Cardinality Constrained Scheduling in Online Models

Leah Epstein
Department of Mathematics, University of Haifa, Israel

Alexandra Lassota
Chair of Discrete Optimization, EPFL, Lausanne, Switzerland

Asaf Levin
Faculty of Industrial Engineering and Management, Technion, Haifa, Israel

Marten Maack
Heinz Nixdorf Institute & Department of Computer Science, Paderborn University, Germany

Lars Rohwedder
School of Business and Economics, Maastricht University, The Netherlands

Abstract

Makespan minimization on parallel identical machines is a classical and intensively studied problem in scheduling, and a classic example for online algorithm analysis with Graham's famous list scheduling algorithm dating back to the 1960s. In this problem, jobs arrive over a list and upon an arrival, the algorithm needs to assign the job to a machine. The goal is to minimize the makespan, that is, the maximum machine load. In this paper, we consider the variant with an additional cardinality constraint: The algorithm may assign at most k jobs to each machine where k is part of the input. While the offline (strongly NP-hard) variant of cardinality constrained scheduling is well understood and an EPTAS exists here, no non-trivial results are known for the online variant. We fill this gap by making a comprehensive study of various different online models. First, we show that there is a constant competitive algorithm for the problem and further, present a lower bound of 2 on the competitive ratio of any online algorithm. Motivated by the lower bound, we consider a semi-online variant where upon arrival of a job of size p, we are allowed to migrate jobs of total size at most a constant times p. This constant is called the migration factor of the algorithm. Algorithms with small migration factors are a common approach to bridge the performance of online algorithms and offline algorithms. One can obtain algorithms with a constant migration factor by rounding the size of each incoming job and then applying an ordinal algorithm to the resulting rounded instance. With this in mind, we also consider the framework of ordinal algorithms and characterize the competitive ratio that can be achieved using the aforementioned approaches. More specifically, we show that in both cases, one can get a competitive ratio that is strictly lower than 2, which is the bound from the standard online setting. On the other hand, we prove that no PTAS is possible.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Cardinality Constrained Scheduling, Makespan Minimization, Online Algorithms, Lower Bounds, Pure Online, Migration, Ordinal Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.28

Related Version *Full Version*: `https://arxiv.org/abs/2201.05113` [13]

Funding L. Epstein and A. Levin were partially supported by GIF – the German-Israeli Foundation for Scientific Research and Development (grant number I-1366-407. 6/2016 on "Polynomial Migration for Online Scheduling").
Alexandra Lassota: A. Lassota was supported by the Swiss National Science Foundation within the project *Lattice algorithms and Integer Programming* (200021_185030/1).
Asaf Levin: A. Levin was also partially supported by a grant from ISF – Israeli Science Foundation (grant number 308/18).

Marten Maack: Partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" under the project number 160364472 – SFB 901/3.

1 Introduction

Scheduling jobs on identical parallel machines is a well-studied problem. Such problems were in particular investigated extensively in online settings, where the algorithm has to make decisions before the whole instance is revealed. Graham's List Scheduling from the 1960's [21] is a textbook algorithm by now and an early example of an online algorithm (although the notion of competitive analysis was not formalized at that time). In this work, we study a generalization that considers an additional cardinality constraint on the number of jobs allowed on a machine.

The Cardinality Constrained Scheduling problem. We are given a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m identical parallel machines and a positive integer k. Each job j has a job size p_j, which is also known as the processing time of the job. A feasible solution is a non-preemptive schedule (each job has to be assigned as a whole) satisfying the condition that for each machine i, the number of jobs assigned to i is at most k. Our goal is to minimize the makespan, that is, the maximum completion time of any job. In the context of makespan minimization, one does not need to explicitly consider the time axis and instead, a non-preemptive schedule can be defined as a partition of the job set to m machines, that is, a function $\sigma : \mathcal{J} \to \mathcal{M}$. The *load* of machine i in schedule σ is the total size of jobs assigned to i, that is, $\sum_{j\in\sigma^{-1}(i)} p_j$. The objective is to minimize the maximum load of a machine. It is easy to see that given σ, one can construct a schedule with makespan equal to the maximum load. Summarizing, the goal for the cardinality constrained scheduling problem is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ such that $\max_{i\in\mathcal{M}} |\sigma^{-1}(i)| \le k$ while minimizing the makespan $C_{\max}(\sigma) = \max_{i\in\mathcal{M}} \sum_{j\in\sigma^{-1}(i)} p_j$.

The cardinality constraint arises naturally in settings where one needs to balance not only the loads of the machines, but also the number of jobs. Suppose, for example, one wants to distribute passengers to airplanes for the same trip, but different times. The passengers' luggage weight may vary and jet fuel usage is very sensitive to excess weight. Thus, the goal is to minimize the maximum loaded airplane (assuming for simplicity that this dominates the fuel cost). Extensions of the original problem to multiple dimensions are well-known and studied in both offline and online settings, see for example the vector scheduling problem in [4]. However, in contrast to this problem, the second "dimension" in our problem is a hard constraint, which makes it much more difficult to handle.

In the offline setting, the job set is given beforehand and the goal is to find a feasible solution of minimum cost. We refer to [8, 10, 9, 23, 26, 27] for previous studies of the offline setting of the problem. Since the problem is NP-hard in the strong sense, the best possible approximation result is an efficient polynomial time approximation scheme (EPTAS), that is, an algorithm that returns a feasible solution (if one exists) of cost at most $(1+\varepsilon)$ times the optimal cost and the time complexity is upper bounded by the product of a computable function in ε and a polynomial in the (binary) encoding length of the input. Such an algorithm was given in [8]. Surprisingly, there exists no previous work on the online setting of this problem.

Computational models studied in this work. In the online setting, the input is given as a sequence of jobs. After a job is released, the algorithm learns the properties of the job (that is, the job size) and decides on the assignment of this job. This assignment decision is irrevocable and the algorithm is forced to maintain the feasibility of the solution after the assignment of each job (as long as the input has a feasible solution). Once the job assignment is decided, the adversary constructing the input sequence learns the algorithm's assignment decision and chooses the size of the next job or stops the input sequence. The *competitive ratio* of the online algorithm is a valid upper bound on the ratio between the cost of the solution returned by the algorithm and the optimal cost of an offline algorithm that sees the entire input sequence in advance (and may run in exponential time).

