Non-clairvoyant Batch Sets Scheduling: Fairness Is Fair Enough

Julien Robert[1] and Nicolas Schabanel[2]

[1] École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France
`http://perso.ens-lyon.fr/julien.robert`
[2] CNRS Centro de Modelamiento Matemático, Blanco Encalada 2120 Piso 7, Santiago de Chile
`http://www.cmm.uchile.fr/~schabanel`

Abstract. In real systems, such as operating systems, the scheduler is often unaware of the remaining work in each job or of the ability of the job to take advantage of more resources. In this paper, we adopt the setting for non-clairvoyance of [3,2]. Based on the particular case of malleable jobs, it is generally assumed in the literature that "**Equi** *never starves a job since it allocates to every job the same amount of processing power*". We provide an analysis of the competitiveness of **Equi** for the makespan objective which shows that under this more general setting this statement is at the same time true and false: false, because, some jobs may be stretched by a factor as large as, but no more than, $\frac{\ln n}{\ln \ln n}$ with respect to the optimal, where n is the size of the largest set; true, because no algorithm can achieve a better competitive ratio up to a constant factor.

In this paper, we extend the results in [2,11] to the batch scheduling of sets of jobs that go through *arbitrary* phases: user request all together at time 0, for the execution of a set of jobs and is served when the last job completes. We prove that the algorithm **Equi**$\circ$**Equi** is $(2 + \sqrt{3} + o(1))\frac{\ln n}{\ln \ln n}$-competitive, where n is the maximum size of a set, which is optimal up to a constant factor. We provide experimental evidences that this algorithm may have the same asymptotic competitive ratio $\Theta(\frac{\ln n}{\ln \ln n})$ (independent of the number of requests) for the flowtime objective when requests have release dates, if it is given sufficiently large extra processing power with respect to the optimum.

Keywords: Online scheduling, Non-clairvoyant algorithm, Batch scheduling, Fairness, Equi-partition, Makespan and Overall Set Completion Time minimization.

1 Introduction

Scheduling questions arise naturally in many different areas among which operating system design, compiling, memory management, communication network, parallel machines, clusters management,... In real systems, the characteristics of the jobs to schedule (such as release time, processing time,...) are often unknown and/or unpredictable beforehand. In particular, the scheduler, such as in an operating system, is typically unaware of the remaining work in each job or of the ability of the job to take advantage of more resources. Such systems are referred as *non-clairvoyant.*

Several settings have been proposed to model non-clairvoyance, in which jobs are fully parallelizable but their amount of work is unknown before their completion [9,6,7,5,8,1]. In this paper, we consider the very general setting for non-clairvoyance proposed by Edmonds [3,2]. Jobs go through a sequence of different phases, each consisting of a certain quantity of work with a speed-up function that quantifies how it takes advantage of the number of processors it receives. For example, during a fully parallel phase, the speed-up function increases linearly with the number of processors received.

Surprisingly, in this setting, even if the scheduler is unaware of the characteristics of the phases, some policies achieve constant factor approximation of the optimal flowtime. More precisely, in [3], the author shows that the **Equi** policy, introduced in the 1980's by [12] and implemented in a lot of real systems (by time-multiplexing), achieves a competitive ratio of $(2+\sqrt{3})$ for overall completion time minimization when all the jobs arrive at time 0. [2] shows that in this setting no non-clairvoyant scheduler can achieve a competitive ratio better than $\Omega(\sqrt{n})$ when jobs arrive at arbitrary time and shows that **Equi** achieves a constant factor approximation of the optimal flowtime if it receives slightly more than twice as much resources as the optimal clairvoyant schedule it is compared to.

Our Contribution. Based on the particular case of malleable jobs, it is generally assumed in the literature that "**Equi** never starves a job" since it allocates to every job the same amount of processing power. We provide an analysis of the competitiveness of **Equi** for the makespan objective (Theorem 1) which shows that this statement is at the same time true and false: false, because, as opposed to the analysis of **Equi** for the flowtime objective in [3,2] where only the fully parallel phases count, for the makespan minimization, these phases may arbitrarily delay sequential work and thus stretch the makespan by a factor as large as, but no more than, $\frac{\ln n}{\ln \ln n}$ with respect to the optimum (Proposition 1); true, because **Equi** is as fair as can be since no algorithm can achieve a better competitive ratio up to a constant factor (Proposition 2). Furthermore, experiments in Section 5 tend to show that this worst case result may also be the typical behavior of **Equi** on random instances.

Non-clairvoyant competitiveness has been shown to be a powerful tool to analyze online strategies in various domains (for instance, [4,11]). In this paper, we aim to extend the non-clairvoyance toolbox by extending the results in [2] and [11] to the batch scheduling of sets of jobs, that go through *arbitrary* phases: each user sends, all at once, at time 0, a request for the execution of a set of jobs and is served when the last job completes. We prove that the natural algorithm **Equi**$\circ$**Equi** achieves a competitive ratio of $(2+\sqrt{3}+o(1))\frac{\ln n}{\ln \ln n}$ (Theorem 2), where n is the maximum size of a set, which is optimal up to a constant factor. We provide experimental evidences (Section 5) that this algorithm may have the same asymptotic competitive ratio $\Theta(\frac{\ln n}{\ln \ln n})$ (independent of the number of requests) for the flowtime objective when requests have release dates, if it is given sufficiently large extra processing power with respect to the optimum.

Besides its theoretical interest, this setting corresponds to the case where several users send sets of unrelated calculations to a cluster farm that proceeds the user requests by batches, for instance because a full hardware check is performed between each pair of consecutive batches. As a byproduct of our analysis, we extend the reduction shown by Edmonds in [2, Lemma 1], by showing (Theorem 3) that one only needs to consider jobs consisting of sequential or parallel work *whatever* the objective function is (flowtime, makespan, overall set completion time, stretch, energy consumption,...) in order to treat the very wide range of non-decreasing sublinear speed-up functions all at once.

