynamic RMQ data structures. Thus, the time for adding one job is bounded by

$\mathcal{O}(n(\log|\mathcal{J}|+\log n))$, so that adding all the $|\mathcal{J}|$ jobs takes $\mathcal{O}(|\mathcal{J}|n(\log|\mathcal{J}|+\log n))$ time. □

B Figures



(a) Case 2-1 before and after assigning job J_i to timeslot s.

(b) Case 2-2 before and after assigning job J_i to timeslot r.

Fig. 2: The two sub-cases of Case 2 in the proof of Lemma 2