The model of ordinal algorithms is different. Here, the algorithm needs to decide an assignment of n jobs to m machines without seeing the sizes of the jobs. The only information that the algorithm can access is how these job sizes relate to each other, that is, the jobs are given as a list sorted non-increasingly by their sizes. If the algorithm has decided upon the assignment σ, it means that the i-th largest job in the input sequence is assigned to machine $\sigma(i)$, and this applies for all i. We say that an ordinal algorithm has *rate* α if for every input that satisfies the ordinal assumptions, the cost of the solution constructed by the assignment of the algorithm is at most α times the optimal cost for the same input.

Further, we study the model of algorithms with constant migration factor (also known as *robust algorithms*) similar to the online setting. But, unlike in online algorithms, once a job j is released, it is also allowed to modify the schedule of a subset of jobs of total size at most $\beta \cdot p_j$ where β is the migration factor. We require that β is a constant. Usually, one cannot maintain an optimal solution using a robust algorithm. Thus, we use robust approximation algorithms. We will use the terms competitive ratio and approximation ratio interchangeably, since some of our models are intermediate between online algorithms and offline algorithms. We say that a polynomial time algorithm that treats the input as a sequence is a robust α-approximation algorithm if it has a constant migration factor and in every sequence of jobs, the resulting solution has cost of at most α times the optimal cost. Similarly, a robust PTAS is a family of algorithms containing a robust $(1+\varepsilon)$-competitive algorithm for all $\varepsilon > 0$. It is called robust EPTAS (or robust FPTAS respectively) if its running time is upper bounded by some computable function of (or a polynomial in) $\frac{1}{\varepsilon}$ times a polynomial in the binary encoding length of the input.

Results and outline of the paper. We present new results for all three of the models mentioned above. An overview can be found in Table 1. In the pure online case, we first prove a lower bound of 2 on the competitive ratio of any (deterministic) algorithm. A natural idea for an online algorithm is to create a balanced schedule, i.e., a schedule in which the property is maintained that any two machines receive approximately the same number of jobs. This should limit the adversary's options to exploit the cardinality constraint. However, we show that such an approach fails by establishing a lower bound of m for the competitive ratio of algorithms maintaining the property that the number of jobs placed on any two machines differs by at most $o(\log(k))$. Another simple approach is to use variants of Greedy algorithms such as the list scheduling algorithm, which always assigns the next job to the machine with the lowest load. One would need to stop considering a machine once it has received k jobs. However, this approach is also deemed to fail, since it may create a large inbalance in the number of jobs assigned to the machines. If for example one machine has only one job and all others are full (which could happen using list scheduling), then the competitive ratio can be $k-1$ (when the next $k-1$ jobs are huge compared to the previous ones).

We utilize these insights in the design of an intricate online algorithm with constant competitive ratio, namely 120. This algorithm avoids both lower bounds by allowing a certain inbalance in the number of jobs, which is then gradually reduced as more and more jobs arrive. These results are presented in Section 2. Furthermore, we give a tight $\frac{1+\sqrt{5}}{2}$-competitive online algorithm for the special case $m = k = 2$ in the full version of the paper, see [13].

Table 1 An overview of the main results of this paper. The results stated in the parentheses have been completely removed from this extended abstract and can be found in the full version, see [13].

Computational Model	Result
Online algorithms	120-competitive algorithm, lower bound of 2 Lower bound of m for balanced algorithms (Tight $\frac{1+\sqrt{5}}{2}$-competitive algorithm for $m = k = 2$)
Ordinal algorithms	Algoritm with rate $\frac{81}{41}$
Robust algorithms	Robust $((1+\epsilon) \cdot \frac{81}{41})$-approximation with $\frac{1+\epsilon}{\epsilon}$ migration factor (Lower bound of ≈ 1.05 for constant migration, $m \geq 3$, unbounded k) (Robust FPTAS for $m = 2, k > 1/\epsilon^2$, and robust EPTAS for constant k)

Next, we consider the mentioned relaxed online settings starting with ordinal algorithms in Section 3. There is a known lower bound of $\frac{3}{2}$ regarding ordinal algorithm for the makespan minimization problem [30] which applies to the CCS problem as well. We present an ordinal algorithm with rate $\frac{81}{41}$ for CCS which is based on spreading out the m largest jobs over all machines and then filling the machines gradually with a repeating overlapping pattern. This gives an improvement over the rate 2, which can be achieved using a very simple round robin strategy.

In Section 4, we turn our attention to robust algorithms. First, we show that an ordinal algorithm with rate at most α can be turned into a robust $((1+\epsilon)\alpha)$-approximation with migration factor $\frac{1+\epsilon}{\epsilon}$. Together with our ordinal algorithm, this shows a separation between the strict online setting (having a lower bound of 2) and the setting with migration. On the other hand, we present a lower bound of roughly 1.05 for the ratio of robust algorithms for CCS. Hence, we cannot hope for a PTAS with a constant migration factor. However, the lower bound only works for cases with $m \geq 3$ and k part of the input, and we are able to present a robust EPTAS or FPTAS for the case with constant k or $m = 2, k > 1/\epsilon^2$, respectively.

Finally, we also show in the full version of the paper, see [13], that the results of this paper cannot be extended to a generalization of CCS called Class Constrained Scheduling by showing a super-constant lower bound on the competitive ratio of robust algorithms for that problem.

All the results, proofs and details that were excluded in this extended abstract can be found in the full version of the paper, see [13].

Related work. The standard problem of makespan minimization on identical machines is obtained from the CCS problem by deleting the constraint saying that the number of jobs assigned to each machine is at most k. At first glance, it seems that letting k grow to infinity in CCS would lead to similar results to the ones known for makespan minimization on identical machines (without cardinality constraints). However, we show that this is not the case and the corresponding possible competitive ratios in our problem are significantly higher than the one achievable for the problem without cardinality constraints. This is the case for the study of online algorithms as well as for robust algorithms.