2 Non-clairvoyant Batch Sets Scheduling

The Model. We consider a collection $S = \{S_1, \ldots, S_m\}$ of sets $S_i = \{J_{i,1}, \ldots, J_{i,n_i}\}$ of n_i jobs, each of them arriving at time zero. A *schedule* $\mathcal{S}_p$ on p processors is a set of piecewise constant functions[1] $\rho_{ij} : t \mapsto \rho_{ij}^t$ where ρ_{ij}^t is the *amount of processors* allotted to job J_{ij} at time t; (ρ_{ij}^t) are arbitrary non-negative real numbers, such that at any time: $\sum_{i,j} \rho_{ij}^t \leqslant p$. Following the definition introduced by [3], each job J_{ij} goes through a series of *phases* $J_{ij}^1, \ldots, J_{ij}^{q_{ij}}$ with different degree of parallelism; the amount of *work* in each phase J_{ij}^k is w_{ij}^k; at time t, during its k-th phase, job J_{ij} progresses at a *rate* given by a *speed-up function* $\Gamma_{ij}^k(\rho_{ij}^t)$ of the amount ρ_{ij}^t of processors allotted to J_{ij}, that is to say that the amount of work accomplished between t and $t + dt$ during phase J_{ij}^k is $\Gamma_{ij}^k(\rho_{ij}^t)dt$. Let t_{ij}^k denote the completion time of the k-th phase of J_{ij}, *i.e.* t_{ij}^k is the first time t' such that $\int_{t_{ij}^{k-1}}^{t'} \Gamma_{ij}^k(\rho_{ij}^t)\, dt = w_{ij}^k$ (with $t_{ij}^0 = 0$). Job J_{ij} is completed at time $c_{ij} = t_{ij}^{q_{ij}}$. A schedule is *valid* if all jobs eventually complete, *i.e.*, $c_{ij} < \infty$ for all i, j. Set S_i is completed at time $c_i = \max_{j=1..n_i} c_{ij}$.

The Problem. The *overall completion time* of the jobs in a schedule $\mathcal{S}_p$ is: $\mathsf{CompletionTime}(\mathcal{S}_p) = \sum_{i,j} c_{ij}$. The *makespan* of the jobs in $\mathcal{S}_p$ is: $\mathsf{Makespan}(\mathcal{S}_p) = \max_{i,j} c_{ij}$. The *overall set completion time* of the sets in $\mathcal{S}_p$ is: $\mathsf{SetCT}(\mathcal{S}_p) = \sum_{i=1}^m c_i$. Note that: if the input collection S consists of a single set S_1, the overall set completion time of a schedule $\mathcal{S}_p$ is simply the makespan of the jobs in S_1; and if S is a collection of singleton sets $S_i = \{J_{i\,1}\}$, the overall set completion time of $\mathcal{S}_p$ is simply the overall completion time of the jobs. The overall set completion time allows then to measure a continuous range of objective functions from makespan to overall completion time. Our goal is to minimize the overall set completion time of a collection of sets of jobs arriving at time 0.

[1] Requiring the functions (ρ_{ij}) to be piecewise constant is not restrictive since any finite set of reasonable (*i.e.*, Riemann integrable) functions can be uniformly approximated from below within an arbitrary precision by piecewise constant functions. In particular, all of our results hold if ρ_{ij} are piecewise continuous functions.

We denote by $\mathrm{OPT}_p(S)$ (or simply OPT_p or OPT if the context is clear) the optimal overall set completion time of a valid schedule on p processors for collection S: $\mathrm{OPT}_p = \inf_{\text{all schedules } \mathcal{S}_p} \mathsf{SetCT}(\mathcal{S}_p)$.

Speed-Up Functions. As in [2], we make the following reasonable assumptions on the speed-up functions. In the following, we consider that each speed-up function is *non-decreasing* and *sub-linear* (*i.e.*, such that for all i, j, k, $\rho < \rho' \Rightarrow \frac{\Gamma_{ij}^k(\rho)}{\rho} \geqslant \frac{\Gamma_{ij}^k(\rho')}{\rho'}$). These assumptions are usually verified (at least desirable...) in practice: non-decreasing means that giving more processors cannot deteriorate the performances; sub-linear means that a job makes a better use of fewer processors: this is typically true when parallelism does not take too much advantage of local caches. As shown in [2], two types of speed-up functions will be of particular interest here:

– the *sequential* phase, where $\Gamma(\rho) = 1$ for all $\rho \geqslant 0$ (the job progresses at constant speed even if *no* processor is allotted to it, similarly to an idle period); and
– the *fully parallel* phase, where $\Gamma(\rho) = \rho$ for all $\rho \geqslant 0$.

Two classes of instances will be useful in the following. We denote by $(\mathsf{Par\text{-}Seq})^*$ the class of all instances in which each phase of each job is either sequential or fully parallel, and by $\mathsf{Par\text{-}Seq}$ the class of all instances in which each job consists of a fully parallel phase followed by a sequential phase. Given a $(\mathsf{Par\text{-}Seq})^*$ job J, we denote by $\mathsf{par}(J)$ (resp., $\mathsf{seq}(J)$) the sum of the fully parallel (resp., sequential) works over all the phases of J. Given a set $S_i = \{J_{i,1}, \ldots, J_{i,n_i}\}$ of $(\mathsf{Par\text{-}Seq})^*$ jobs, we denote by $\mathsf{par}(S_i) = \sum_{j=1}^{n} \mathsf{par}(J_{ij})$ and $\mathsf{seq}(S_i) = \max_{j=1,\ldots,n_i} \mathsf{seq}(J_{ij})$.

Non-clairvoyant Scheduling. As in [3,2], we consider that the scheduler knows nothing about the progress of each job and is only informed that a job is completed *at the time of its completion.* In particular, it is not aware of the different phases that the job goes through (neither of the amount of work nor of the speed-up function). It follows that even if all the job sets arrive at time 0, the scheduler has to design an *online strategy* to adapt its allocation on-the-fly to the overall progress of the jobs.