The possible competitive ratio of the online algorithm for makespan minimization on identical machines is approximately 1.92 [1, 19] whereas the best lower bound for that problem is 1.88 by Rudin [31]. For small constant number of machines, it is known that there are better algorithms, for example, two machines List Scheduling (LS) [21] has a competitive ratio of $\frac{3}{2}$. We establish a lower bound of $2 - \frac{1}{k}$ on the competitive ratio for CCS that shows that the possible competitive ratios for CCS are strictly higher than the ones achievable for the problem without cardinality constraints both in the regime of a small fixed number of machines and in the general case (in both of these scenarios, we establish a lower bound of 2 when k grows unboundedly). Makespan minimization was also studied in terms of ordinal algorithms [30] and it is known that there is an ordinal algorithm for this problem on identical machines with constant rate. In particular, for large numbers of machines m, there is an algorithm of rate at most $\frac{5}{3}$ and no algorithm has rate smaller than $\frac{3}{2}$. The model of robust algorithms was introduced in [32] for makespan minimization on identical machines, where it is shown that there is a robust polynomial time approximation scheme for this problem. Namely, for every $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm whose migration factor is upper bounded by some function of ε.

Cardinality Constrained Bin Packing (CCBP) [2, 3, 11, 28] is the variant of CCS where the maximum job size is at most 1, the makespan is forced to be at most 1 in every feasible solution, but the algorithm is allowed to buy machines. The goal is to minimize the number of machines bought by the algorithm. The best possible competitive ratio for CCBP is 2 with respect to the absolute competitive ratio as well as with respect to the asymptotic competitive ratio [2, 3, 5]. Regarding robust algorithms, it was shown in [15] that for every fixed value of k such that $k \geq 3$, there is no asymptotic approximation scheme for CCBP with constant migration factor. Observe the difference with our results for CCS where for fixed constant value of k, we establish the existence of an approximation scheme for CCS with constant migration factor.

Ordinal algorithms were studied for other scheduling problems as well, see e.g. [12, 22, 29, 34, 35], and robust algorithms were designed and analyzed for various scheduling problems and other packing problems (see e.g. [6, 7, 14, 15, 16, 17, 18, 20, 24, 25, 33]).

Notation. Throughout the paper, log refers to a logarithm with base 2. For a job subset J, we let $p(J) = \sum_{j \in J} p_j$. For a positive integer x, we let $[x] = \{1, 2, \ldots, x\}$. Without loss of generality, we assume $\mathcal{M} = [m]$. When we consider a specific algorithm (online, ordinal, or an algorithm with constant migration), we let ALG denote the cost of the solution constructed by the algorithm, and we let OPT denote the optimal offline cost for the same instance.

2 Pure online algorithms

In this section, we study the competitive ratios of online algorithm for CCS, starting with the lower bounds and then continuing with a constant competitive algorithm.

2.1 Lower bound

We show that when considering the online problem, there is no (deterministic) algorithm that has a competitive ratio smaller than 2.

▶ **Theorem 1.** *No online algorithm for CCS has a competitive ratio strictly smaller than* 2*, and for a fixed value of k, no algorithm has a competitive ratio strictly smaller than $2 - \frac{1}{k}$.*

Proof. We assume that $m \geq k$. The input consists of two phases. In the first phase, $m \times (k-1)$ jobs of size 1 arrive. Then the adversary examines the output of the algorithm. If the algorithm has assigned exactly $k-1$ jobs to each machine, then the input continues with one big job of size k that is the last job of the input. Otherwise the input continues with m jobs of size N where N is some very big number. In the first case, we have a ratio of $2 - 1/k$. In the second case, the competitive ratio is at least $2N/(N+k)$. ◀

We sometimes use the intuition of every machine having k slots, each of which may contain exactly one job or may be empty. Note that the ratio in the second case gets worse if a larger number of slots remain free on some machine since the total number of free slots remaining after the arrival of the first phase is m. This gives us the intuition that an algorithm should try to balance the number of jobs each machine receives. However, this possible strategy cannot guarantee a constant competitive algorithm as the next proposition shows. The proof can be found in the full version of the paper, see [13].

▶ **Proposition 2.** *Let $t \geq 1$ be an integer number that may depend on k such that $t = o(\log k)$. Let* ALG *be an algorithm that maintains the invariant that the number of jobs placed on any two machines may differ by at most t. Then the competitive ratio of* ALG *is at least m.*

Observe that obtaining an online algorithm with a competitive ratio of $\min\{m, k\}$ is trivial, as any feasible solution has a cost that is at most $\min\{m, k\}$ times the optimal cost. Thus, this is the competitive ratio of scheduling the jobs in a round-robin manner. Hence, the lower bound of Proposition 2 for this class of algorithms means that in order to establish small competitive algorithms, we need to exhibit an algorithm that does not belong to this class. In fact, in our algorithm, we have situations where there is a pair of machines with cardinalities that differ by $\Theta(\log k)$.

2.2 A competitive algorithm for CCS

Next, we present an algorithm for CCS with a constant competitive ratio. For simplicity we assume that every job's size is a power of two, that is, $p_j = 2^i$ for some (not necessarily positive) integer i. More precisely, every incoming job is rounded down accordingly. Then the resulting makespan (and the competitive ratio) will only increase by a factor of 2 when considering the correct sizes.

The general idea of the algorithm. We start by briefly discussing the main idea of the algorithm. For simplicity of presentation, assume that we know the value $p^{\infty}_{\max}$, which is the maximum size of any job by the end of the instance.

We group jobs by size. Group G_i contains the jobs j with $p_j = p^{\infty}_{\max}/2^i$ for $i = 0, \ldots, \lfloor \log k \rfloor$. Further, group G_∞ contains all smaller jobs, that is, jobs of sizes below $p^{\infty}_{\max}/2^{\lfloor \log k \rfloor + 1} \leq p^{\infty}_{\max}/k$ (recall that the jobs are rounded to powers of 2). Consider the following approach: Each group is scheduled independently using a round-robin strategy. For each group, the first job of this size is assigned to machine 1, the next one to machine 2, etc. Once every machine has one job of G_i, we continue with machine 1 again for this group. This is done for all values of i including ∞, see Figure 1 for an illustration. This method approximately balances both the loads of the machines and their cardinalities: There are still differences between machine loads due to two reasons. First, each group may have one additional job (of the group) assigned to some machines compared to the other machines (this happens when the number of jobs of the group is not divisible by m). Second, the group G_∞ may have jobs of very different sizes. All these sizes are small with respect to the maximum job size, and they are smaller by a factor of at least k (so the total size of k such jobs, which is the maximum per machine, is still at most $p^{\infty}_{\max}$). Thus, the load of every machine is at most the average load plus an additional additive error term that is at most

$$\sum_{i=0}^{\lfloor \log k \rfloor} \frac{p^{\infty}_{\max}}{2^i} + k \cdot \frac{p^{\infty}_{\max}}{2^{\log k}} \leq 3p^{\infty}_{\max}.$$

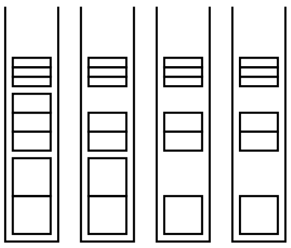

Figure 1 Example schedule of the round-robin based algorithm.

We use OPT to denote the optimal makespan for the entire input. As $p^\infty_{\max}$ forms a lower bound on OPT, and the average load is also a valid lower bound on OPT, the makespan is never larger than four times the optimal makespan. An issue arises once the cardinality on some machines arrives at k. In that case it is no longer feasible to schedule all groups independently. On the other hand, every pair of cardinalities (for two machines) differs by at most $\lfloor \log k \rfloor + 2$. Thus, if this difficulty occurs, then on each machine, there is only space for $O(\log k)$ more jobs (this is the number of remaining slots). If we assign the remaining jobs arbitrarily, the difference in loads can increase only by $O(\log k) \cdot p^\infty_{\max}$. This implies we get a $O(\log k)$-competitive algorithm assuming we know the value $p^\infty_{\max}$.

Indeed, the algorithm above is even 8-competitive (including the factor of 2 due to rounding) as long as no machine's cardinality is full, that is, as long as no machine has k jobs assigned to by the algorithm. On the other hand, we showed in Proposition 2 that no competitive algorithm can maintain a constant difference in the cardinalities of the job sets of any two machines, or even a difference of $o(\log k)$. Nevertheless, we manage to obtain a constant competitive algorithm by modifying the approach stated above further. Towards this we gradually decrease the number of groups G_i as the machines get filled more and more, so that by the time some machine is full (i.e., has been assigned k jobs) there are only a constant number of groups and, in particular, the cardinalities of the job sets assigned to any two machines differ only by a constant. We will also remove the assumption of knowing $p^\infty_{\max}$ in advance.

The algorithm. We first need to introduce some formal definitions. We think of every machine as having k slots, each of which may contain one job or it can be empty. We form k rows of these slots, where every row contains one slot of each machine. A row is *full* if each slot of the row has a job, and it is *not full* otherwise, i.e., at least one slot of the row is empty. A row is *free* if it has no job at all, i.e., all its slots are empty. In the following, let $p_{\max}$ denote the maximum job size of a job seen so far, and let $p^\infty_{\max}$ denote the maximum job size by the end of the instance. We keep track of $p_{\max}$ in an online fashion in the sense that whenever a new job is released, we check if we need to update (increase) these values.

We form a partition of the jobs (released so far) into the *groups* $G_0, \dots, G_\ell, G_\infty$, where $\ell = \lfloor 2\log(k) \rfloor$ and G_i, $i = 0, \dots, \ell$, contains all jobs j of size $p_j = p_{\max}/2^i$. Group G_∞ contains all other (smaller) jobs.

The algorithm is defined recursively and in the recursive calls, it may remove full rows from the existing schedule. The removal of the rows is only with respect to the rules of assigning future jobs to the machines (in the sense that the jobs that were assigned to slots of these rows are still assigned to the corresponding machines and are not actually removed). When rows are removed, and the number of rows is decreased, the value of k will be decreased and the value of ℓ will be updated accordingly. For a fixed value of ℓ, we can maintain the partition into groups in an online fashion. The value of ℓ may become smaller, in which case some groups are merged into the group of small jobs. Since the jobs of each group except for G_∞ share a common size, groups that are not merged remain unchanged.

To cope with the dynamic grouping where large jobs may later become small over time, we change the previous algorithm in the following way. Instead of keeping one row that is currently being filled for each group, we keep two. One of these two rows may also contain jobs from G_∞. To make this more precise, we now formally state the structural invariants of the algorithm's schedule.

The invariants of the algorithm. Consider the following structure in a schedule. For each $i = 0, \dots, \ell$ there are exactly two rows r_i, r'_i containing elements of the group G_i. Row r_i contains only elements of G_i, whereas row r'_i contains elements of G_i and of G_∞. Moreover, of each pair of rows corresponding to a common group, at least one is not full. Finally, there are $\lceil k/2 \rceil - 2(\ell+1)$ rows containing only elements from G_∞ and none of them is full. The remaining $\lfloor k/2 \rfloor$ rows are free.

The structure can be built trivially in the beginning when all rows are empty and $2(\ell+1) = 2\lfloor 2\log(k) \rfloor + 2 \le \lceil k/2 \rceil$, which holds for all $k \ge 49$. In the case where k is initially smaller, the algorithm will output an arbitrary feasible schedule, which is 48-competitive.

The definition of this structure may give the (false) impression that the algorithm tries to keep the rows not full. This is not the case. In its recursive calls, the algorithm removes certain rows from the instance when they become full and the invariants above can also be read as: When two rows r_i, r'_i (or one row belonging to G_∞) become full, they need to be removed from the instance and new rows (e.g., from the empty ones) need to be allocated to take their place. When the algorithm removes a pair of rows (two rows corresponding to a common group become full), it also decreases k by 2.