We say that a given scheduler A_p is *c-competitive* if it computes a schedule $A_p(S)$ whose overall set completion time is at most c times the optimal *clairvoyant* overall set completion time (that is aware of the characteristics of the phases of each job), *i.e.*, such that $\mathsf{SetCT}(A_p(S)) \leqslant c \cdot \mathrm{OPT}_p(S)$ for all instances S. Due to the overwhelming advantage granted to the optimum which knows all the hidden characteristics of the jobs, it is sometimes necessary for obtaining relevant informations on an non-clairvoyant algorithm to limit the power of the optimum by reducing its resources. We say that a scheduler A_p is *s-speed c-competitive* if it computes a schedule $A_{sp}(S)$ on sp processors whose overall set completion time is at most c times the optimal overall set completion time on only p processors, *i.e.*, such that $\mathsf{SetCT}(A_{sp}(S)) \leqslant c \cdot \mathrm{OPT}_p(S)$ for all instances S.

We analyse two non-clairvoyant schedulers, namely **Equi** and **Equi** $\circ$ **Equi**, and show that they have an optimal competitive ratio up to constant multiplicative factors. The following two theorems are our main results and are proved in Propositions 1, 2 and 3.

Theorem 1 (Makespan minimization). Equi *is a* $\frac{(1+o(1))\ln n}{\ln\ln n}$*-competitive non-clairvoyant algorithm for the makespan minimization of a set of* n *jobs arriving at time* $t = 0$*. Furthermore, no non-clairvoyant deterministic (resp. randomized) algorithm is* s*-speed* c*-competitive for any* $s = o(\frac{\ln n}{\ln\ln n})$ *and* $c < \frac{\ln n}{2\ln\ln n}$ *(resp.* $c < \frac{\ln n}{4\ln\ln n}$*).*

Theorem 2 (Main result). Equi∘Equi *is a* $\frac{(2+\sqrt{3}+o(1))\ln n}{\ln\ln n}$*-competitive non-clairvoyant algorithm for the overall set completion time minimization of a collection of sets of jobs arriving at time* $t = 0$*, where* n *is the maximum cardinality of the sets. (Clearly the lower bound on competitive ratio given above holds as well for this problem).*

3 Reduction to (Par-Seq)* Instances

In [2], Edmonds shows that for the flowtime objective function, one can reduce the analysis of the competitiveness of non-clairvoyants algorithm to the instances composed of a sequence of infinitesimal sequential or parallel work. It turns out that as shown in Theorem 3 below, his reduction is far more general and applies to *any* reasonable objective function (including makespan, overall set completion time, stretch, energy consumption,...), and furthermore reduces the analysis to instances where jobs are composed of a finite sequence of positive sequential or fully parallel work, *i.e.*, to *true* (Par-Seq)* instances. It follows that for *any* non-clairvoyant scheduling problem, it is enough to analyse the competitiveness of a non-clairvoyant algorithm on (Par-Seq)* instances. Sequential and parallel phases are both unrealistic in practise (sequential phases that progress at a constant rate even if they receive no processors are not less legitimate than fully parallel phases which do not exist for real either). Nevertheless, these are much easier to handle in competitive analysis.

Consider a collection of n jobs $J_1, \ldots, J_n$ where J_i consists of a sequence of phases $J_i^1, \ldots, J_i^{q_i}$ of work $w_i^1, \ldots, w_i^{q_i}$ with speed-up functions $\varGamma_i^1, \ldots, \varGamma_i^{q_i}$. Consider a speed $s > 0$. Let A_{sp} be an arbitrary non-clairvoyant scheduler on sp processors, and $\mathcal{O}_p$ a valid schedule of $J_1, \ldots, J_n$ on p processors. The principle, see [2], is to remap the phases of the jobs within the two schedules $A_{sp}(J)$ and $\mathcal{O}_p$ as follows: each time A_{sp} allots more processors to some phase of a job than $\mathcal{O}_p$, this phase is substituted by a sequential phase and thus A_{sp} allots these resources pointlessly; and reciprocally, each time A_{sp} allots less processors to some other phase of a job than $\mathcal{O}_p$, we substitute this phase by a parallel phase. We adjust the substituted sequential and parallel works so that they fit exactly in the schedule computed by A_{sp}, which implies, as A_{sp} is non-clairvoyant, that A_{sp} will compute the exact same schedule as before; and since the instance is made easier to $\mathcal{O}_p$, the optimum can only decrease. This ensures that the competitive ratio of A_{sp} on any instance is upper bounded by the competitive ratio on (Par-Seq)* jobs.

Note that [2] implicitly assumed that the algorithm is monotonic (*i.e.*, its flowtime increases if some phase gets more work, which is the case of **Equi**), while the present reduction to $(\mathsf{Par\text{-}Seq})^*$ instances applies to any algorithm and furthermore to settings with release dates, precedences constraints, or any other type of constraints, since Lemma 1 simply consists in remapping the phases of the jobs *within* two *valid* schedules that already satisfy these additional constraints.

Lemma 1 (Reduction to $(\mathsf{Par\text{-}Seq})^*$ instances). *There exists a collection of* $(\mathsf{Par\text{-}Seq})^*$ *jobs* $J'_1, \ldots, J'_n$ *such that* $\mathcal{O}_p[J'/J]$ *is a valid schedule of* $J'_1, \ldots, J'_n$ *and* $A_{sp}(J') = A_{sp}(J)[J'/J]$*, where* $\mathcal{S}[J'/J]$ *denotes the schedule obtained by scheduling job* J'_i *instead of* J_i *in a schedule* $\mathcal{S}$.

Proof. The present proof only simplifies the proof originally given in [2] in the following ways: the jobs $J'_1, \ldots, J'_n$ consist of a *finite* number of phases (and are thus a valid finitely described instance), and the schedules computed by algorithm A_{sp} on instances $J'_1, \ldots, J'_n$ and $J_1, \ldots, J_n$ are identical, which avoids to consider infinitely many schedules to construct J' from J.

Consider the two schedules $A_{sp}(J)$ and $\mathcal{O}_p$. Consider job J_1 (the construction of J'_i is identical for J_i, $i \geqslant 2$). Let $\rho_A(t)$ and $\rho_{\mathcal{O}}(t)$ be the number of processors allotted over time to J_1 by $A_{sp}(J)$ and $\mathcal{O}_p$ respectively. Let $\varphi(t)$ be the time t' at which the portion of work of J_1 executed in $\mathcal{O}_p$ at time t, is executed in $A_{sp}(J)$. Let $\Gamma_{t'}$ be the speed-up function of the portion of work of J_1 executed in $A_{sp}(J)$ at time t'. By construction, for all t, the same portion of work dw of J_1 is executed between t and $t+dt$ in $\mathcal{O}_p$ and between $\varphi(t)$ and $\varphi(t+dt) = \varphi(t)+d\varphi(t)$ in $A_{sp}(J)$ with the same speed-up function $\Gamma_{\varphi(t)}$, thus: $dw = \Gamma_{\varphi(t)}(\rho_{\mathcal{O}}(t))\,dt = \Gamma_{\varphi(t)}(\rho_A(\varphi(t)))\,d\varphi(t)$; it follows that φ's derivative is $\varphi'(t) = \frac{\Gamma_{\varphi(t)}(\rho_{\mathcal{O}}(t))}{\Gamma_{\varphi(t)}(\rho_A(\varphi(t)))}$ $(\geqslant 0$, φ is an increasing function). $\rho_A(\varphi(t))$ and $\rho_{\mathcal{O}}(t)$ are (by definition) piecewise constant functions. Let $t_1 = 0 < t_2 < \cdots < t_\ell$ such that $\rho_A(\varphi(t))$ and $\rho_{\mathcal{O}}(t)$ are constant on each time interval $[t_k, t_{k+1})$ and zero beyond t_ℓ; let $t'_k = \varphi(t_k)$, $\rho_A(t')$ is constant on each time interval (t'_k, t'_{k+1}); let $\rho_A^k = \rho_A(t'_k)$ and $\rho_{\mathcal{O}}^k = \rho_{\mathcal{O}}(t_k)$. By construction, the portion of work of J_1 executed by $A_{sp}(J)$ between times t'_k and t'_{k+1}, is executed by $\mathcal{O}_p$ between times t_k and t_{k+1}. J'_1 consists of a sequence of $(\ell - 1)$ phases, sequential or fully parallel depending on the relative amount of processors $\rho_{\mathcal{O}}^k$ and ρ_A^k alloted by $\mathcal{O}_p$ and $A_{sp}(J)$ to J_1 during time intervals $[t_k, t_{k+1}]$ and $[t'_k, t'_{k+1}]$ respectively. The k-th phase of J'_1 is defined as follows:

– If $\rho_{\mathcal{O}}^k \leqslant \rho_A^k$, the k-th phase of J'_1 is a sequential work of $w_k = t'_{k+1} - t'_k$.
– If $\rho_{\mathcal{O}}^k > \rho_A^k$, the k-th phase of J'_1 is a fully parallel work of $w_k = \rho_A^k \cdot (t'_{k+1} - t'_k)$.

The k-th phase of J'_1 is designed to fit exactly in the overall amount of processors allotted by A_{sp} to J_1 during $[t'_k, t'_{k+1}]$; thus, since A_{sp} is non-clairvoyant, $A_{sp}(J') = A_{sp}(J)[J'/J]$. Let now verify that the k-th phase of J'_1 fits in the overall amount of processors allotted by $\mathcal{O}_p$ to J_1 during $[t_k, t_{k+1}]$.

– If $\rho_{\mathcal{O}}^k \leqslant \rho_A^k$, $w_k = \int_{t'_k}^{t'_{k+1}} dt' = \int_{t_k}^{t_{k+1}} \varphi'(t)dt = \int_{t_k}^{t_{k+1}} \frac{\Gamma_{\varphi(t)}(\rho_{\mathcal{O}}^k)}{\Gamma_{\varphi(t)}(\rho_A^k)} dt \leqslant \int_{t_k}^{t_{k+1}} dt = t_{k+1} - t_k$ since the $\Gamma_{\varphi(t)}$ are non-decreasing functions.

– If $\rho_{\mathcal{O}}^k > \rho_A^k$, $w_k = \rho_A^k \int_{t'_k}^{t'_{k+1}} dt' = \rho_A^k \int_{t_k}^{t_{k+1}} \frac{\Gamma_{\varphi(t)}(\rho_{\mathcal{O}}^k)}{\Gamma_{\varphi(t)}(\rho_A^k)} dt \leqslant \rho_A^k \int_{t_k}^{t_{k+1}} \frac{\rho_{\mathcal{O}}^k}{\rho_A^k} dt = \rho_{\mathcal{O}}^k \cdot (t_{k+1} - t_k)$, since the $\Gamma_{\varphi(t)}$ are sub-linear functions.

It follows that in both cases, the k-th phase of J'_1 can be completed in the space allotted to J_1 in $\mathcal{O}_p$ during $[t_k, t_{k+1}]$. □

Consider an arbitrary non-clairvoyant scheduling problem where the goal is to minimize an objective function F over the set of all valid schedules of an instance of jobs $J_1, \ldots, J_n$. Assume that F is *monotonic* in the sense that if $\mathcal{S}$ and $\mathcal{S}[J'/J]$ are valid schedules of J and J' respectively, then $F(\mathcal{S}[J'/J]) \leqslant F(\mathcal{S})$, which means essentially that wasting resources (allocating more resources than needed or allocating processors after the completion of a job) costs no more than using them: putting a larger thing into a box costs at least as much as putting a smaller thing into the same box. Note that *all* standard objective functions are monotonic: flowtime, makespan, overall completion time, overall set completion time, stretch, energy consumption, etc. Then,

Theorem 3. *Any non-clairvoyant algorithm A^F for a monotonic objective function F that is s-speed c-competitive over* $(\mathsf{Par\text{-}Seq})^*$ *instances, is also s-speed c-competitive over all instances of jobs going through phases with arbitrary non-decreasing sublinear speed-up functions.*