We will argue inductively that given such a structure, we can assign the remaining jobs using the recursive algorithm in a way that maintains the invariants, and so that each machine ends up with a total load (including the jobs already in the schedule) of at most

$$\frac{2}{m} \cdot p(J) + 8 \sum_{i=\log(p_{\max})}^{\log(p^\infty_{\max})} 2^i + \left(42 - \frac{1}{k-1}\right) p^\infty_{\max}. \tag{1}$$

Notice that the second term is the sum of all job sizes (that is, all powers of 2) between $p_{\max}$ and $p^\infty_{\max}$ (multiplied with 8). Since the average load $p(J)/m$ and $1/2 \cdot \sum_{i=\log(p_{\max})}^{\log(p^\infty_{\max})} 2^i \le p^\infty_{\max}$ form lower bounds on OPT, this constitutes a 60-competitive algorithm (120-competitive when taking into account the initial rounding). For $k \in \{48, 49\}$ it is trivial to see that there is an algorithm which (given the structure above) assigns all remaining jobs online while maintaining the bound (1) on the loads. This is because (1) is greater than $49 p^\infty_{\max}$ and therefore any feasible schedule satisfies the bound. This proves the base case of our inductive argument.

The algorithm for scheduling the next job while maintaining the invariants and satisfying the load bound (1). Let h denote the number of free slots in our current schedule, that is, $k \cdot m$ minus the number of jobs in the schedule. Our induction is over k and h: If $k \in \{48, 49\}$, we observed that we can guarantee the makespan bound (1); the base case $h = 0$ does not have to be considered, since this contradicts the presumed structure on the current schedule (the induction will always end in $k \in \{48, 49\}$). Assume now that $k \ge 50$ and that we have an algorithm that for all $k' \in \{k, k-1, k-2\}$ and $h' < h$ can continue the schedule with the specified structure while guaranteeing the bound (1). For all $k \geq 50$ it holds that $2(\ell+1) = 2 + 2\lfloor 2\log(k)\rfloor < \lceil k/2\rceil$. This implies that the number of rows dedicated to G_∞, $\lceil k/2\rceil - 2(\ell+1)$, is strictly positive and the number of free rows, $\lfloor k/2\rfloor$, is at least 25, which will be important for our induction step. Suppose some job j_{new} arrives.

First consider the case that $p_{j_{\text{new}}} \leq p_{\max}$ (referring to the value of $p_{\max}$ before the arrival of j_{new}) and $j_{\text{new}} \notin G_\infty$. Let G_i be the group with $j_{\text{new}} \in G_i$. We assign j_{new} to an arbitrary empty slot in r_i or r_i'. If one of the rows remains not full, we have maintained the structure and use the induction hypothesis to prove that we can construct the remaining schedule with the desired guarantee. If on the other hand this makes both rows r_i and r_i' full, we first remove the two full rows and then we are going to use the induction hypothesis as described below: We set $k' := k-2$. This reduces the index of the last group to $\ell' = \lfloor 2\log(k')\rfloor$. However, notice that $\ell \geq \ell' \geq \ell - 1$ (so it is possible that one group is merged into the group of small jobs).

Case 1: $\ell' = \ell$**.** We remove one row dedicated to G_∞ and pair it with an empty row to form new rows r_i' and r_i respectively (we recall that the numbers of such rows are positive before the removal). The number of rows dedicated to G_∞ is now $\lfloor k/2\rfloor - 2(\ell+1) - 1 = \lfloor k'/2\rfloor - 2(\ell'+1)$. Hence, the structure is repaired and we can use the induction hypothesis.

Case 2: $\ell' = \ell - 1$ **and** $i = \ell$**.** Then group G_i disappears, the number of rows dedicated to G_∞ is still $\lfloor k/2\rfloor - 2(\ell+1) = \lfloor k'/2\rfloor - 2(\ell'+1) - 1$. To repair the structure, we add one empty row to the group G_∞.

Case 3: $\ell' = \ell - 1$ **and** $i < \ell$**.** Then group G_ℓ is merged into G_∞, and it creates two new rows for G_∞ that were corresponding previously to G_ℓ. We create new rows r_i and r_i' corresponding to G_i by taking one of these rows previously corresponding to G_ℓ (the complete one if there is such a row) as r_i' and an empty one as r_i. This means the number of rows dedicated to G_∞ is $\lfloor k/2\rfloor - 2(\ell+1) + 1 = \lfloor k'/2\rfloor - 2(\ell'+1)$ and no row dedicated to G_∞ is full (because full rows of G_∞ will always be removed, and the only one that was perhaps temporarily added to the set of its rows was moved to the set of another group).

Let J_C denote the jobs in the two full rows that we have just removed. By induction hypothesis, we obtain a schedule for $J \setminus J_C$ where the load of each machine is at most

$$\frac{2}{m}\cdot p(J\setminus J_C) + 8\sum_{i=\log(p_{\max})}^{\log(p^\infty_{\max})} 2^i + \left(42 - \frac{1}{k'-1}\right)p^\infty_{\max}.$$

Notice that the total size of the two jobs of J_C on each machine is at least $p_{\max}/2^i$ and at most $2\cdot p_{\max}/2^i \leq 2/m \cdot p(J_C)$. Thus, the total load on each machine is at most

$$\frac{2}{m}p(J_C) + \frac{2}{m}p(J\setminus J_C) + 8\sum_{i=\log(p_{\max})}^{\log(p^\infty_{\max})} 2^i + \left(42 - \frac{1}{k'-1}\right)p^\infty_{\max}$$
$$\leq \frac{2}{m}p(J) + 8\sum_{i=\log(p_{\max})}^{\log(p^\infty_{\max})} 2^i + \left(42 - \frac{1}{k-1}\right)p^\infty_{\max}.$$

Now consider the case that $j_{\text{new}} \in G_\infty$. We add j_{new} to an arbitrary row dedicated to G_∞. If the row remains not full, we can directly use the induction hypothesis as the structure is maintained. Otherwise, we remove the row and set $k' = k-1$ (accordingly, $\ell' = \lfloor 2\log(k')\rfloor$). The number of rows dedicated to G_∞ has reduced to

$$\lfloor k/2\rfloor - 2(\ell+1) - 1 \leq \lfloor k'/2\rfloor - 2(\ell+1) \leq \lfloor k'/2\rfloor - 2(\ell'+1).$$

That means, we potentially have too few rows in G_∞, but not too many. On the other hand,

$$\lfloor k/2 \rfloor - 2(\ell+1) - 1 \geq \lfloor k'/2 \rfloor - 2(\ell'+2) - 1 = \lfloor k'/2 \rfloor - 2(\ell'+1) - 3.$$

So G_∞ is missing at most three rows. We fill up these missing row with empty ones (recall there are at least 25 empty rows) and we have maintained the required structure. Let again J_C denote the jobs in the full row. Using the induction hypothesis and that each job in J_C is of size less than $p_{\max}/k^2$ (by using the previous values of ℓ, this is the upper bound on the sizes of jobs of G_∞), we get a schedule with maximum load at most

$$\begin{aligned}
&\frac{2}{m}p(J \setminus J_C) + 8 \sum_{i=\log(p_{\max})}^{\log(p_{\max}^\infty)} 2^i + \left(42 - \frac{1}{k'-1}\right) p_{\max}^\infty + \frac{1}{k^2}p_{\max} \\
&\leq \frac{2}{m}p(J) + 8 \sum_{i=\log(p_{\max})}^{\log(p_{\max}^\infty)} 2^i + \left(42 - \frac{1}{k-2} + \frac{1}{k^2}\right) p_{\max}^\infty \\
&\leq \frac{2}{m}p(J) + 8 \sum_{i=\log(p_{\max})}^{\log(p_{\max}^\infty)} 2^i + \left(42 - \frac{1}{k-1}\right) p_{\max}^\infty.
\end{aligned}$$

Finally consider the case that $p_{j_{\text{new}}} > p_{\max}$. This means the new maximal job size is $p'_{\max} = p_{j_{\text{new}}}$. From the job set J^0 of the jobs in our current schedule (not including jobs of removed rows) and excluding j_{new}, we construct a new job instance $J^{0'}$ where we increase the size of the jobs in G_i, $i = 0, \ldots, \ell$, from $p_{\max}/2^i$ to $p'_{\max}/2^i$ and we consider the same schedule for $J^{0'}$, which satisfies our required structure. Let J' be the union of jobs $J^{0'}$, j_{new}, and all remaining jobs that have not arrived yet. If we can assign all remaining jobs in this bigger instance J' with some bound on the loads of the machines, then we can in particular obtain the same bound on the maximum load for our original instance J. Notice that since in the current schedule there are only two rows containing jobs of each group G_i, $i < \infty$, there can be at most $2m$ jobs in total of each such group. Thus,

$$p(J') \leq p(J) + 2m \sum_{i=0}^{\ell} \frac{p'_{\max}}{2^i} \leq p(J) + 8m \cdot \frac{p'_{\max}}{2}.$$

In the previous two cases, we have already proved that (for the given numbers h and k) we can schedule the remaining jobs of J' with a bounded makespan, since we are in the case that $p_{j_{\text{new}}} \leq p'_{\max}$. More precisely, we obtain with the previous arguments a schedule for J' (and in particular for J) with a maximum load of at most

$$\frac{2}{m}p(J') + 8 \sum_{i=\log(p'_{\max})}^{\log(p_{\max}^\infty)} 2^i + \left(42 - \frac{1}{k-1}\right) p_{\max}^\infty \leq \frac{2}{m}p(J) + 8 \sum_{i=\log(p_{\max})}^{\log(p_{\max}^\infty)} 2^i + \left(42 - \frac{1}{k-1}\right) p_{\max}^\infty,$$

where the inequality holds since $\log(p'_{\max}) \geq \log(p_{\max}) + 1$ due to the rounding. This concludes the proof.

▶ **Theorem 3.** *For cardinality constrained scheduling there is a* 120*-competitive online algorithm.*

3 Ordinal algorithms

A straight-forward approach in the ordinal setting is to assign the jobs via round-robin, i.e., the i-th largest job is assigned to machine $((i-1) \bmod m)+1$. Note that this already gives an ordinal algorithm of rate 2 and applies to both, the makespan minimization problem on identical machines as well as our problem. Thus, the goal of this section is to show that by delicately defining a different algorithm, we can get an ordinal algorithm of rate strictly smaller than 2.

Preliminaries and easy cases. We can always assume that the job sequence has $n = m \cdot k$ jobs. If it has more jobs then we can safely output that there is no feasible solution, and otherwise, we can add $n - m \cdot k$ jobs of size 0 at the end of the input sequence. These zero sized jobs do not change the optimal cost, and for every feasible solution of the original instance, there is a corresponding feasible solution of the same cost with the zero sizes jobs (by adding for each machine the number of jobs so that it will have exactly k jobs).

Furthermore, we will assume that $m \geq 2$ and $k \geq 3$. For one machine ($m = 1$), placing the k input jobs on the single given machine is a trivial ordinal algorithm with rate 1. Similarly, if $k = 1$, assigning the i-th largest of the m input jobs to machine i is again ordinal and optimal. Lastly, in the case $k = 2$, an optimal schedule can be achieved by assigning the first (largest) job to the first machine, the second to the second and so on until each machine received one job. Afterwards, we change the direction, that is, job $m + 1$ is assigned to machine m, job $m + 2$ to machine $m - 1$, and so forth.

First ideas. Observe that assigning the first m jobs to different machines is necessary in any algorithm of rate strictly smaller than 2 as the ordered input might consist of m jobs of size 1 and $m(k-1)$ jobs of size 0. On the other hand, the mentioned round-robin approach behaves badly if we have one big and many small jobs, e.g., if we have $m = k$, one job of size k, $k(k-1)$ jobs of size 1, and the remaining jobs of size 0. Then the first machine receives load $2k - 1$ in the round-robin approach while there is a trivial solution with objective value k.