Proof. Consider a non-$(\mathsf{Par\text{-}Seq})^*$ instance $J = \{J_1, \ldots, J_n\}$. Denote by $\mathrm{OPT}_p^F(J)$ the optimal cost for J, *i.e.*, $\mathrm{OPT}_p^F(J) = \inf\{F(\mathcal{S}) : \mathcal{S} \text{ is a valid schedule of } J \text{ on } p \text{ processors}\}$. Consider an arbitrary small $\epsilon > 0$ and $\mathcal{O}$ a valid schedule of J such that $F(\mathcal{O}) \leqslant \mathrm{OPT}_p^F(J) + \epsilon$ (note that we do not need that an optimal schedule exists). Let J' be the $(\mathsf{Par\text{-}Seq})^*$ instance given by Lemma 1 from J, A_{sp}^F, and $\mathcal{O}$. Since $A_{sp}^F(J') = A_{sp}^F(J)[J'/J]$, $F(A_{sp}^F(J)) = F(A_{sp}^F(J'))$. But A_{sp}^F is s-speed c-competitive for J', so: $F(A_{sp}^F(J)) \leqslant c \cdot \mathrm{OPT}_p^F(J') \leqslant c \cdot F(\mathcal{O}[J'/J]) \leqslant c \cdot F(\mathcal{O}) \leqslant c\,\mathrm{OPT}_p^F(J) + c\epsilon$, as $\mathcal{O}[J'/J]$ is a valid schedule of J' and F is monotonic. Decreasing ϵ to zero completes the proof. □

We shall from now on consider only $(\mathsf{Par\text{-}Seq})^*$ instances.

4 Fairness Is Fair Enough

4.1 The Single Set Case

In this section, we focus on the case where the collection S consists of a unique set $S_1 = \{J_1, \ldots, J_n\}$. The problem consists thus in minimizing the *makespan* of the set of jobs S_1. This problem is interesting on its own and, as far as we know, no competitive non-clairvoyant algorithm was known. Furthermore, the analysis that follows is one of the keys to the main result of the next section.

Equi *Algorithm.* **Equi** is the classic operating system approach to non-clairvoyant scheduling. It consists in giving an equal amount of processors to each uncompleted job (operating systems approximate this strategy by a preemptive round robin policy). Formally, given p processors, if $N(t)$ denotes the number of uncompleted jobs at time t, **Equi** allots $\rho_i^t = p/N(t)$ processors to each uncompleted job J_i at time t.

Analysis of Equi for makespan minimization. Thanks to Theorem 3, we focus on a $(\mathsf{Par\text{-}Seq})^*$ instance $S = \{J_1, \ldots, J_m\}$. By rescaling the parallel work in each job, we can assume w.l.o.g. that $p = 1$. Let us define the Par-Seq instance $S' = \{J_1', ..., J_n'\}$ where each J_i' consists of a fully parallel phase of work $\mathsf{par}(J_i)$ followed by a sequential phase of work $\mathsf{seq}(J_i)$. Observe that:

Lemma 2. $\mathsf{Makespan}(\mathbf{Equi}(S')) \geqslant \mathsf{Makespan}(\mathbf{Equi}(S))$.

Proof. Since all the jobs arrive at time 0, the number of uncompleted jobs is a non-increasing function of time. It follows that the amount of processors alloted by **Equi** to a given job is a non-decreasing function of time. Thus, moving all the parallel work to the front, can only delay the completion of the jobs since less processors will then be allocated to each given piece of parallel work. □

Now, every job consists of a parallel phase followed by a sequential phase of length at most $\mathsf{seq}(S')$. The key to the analysis is to observe that: if more than a proportion α of jobs are in a parallel phase, then the overall parallel work progresses at a rate at least α; and if more than a proportion $(1-\alpha)$ of jobs are in a sequential phase then after $\mathsf{seq}(S')$ time, these jobs are completed and the number of jobs decrease by a factor α. It follows that the parallel work is at most dilated by some factor $1/\alpha$ and the sequential phases get extended by some logarithmic factor. We thus obtain the following competitive ratio:

Proposition 1. Equi *is* $(1 + o(1))\frac{\ln n}{\ln \ln n}$*-competitive for the makespan minimization problem.*

Proof. Consider the schedule $\mathbf{Equi}(S')$ and let $T = \mathsf{Makespan}(\mathbf{Equi}(S'))$. We write $[0, T]$ as the disjoint union of two sets A and $\bar{A}$. Set $\alpha = \frac{(\ln \ln n)^2}{\ln n}$. Recall that $N(t)$ is the number of uncompleted jobs at time t. Let s_t be the number of uncompleted jobs in a sequential phase at time t. Set A is the set of all the instants where the fraction of jobs in a sequential phase is larger than α, and $\bar{A}$ is its complementary set: *i.e.*, $A = \{0 \leqslant t \leqslant T : s_t \geqslant (1-\alpha)N(t)\}$ and $\bar{A} = \{0 \leqslant t \leqslant T : s_t < (1-\alpha)N(t)\}$. Clearly, $T = |A| + |\bar{A}|$, with $|X| = \int_X dt$. We now bound $|A|$ and $|\bar{A}|$ independently.

At any time t in $\bar{A}$, the total amount of parallel work completed between t and $t + dt$ is at least $\alpha\, dt$. Since the total amount of parallel work is $\mathsf{par}(S')$, we get $\int_{\bar{A}} \alpha\, dt \leqslant \mathsf{par}(S')$. Thus, $|\bar{A}| \leqslant \mathsf{par}(S')/\alpha$.

Now, let $t_1 < \cdots < t_q$ with $t_k \in A$ for all k, such that the time intervals $I_1 = [t_1, t_1 + \mathsf{seq}(S')), \ldots, I_q = [t_q, t_q + \mathsf{seq}(S'))$ form a collection of non-overlapping intervals of length $\mathsf{seq}(S')$ that covers A. Once the sequential phase of a Par-Seq job has begun at or before time t, the job

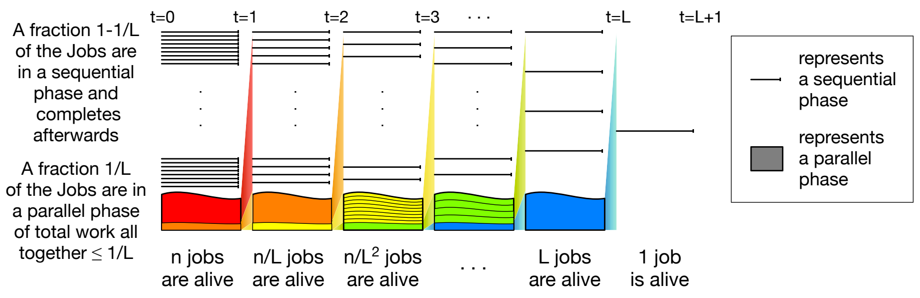
A fraction 1-1/L of the Jobs are in a sequential phase and completes afterwards
A fraction 1/L of the Jobs are in a parallel phase of total work all together ≤ 1/L
t=0
t=1
t=2
t=3
t=L
t=L+1
n jobs are alive
n/L jobs are alive
n/L² jobs are alive
L jobs are alive
1 job is alive
represents a sequential phase
represents a parallel phase

Fig. 1. Worst case instance designed by the mean adversary

completes before time $t + \mathsf{seq}(S')$. Since at time t_k, at least $(1-\alpha)\cdot N(t_k)$ jobs are in a sequential phase, at time $t_{k+1} \geqslant t_k + \mathsf{seq}(S')$, we have thus: $N(t_{k+1}) \leqslant \alpha N(t_k)$. It follows that $N(t_k) \leqslant \alpha^k \cdot n$. Since $N(t_q) \geqslant 1$, $q \leqslant \frac{\ln n}{\ln(1/\alpha)}$. But A is covered by q time intervals of length $\mathsf{seq}(S')$, so: $|A| \leqslant \frac{\ln n}{\ln(1/\alpha)}\,\mathsf{seq}(S')$. Finally, $\mathsf{Makespan}(\mathbf{Equi}(S)) \leqslant \mathsf{Makespan}(\mathbf{Equi}(S')) = T \leqslant \frac{1}{\alpha}\,\mathsf{par}(S') + \frac{\ln n}{\ln(1/\alpha)}\,\mathsf{seq}(S') \leqslant (1+o(1))\frac{\ln n}{\ln\ln n}\max(\mathsf{par}(S'),\mathsf{seq}(S')) = (1+o(1))\frac{\ln n}{\ln\ln n}\max(\mathsf{par}(S),\mathsf{seq}(S)) \leqslant (1+o(1))\frac{\ln n}{\ln\ln n}\mathrm{OPT}(S)$. □

Equi is asymptotically optimal up to a factor 2. The following lemma shows that **Equi** is asymptotically optimal in the worst case. Note that increasing the number of processors by a factor s does *not* improve the competitive ratio of any deterministic or randomized algorithm as long as $s = o(\frac{\ln n}{\ln\ln n})$, *i.e.*, the competitive ratio does not improve even if the number of processors increases (not too fast) with the number of jobs. Figure 1 presents the worst case instance used below.

Proposition 2 (Lower bound for any non-clairvoyant algorithm). *No non-clairvoyant algorithm A has a competitive ratio less than $\gamma_D = \frac{\ln n}{2\ln\ln n}$ if A is deterministic, and $\gamma_R = \frac{\ln n}{4\ln\ln n}$ if A is randomized.*

Furthermore, no non-clairvoyant algorithm A is s-speed c-competitive for any speed $s = o(\frac{\ln}{\ln\ln n})$ if $c < \gamma_D$ and A is deterministic, or $c < \gamma_R$ if A is randomized.

Proof. Consider the execution of an algorithm A_s given s processors on the following instance (see Fig. 1). At time 0, $n = (s\ell)^\ell$ jobs are given. Since the algorithm is non-clairvoyant, we set the phase afterwards. At time 1, we renumber the jobs $J_1,\ldots,J_n$ by non-decreasing processing power received between $t=0$ and $t=1$ in A_s. Between time $t=0$ and $t=1$, we set the jobs $J_{(s\ell)^{\ell-1}+1},\ldots,J_n$ (*i.e.*, the last fraction $1-1/(s\ell)$ of the $(s\ell)^\ell$ jobs) to be in a sequential phase of work 1 and say that all of them complete at time 1; each J_j of the $J_1,\ldots,J_{(s\ell)^{\ell-1}}$ are set in a parallel phase of work $\int_0^1 \rho_j^t\,dt$ each between time 0 and 1, where ρ_j^t is the amount of processors alloted to J_i at time t. The processing power received by the last $1-1/(s\ell)$ fraction of jobs between $t=0$ and $t=1$ is at least $s-1/\ell$

and thus, the total parallel work assigned to the jobs between 0 and 1 is at most $1/\ell$. At time 1 only remains the jobs $J_1, \ldots, J_{(s\ell)^{\ell-1}}$ that just have finished their first parallel phase. We continue recursively as follows until time $t = \ell$: at integer time $t = i < \ell$, $(s\ell)^{\ell-i}$ jobs are still uncompleted; between time $t = i$ and $t = i + 1$, the fraction $1 - 1/(s\ell)$ of the $(s\ell)^{\ell-i}$ jobs that received the most processing power are set in a sequential phase of work 1 and all of them complete at time $i + 1$; each job J_j of the other $1/(s\ell)$ fraction is set in a parallel phase of work $\int_i^{i+1} \rho_j^t \, dt$ each; At time $i + 1$ only remains the $(s\ell)^{\ell-i}/(s\ell) = (s\ell)^{\ell-(i+1)}$ jobs that just have finished their i-th parallel phase. At time $t = \ell$, there only remains one job which completes at time $\ell+1$ after a sequential phase of work 1. It follows that for this instance, A_s achieves a makespan of $\ell+1$. But, the amount of parallel work executed within each time interval $[i, i + 1]$ for $i = 0, ..., \ell - 1$, is at most $1/\ell$. It follows that an optimal (clairvoyant) scheduler on 1 processor can complete all the parallel work in one time unit and then finish the remaining sequential work before time 2. But $n = (s\ell)^\ell$, $\ell > \frac{\ln n}{\ln \ln n}$, which concludes the proof. *(The proof for randomized algorithms, based on Yao's principle [13,10], is omitted due to space constraints).* □

4.2 Non-clairvoyant Batch Set Scheduling

We now go back to the general problem. Consider a collection $S = \{S_1, \ldots, S_m\}$ of m sets $S_i = \{J_{i,1}, \ldots, J_{i,n_i}\}$ of n_i $(\mathsf{Par\text{-}Seq})^*$ jobs, each of them arriving at time zero. The goal is to minimize the overall set completion time of the sets.