Keeping these examples in mind, a first idea for an ordinal algorithm might be to spread out the first m jobs and afterward place fewer jobs on the machines that received the largest jobs. More concretely we could, for instance, place the first m jobs as described and then alternately take m and $\lceil m/2 \rceil$ jobs (from the input sequence, i.e., in non-increasing order) and place them on all and the last $\lceil m/2 \rceil$ machines, respectively. This is, in fact, the central idea for the known [30] ordinal algorithm with rate $\frac{5}{3}$ for the case without cardinality constraints. But in our case, we need another strategy after the last $\lceil m/2 \rceil$ machines each received k jobs. The most obvious idea at this point, would be to apply round robin to the remaining jobs and first $\lfloor m/2 \rfloor$ machines. However, it is relatively easy to see that we can adapt the bad example for round-robin to the resulting algorithm by simply doubling the number of machines and hence, a more sophisticated approach is needed.

Now, the first step of the algorithm presented here is again to place the i-th biggest job to machine i for $i \in [m]$. Then we use the above approach of placing jobs alternately on a smaller and a larger part of the last $\lceil m/2 \rceil$ machines and apply it repeatedly so that the machines are gradually filled up starting from the last machines. In the following, this approach is described in detail.

The algorithm for the general case. The assignment procedure works in *phases*, each of which is composed of *rounds*. Let $\xi = \lfloor \log m \rfloor + 2$ be the number of phases. A round is defined as an interval of consecutive machines $[m_\ell, m_r - 1]$, where the two indexes $m_\ell, m_r \in [m + 1]$

are called *border machines* (the value $m+1$ is to allow that an interval ends with the last machine). In a round defined by the interval $[m_\ell, m_r - 1]$, we assign the next $m_r - m_\ell$ jobs to the machines of this interval where the i-th largest job in this set of jobs that we assign in the round is assigned to machine $m_\ell + i - 1$.

Next, we define a subset of the machines that are border machines of some round of the algorithm. We set $\mu(i) = \left\lfloor \frac{m}{2^{\xi - i}} \right\rfloor + 1$ for each $i \in [\xi]$. The border machines of the rounds are always from the set $\{\mu(b) \mid b \in [\xi]\}$. Note that $\mu(1) = 1$, $\mu(2) = 2$ and $\mu(\xi) = m+1$ for any value of m. Observe that the difference between two consecutive border machines, i.e., $\mu(i+1) - \mu(i)$, grows approximately as a geometric sequence. We briefly consider some examples:

- If $m = \{2, 3\}$, we have $\xi = 3$ and the three borders are 1, 2, and 3 or 4 respectively.
- If $m \in \{4, 5, 6, 7\}$, we have $\xi = 4$, and the third border is 3 for $m = 4, 5$ and 4 for $m = 6, 7$.
- If $m = 2^q$ for some integer $q \geq 1$, we have $q+2$ phases and $\{1\} \cup \{1 + 2^i \mid i \in \{0, \ldots, q\}\}$ is the set of border machines.

We conclude that in order to define the assignment, we need to define the intervals of the rounds of every phase. Like in the last chapter, we use the intuition that each machine has k slots to be filled by jobs. Our algorithm works as follows, due to space restrictions, the proof is excluded and can be found in the full version, see [13]:

1. In the first phase, there is only one round with borders $\mu(1) = 1$ and $\mu(\xi) = m+1$.
2. In the second phase, we repeat the following until all the slots between $\mu(\xi - 1)$ and $\mu(\xi)$ are filled: One round with borders $\mu(\xi - 2)$ and $\mu(\xi)$, followed by two rounds with borders $\mu(\xi - 1)$ and $\mu(\xi)$ (or less rounds if each machine of this last interval has exactly k jobs).
3. In phase $s \in \{3, \ldots, \xi - 1\}$, there are alternating rounds with borders $\mu(\xi - s)$, $\mu(\xi - s + 2)$ and $\mu(\xi - s + 1)$, $\mu(\xi - s + 2)$ respectively. There are as many rounds as are needed to fill all the slots of the interval between $\mu(\xi - s + 1)$ and $\mu(\xi - s + 2)$.
4. In the last phase, each round has borders $\mu(1) = 1$ and $\mu(2) = 2$, so in each such round we assign one job to machine 1 (and no other jobs to other machines). The number of rounds of this phase is so that the resulting number of jobs assigned to machine 1 in all phases is exactly k.

▶ **Theorem 4.** *There is an ordinal algorithm for cardinality constrained scheduling of rate at most $\frac{81}{41}$.*

4 Algorithms with constant migration factors

In this section, we consider algorithms with constant migration factor. Due to space restrictions, most of our results in this context are not included in this extended abstract. In the following, we focus on the connection between ordinal and robust algorithms.

▶ **Theorem 5.** *Given a polynomial time algorithm* ALG *for the ordinal settings of rate at most α, there is a robust $((1+\varepsilon)\alpha)$-approximation algorithm whose migration factor is $\frac{1+\varepsilon}{\varepsilon}$.*

Proof. Upon the release of a new job j, we immediately round up its size p_j to the next integer power of $(1+\varepsilon)$. Let p'_j be the rounded size of job j. Our algorithm ignores the original sizes of jobs and simply schedules the jobs of this rounded input. Observe that every feasible solution of the original instance is also a feasible solution of the rounded instance and vice-versa. Furthermore, the cost of a solution in terms of the original instance is at most its cost in terms of the rounded instance, and this last term is again at most $(1+\varepsilon)$ times the cost of the solution with respect to the original instance. Regarding the migration factor, there may be a multiplicative increase by a factor of $1+\varepsilon$. Thus, in order to prove the claim, it suffices to present an α-approximation algorithm for the rounded instance whose migration factor is at most $\frac{1}{\varepsilon}$. Thus, in the remainder of this proof we consider the rounded instance.