Equi∘Equi *Algorithm.* We consider the **Equi∘Equi** strategy which splits evenly the amount of processors given to each set among the uncompleted jobs within that set. Formally, let $N(t)$ be the number of uncompleted sets at time t, and $N_i(t)$ the number of uncompleted jobs in each uncompleted set S_i at time t. At time t, **Equi∘Equi** on p processors allots to each uncompleted job J_{ij} an amount of processors $\rho_{ij}^t = \frac{p}{N(t) \cdot N_i(t)}$. The following section shows that indeed the competitive ratio of this strategy is asymptotically optimal (up to a constant multiplicative factor).

Competitiveness of **Equi∘Equi**. Scaling by a factor p each sequential work, again we assume w.l.o.g. that $p = 1$. Consider the $\mathsf{Par\text{-}Seq}$ instance $S' = \{S'_1, \ldots, S'_m\}$ where $S'_i = \{J'_{i,1}, \ldots, J'_{i,n_i}\}$ and each job J'_{ij} consists of a fully parallel phase of work $\mathsf{par}(J_{ij})$ followed by a sequential phase of work $\mathsf{seq}(J_{ij})$. Following the proof of Lemma 2, we get:

Lemma 3. $\mathsf{SetCT}(\mathbf{Equi} \circ \mathbf{Equi}(S')) \geqslant \mathsf{SetCT}(\mathbf{Equi} \circ \mathbf{Equi}(S))$.

The next lemmas are the keys to the result. They reduce the analysis of **Equi∘Equi** to the analysis of the *overall completion time* of **Equi** for a collection of *jobs*, which is known from [3] to be $(2 + \sqrt{3})$-competitive when all the jobs arrive at time 0. Let $n = \max_{i=1,\ldots,m} n_i$ be the maximum size of a set S_i, and let $\alpha = \frac{(\ln \ln n)^2}{\ln n}$. The principle is to reduce the analysis of sets of jobs to the analysis of single $(\mathsf{Par\text{-}Seq})^*$ jobs as follows. Consider the lifetime of a set, each time a proportion more than α of the jobs are in a parallel phase within the set, we create a parallel phase; and each time a proportion more than $(1-\alpha)$ of the jobs are in sequential phase, we create a sequential phase. For the same raison as before, the overall parallel work gets dilated by at most $1/\alpha$ while the overall sequential work is extended by at most a logarithmic factor.

Lemma 4. *There exists a* (Par-Seq)* *instance* $J = \{J_1, \dots, J_m\}$ *of Non-Clairvoyant Batch* Job *Scheduling, such that:* $\mathbf{Equi}(J) = \mathbf{Equi} \circ \mathbf{Equi}(S')[J/S']$, $\mathsf{par}(J_i) \leqslant \frac{1}{\alpha}\,\mathsf{par}(S'_i)$, *and* $\mathsf{seq}(J_i) \leqslant \frac{\ln n}{\ln(1/\alpha)}\,\mathsf{seq}(S'_i)$, *where* $\mathcal{S}[J/S']$ *denotes the schedule where* J_i *receives at any time the total amount of processors alloted to the jobs* J'_{ij} *of* S'_i *in schedule* $\mathcal{S}$.

Proof. Let $\mathcal{E} = \mathbf{Equi} \circ \mathbf{Equi}(S')$. Let us construct J_1 (the construction of J_i, $i \geqslant 2$, is identical). Consider the jobs $J'_{1,1}, \dots, J'_{1,n_1}$ of S'_1 in the schedule $\mathcal{E}$. Let $t_1 = 0 < \dots < t_q = c'_1$ (where c'_1 denotes the completion time of S'_1 in $\mathcal{E}$), such that during each time interval $[t_k, t_{k+1})$, each job $J'_{1,j}$ remains in the same phase; during $[t_k, t_{k+1})$, the number of jobs of S'_1 in a sequential (resp. fully parallel) phase is constant, say s_k (resp. $N_1(t_k) - s_k$). J_1 has $(q-1)$ phases:

- if $s_k \geqslant (1-\alpha)N_1(t_k)$, the k-th phase of J_1 is sequential of work $w_k = t_{k+1} - t_k$.
- if $s_k < (1-\alpha)N_1(t_k)$, the k-th phase of J_1 is fully parallel of work $w_k = \int_{t_k}^{t_{k+1}} \frac{1}{N(t)} dt$.

J_1 is designed to fit exactly in the space alloted to S'_1 in $\mathcal{E}$, thus $\mathbf{Equi}(J) = \mathcal{E}[J/S']$. We now have to bound the total parallel and total sequential works in J_1. Let $K = \{k : s_k \geqslant (1-\alpha)N_1(t_k)\}$ and $\bar{K} = \{1, \dots, q-1\} \setminus K$; by construction, $\mathsf{seq}(J_1) = \sum_{k\in K} w_k$ and $\mathsf{par}(J_1) = \sum_{k \in \bar{K}} w_k$. For each $t \in [t_k, t_{k+1})$ with $k \in \bar{K}$, the amount of parallel work of jobs in S'_1 between t and $t + dt$ is at least $\frac{\alpha N_1(t)}{N(t) \cdot N_1(t)}\, dt = \frac{\alpha}{N(t)}\, dt$. It follows that the amount of parallel work of jobs in S'_1 scheduled in $\mathcal{E}$ during $[t_k, t_{k+1})$ is at least $\alpha \int_{t_k}^{t_{k+1}} \frac{1}{N(t)} dt = \alpha\, w_k$. Thus, $\mathsf{par}(S'_1) \geqslant \sum_{k\in\bar{K}} \alpha\, w_k = \alpha\,\mathsf{par}(J_1)$, which is the claimed bound. Now, let $A = \cup_{k\in K}[t_k, t_{k+1})$, we have $|A| = \mathsf{seq}(J_1)$. Since the bound on the size of A in proof of Proposition 1 relies on a counting argument (and is thus independent of the amount of processors given to the set) and the jobs in S'_1 are Par-Seq, the same argument applies and $|A| \leqslant \frac{\ln n_1}{\ln(1/\alpha)}\,\mathsf{seq}(S'_1) \leqslant \frac{\ln n}{\ln(1/\alpha)}\,\mathsf{seq}(S'_1)$, which conclude the proof. □

Let $J' = \{J'_1, \dots, J'_m\}$ be the Par-Seq instance of Batch Job Scheduling where each job J'_i consists of a fully parallel work of $\mathsf{par}(J_i)$ followed by a sequential work of $\mathsf{seq}(J_i)$. Again, as the amount of processors alloted by **Equi** to each job is a non-decreasing function of time, pushing parallel work upfront can only make it worse, thus: $\mathbf{Equi}(J) \leqslant \mathbf{Equi}(J')$.

We can now conclude on the competitiveness of $\mathbf{Equi} \circ \mathbf{Equi}$ since **Equi** is proved to be $(2+\sqrt{3})$-competitive for instance J' in [3, Theorem 3.1].

Proposition 3. $\mathbf{Equi}\circ\mathbf{Equi}$ *is* $\frac{(2+\sqrt{3}+o(1))\ln n}{\ln\ln n}$*-competitive for the overall set completion time minimization problem.*

5 Experimental Study of Equi and Equi∘Equi

Fig. 2(a) presents the empirical overall and maximum competitive ratio of **Equi** for the Makespan objective on random stream-lined instances [2]: for each n, a set of n jobs, of $5n$ phases of unit work each, is requested at time 0; for all i, the type of the i-th phase of the n jobs are defined as follows: a job is chosen uniformly at random and its i-th phase is set to parallel, and the i-th phases of all the other jobs are all sequential. It appears that the empirical competitive ratio measured for **Equi** on this random instance appears to be asymptotically $\sim .7\frac{\ln n}{\ln\ln n}$. The worst case ratio proven in Proposition 2 seems thus to be the typical behavior of **Equi**.

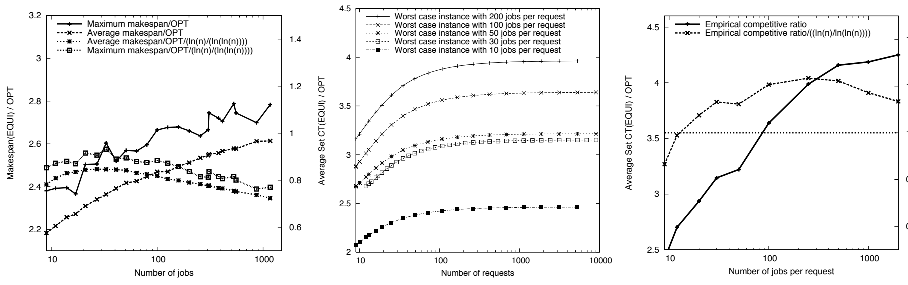

Fig. 2. From left to right: a) Makespan of **Equi** on random stream-lined instances; b) Average Set Flowtime of **Equi**∘**Equi** on a stream of worst case instances for Makespan on 2 processors; c) Worst empirical competitive ratio for **Equi** ∘ $\mathbf{Equi}_2$ w.r.t. OPT_1

Fig. 2(b) and (c) present the competitive ratio of **Equi**∘**Equi** for the Average Set Flowtime objective (the average flowtime of each set of jobs) with 2 processors with respect to the optimal with 1 processor, on the following instance: at each time step $t = 0..10\,000$, we request an instance of the worst case type described in Fig. 1 with n jobs, $n \in \{10, 30, 50, 100, 200\}$. Fig. 2(c) shows that the empirical competitive ratio seems to tend asymptotically to $\sim \frac{\ln n}{\ln\ln n}$ as time grows. It seems thus that the competitive ratio of **Equi**∘**Equi** is independent of the number of requests for the Average Set Flowtime objective as well.

References

1. Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.(eds.): Handbook on Scheduling: Models and Methods for Advanced Planning, chapter Online Scheduling. International Handbooks on Information Systems. Springer, Heidelberg (2007), at `http://www.cs.pitt.edu/~kirk/papers/index.html`
2. Edmonds, J.: Scheduling in the dark. In: Proc. of the 31st ACM Symp. on Theory of Computing (STOC), pp. 179–188. ACM Press, New York (1999)
3. Edmonds, J., Chinn, D.D., Brecht, T., Deng, X.: Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. J. Scheduling 6(3), 231–250 (2003)

4. Edmonds, J., Pruhs, K.: Broadcast scheduling: when fairness is fine. In: Proc. of the 13th annual ACM-SIAM symp. Society for Industrial and Applied Mathematics, pp. 421–430. ACM Press, New York (2002)
5. Feldmann, A., Kao, M.-Y., Sgall, J., Teng, S.-H.: Optimal online scheduling of parallel jobs with dependencies. J. of Combinatorial Optimization 1, 393–411 (1998)
6. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45, 1563–1581 (1966)
7. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 263–269 (1969)
8. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47(4), 617–643 (2000)
9. Motwani, R., Philipps, S., Torng, E.: Non-clairvoyant scheduling. Theoretical Computer Science 130, 17–47 (1994)
10. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
11. Robert, J., Schabanel, N.: Pull-based data broadcast with dependencies: Be fair to users, not to items. In: Proc. of Symp. on Discrete Algorithms (SODA) (2007)
12. Tucker, A., Gupta, A.: Process control and scheduling issues for mulitprogrammed shared memory multiprocessors. In: Proc. of the 12th ACM Symp. on Op. Syst. Principles, pp. 159–166. ACM Press, New York (1989)
13. Yao, A.: Probabilistic computations: Towards a unified measure of complexity. In: Proc. of 17th Symp. on Fond. of Computer Science (FOCS), pp. 222–227 (1977)