Our algorithm maintains a list of the jobs, ordered non-increasingly, that were already released followed by a sequence of jobs of size 0. The (already) released jobs are sorted in a non-decreasing order of their sizes. Based on this ordered list of jobs, we assign the jobs using the ordinal algorithm. The approximation ratio of the resulting algorithm is at most α (for the rounded input) based on the assumption on the rate of the ordinal algorithm. Thus, in order to prove the claim it suffices to show that we can maintain this sorted list (and its corresponding schedule) by migrating at most one job of each size that is smaller than the size of the newly arrived job. To see that, recall that in the rounded instance, all job sizes are integer powers of $1+\varepsilon$, and so the total size of migrated jobs when j is released would be at most $p'_j \cdot \sum_{i=1}^{\infty} \frac{1}{(1+\varepsilon)^i} = p'_j \cdot \frac{1}{\varepsilon}$ as we claimed.

In the rounded instance, we let a *size class* be the set of jobs of a common size, and this appears as a consecutive sublist of jobs in the sorted list. Observe that we can modify the sorted order of jobs by changing the order of jobs of a common size class (but when reflecting this change to the schedule this may create further migration). We append the new job j as the smallest job of its size class. Hence, it will be placed at the position of the largest job of the next smallest non-empty size class. We can remove and reinsert this job treating it the same way as we did the new job. Hence, for every non-empty size class whose common size is smaller than p'_j we take its largest job j' and move it to become the smallest of its size class. As the last step of this procedure, one of the size 0 dummy jobs is removed from the list. Observe that when reflected to the schedule, the jobs which were not the largest among their size class were not migrated by this resorting of the jobs. Furthermore only one job of each such size class is migrated. The running time of this procedure is linear (for every arriving job), so the claim follows. ◀

References

1 Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

2 Luitpold Babel, Bo Chen, Hans Kellerer, and Vladimir Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004.

3 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. Online bin packing with cardinality constraints resolved. *Journal of Computer and System Sciences*, 112:34–49, 2020.

4 Nikhil Bansal, Tim Oosterwijk, Tjark Vredeveld, and Ruben van der Zwaan. Approximating vector scheduling: Almost matching upper and lower bounds. *Algorithmica*, 76(4):1077–1096, 2016.

5 József Békési, György Dósa, and Leah Epstein. Bounds for online bin packing with cardinality constraints. *Information and Computation*, 249:190–204, 2016.

6 Sebastian Berndt, Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack, and Lars Rohwedder. Online bin covering with limited migration. In *Proc. of the 27th Annual European Symposium on Algorithms, ESA 2019*, volume 144, pages 18:1–18:14, 2019.

7 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. *Mathematical Programming*, 179(1):109–155, 2020.

8 Lin Chen, Klaus Jansen, Wenchang Luo, and Guochuan Zhang. An efficient PTAS for parallel machine scheduling with capacity constraints. In *International Conference on Combinatorial Optimization and Applications*, pages 608–623. Springer, 2016.

9 Mauro Dell'Amico and Silvano Martello. Bounds for the cardinality constrained $P||C_{max}$ problem. *Journal of Scheduling*, 4(3):123–138, 2001.

10 Mauro Dell'Amico, Manuel Iori, Silvano Martello, and Michele Monaci. Lower bounds and heuristic algorithms for the k_i-partitioning problem. *European Journal of Operational Research*, 171(3):725–742, 2006.

11 L. Epstein. Online bin packing with cardinality constraints. *SIAM Journal on Discrete Mathematics*, 20(4):1015–1030, 2006.

12 Leah Epstein. A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3):269–284, 2018.

13 Leah Epstein, Alexandra Lassota, Asaf Levin, Marten Maack, and Lars Rohwedder. Cardinality constrained scheduling in online models, 2022. `arXiv:2201.05113`.

14 Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Mathematical Programming*, 119(1):33–49, 2009.

15 Leah Epstein and Asaf Levin. Robust approximation schemes for cube packing. *SIAM Journal on Optimization*, 23(2):1310–1343, 2013.

16 Leah Epstein and Asaf Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69(1):26–57, 2014.

17 Leah Epstein and Asaf Levin. Robust algorithms for total completion time. *Discrete Optimization*, 33:70–86, 2019.

18 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *Proc. of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 51:1–51:24, 2018.

19 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

20 Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine covering with bounded migration. In *Proc. of the 26th European Symposium on Algorithms, ESA 2018*, pages 32:1–32:14, 2018.

21 Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

22 Yong He and Zhiyi Tan. Ordinal on-line scheduling for maximizing the minimum machine completion time. *Journal of Combinatorial Optimization*, 6(2):199–206, 2002.

23 Yong He, Zhiyi Tan, Jing Zhu, and Enyu Yao. k-partitioning problems for maximizing the minimum load. *Computers & Mathematics with Applications*, 46(10-11):1671–1681, 2003.

24 Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration. *SIAM Journal on Discrete Mathematics*, 33(4):2062–2091, 2019.

25 Klaus Jansen, Kim-Manuel Klein, Maria Kosche, and Leon Ladewig. Online strip packing with polynomial migration. In *Proc. of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2017*, pages 13:1–13:18, 2017.

26 Yasushi Kawase, Kei Kimura, Kazuhisa Makino, and Hanna Sumita. Optimal matroid partitioning problems. *Algorithmica*, 2021. To appear.

27 Hans Kellerer and Vladimir Kotov. A 3/2-approximation algorithm for k_i-partitioning. *Operations Research Letters*, 39(5):359–362, 2011.

28 K. L. Krause, V. Y. Shen, and H. D. Schwetman. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *Journal of the ACM*, 22(4):522–550, 1975.

29 Wei-Ping Liu and Jeffrey B Sidney. Bin packing using semi-ordinal data. *Operations research letters*, 19(3):101–104, 1996.

30 Wei-Ping Liu, Jeffrey B Sidney, and André Van Vliet. Ordinal algorithms for parallel machine scheduling. *Operations Research Letters*, 18(5):223–232, 1996.

31 John F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.

32 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

33 Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Mathematics of Operations Research*, 41(3):991–1021, 2016.

34 Zhiyi Tan and Yong He. Semi-on-line scheduling with ordinal data on two uniform machines. *Operations Research Letters*, 28(5):221–231, 2001.

35 Zhiyi Tan, Yong He, and Leah Epstein. Optimal on-line algorithms for the uniform machine scheduling problem with ordinal data. *Information and Computation*, 196(1):57–70, 2005.