Non-Clairvoyant Scheduling with Precedence Constraints

Julien Robert[*,†]   Nicolas Schabanel[†,*]

Abstract

We consider Edmonds's model (1999) extended by precedence constraints. In our setting, a scheduler has to schedule non-clairvoyantly jobs consisting in DAGs of tasks arriving over time, each task going through phases of different degrees of parallelism, unknown to the scheduler. As in the original model without precedence constraints, the scheduler is only informed of the arrival and the completion of each task, at the time of these events, and nothing more. Furthermore, it is not aware of the DAG structure of each job beforehand neither of the precise characteristics of the phases of the tasks that compose each job.

We consider the preemptive strategy EQUI∘EQUI, that divides the processors evenly among the alive jobs and then divides the processing power alloted to each job evenly among its alive tasks. We show that whatever how complex the precedences are, EQUI∘EQUI is $(2 + \epsilon)$-speed $O(\kappa/\epsilon)$-competitive for the flowtime metric, where κ is the maximum number of independent tasks in each job. That is to say, the flowtime of the schedule computed by EQUI∘EQUI is at a constant ratio of the optimal flowtime as soon as EQUI is given slightly more than twice the resources as the optimum it is compared to. Interestingly, the extra speed needed to obtain a competitive algorithm, namely $(2+\epsilon)$, is the *same* in presence of precedence constraints, as in the original setting without precedences studied by Edmonds in 1999. This means that the maximum load that the system can handle without diverging, is the *same* with or without precedence constraints.

Furthermore, we propose a simple scheme to analyze a special class of schedulers, namely EQUI-schedulers, which allows to obtain upper and lower bounds on particular precedences structures, such as independent chains, IN-trees, OUT-trees and Serial-parallel DAGs.

Keywords: Online scheduling, Precedences, Non-clairvoyant algorithm, Fairness, Equi-partition.

1 Introduction

We consider Edmonds's model [2] extended by precedence constraints. In our setting, a non-clairvoyant scheduler has to schedule jobs consisting in DAGs of tasks arriving over time, each task going through phases of different degrees of parallelism, unknown to the scheduler. As in the original model without precedence constraints, the scheduler is informed of nothing more than the arrival and the completion of each task, at the time of these events, and not before. Furthermore, it is not aware of the DAG structure of each job beforehand neither of the characteristics of the phases of the tasks that compose each job. The DAG structure unfolds as the tasks are scheduled, without informing the scheduler of the precise precedence constraints that are activated. We aim to minimize the flowtime of the jobs, i.e. the sum of the time elapsed between the release of the job and the completion of the last of its tasks.

It is known since [2], that no non-clairvoyant algorithm is $o(\sqrt{n})$-competitive with respect to the optimum even in absence of precedence constraints. Furthermore no non-clairvoyant algorithm is known to be competitive with respect to an optimal offline strategy even if the latter is given only half of the processors the algorithm receives.

Non-clairvoyant scheduling was introduced by [10] in an attempt to design algorithms that are provably efficient for practical purposes. Several extensions have been proposed to include precedence constraints. One of the first extensions is [6] which studies the case where DAGs of fully parallelizable jobs to arrive over time. More recently, several other articles [1, 9, 7] study non-clairvoyant scheduling of DAGs with simpler job models than [2] but with more restrictions on the schedules (such as costly preemption, discrete slots,...).

Our contribution. We consider the preemptive strategy EQUI∘EQUI, that divides the processors evenly among the alive jobs and then divides the processing power alloted to each job evenly among its alive tasks. Since no algorithm can compete directly with the optimum, our analysis proceeds by resource augmentation. We show that whatever how complex the precedences are, EQUI∘EQUI is $(2 + \epsilon)$-speed $O(\kappa/\epsilon)$-competitive for the flowtime metric, where κ is the maximum number of independent tasks in each job. That is to say, the flowtime of the schedule computed by EQUI∘EQUI is at a constant ratio of the optimal flowtime as soon as EQUI is given slightly more than twice the resources as

---

[*]Université de Lyon – École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France. `http://perso.ens-lyon.fr/julien.robert`.

[†]CNRS – Centro de Modelamiento Matemático, Blanco Encalada 2120 Piso 7, Santiago de Chile. `http://www.cmm.uchile.cl/~schabanel`.

the optimum it is compared to. Surprisingly, the extra speed needed to obtain a competitive algorithm, namely $(2+\epsilon)$, is the *same* in presence of precedence constraints, as in the original setting without precedences studied by Edmonds in 1999. This means that the maximum load that the system can handle without diverging, namely $1/(2+\epsilon)$, is the *same* with or without precedence constraints.

More generally, we consider a special class of schedulers, the EQUI-schedulers, which divides evenly the processors among the alive jobs and then delegates to a task scheduler, the division of the processors received by each job among its alive tasks. We provide a simple framework to analyze these schedulers, from which we obtain upper and lower bounds on particular precedences structures, such as independent chains, IN-trees, OUT-trees and Serial-parallel DAGs.

Interestingly, competitiveness of non-clairvoyant algorithms in Edmonds's model has been proved to be a powerful tool to analyze other online algorithms in various settings, including TCP and databroadcast protocols [4, 5, 12]. Our results extend the non-clairvoyant toolbox by allowing precedence constraints.

Roadmap. Section 2 introduces the model studied in this paper. Section 3 shows that as in [2], one can reduce the analysis of the competitiveness of a non-clairvoyant scheduler to only two types of tasks: sequential and fully parallel. Section 4 first introduces the key notion of α-scatterer task scheduler which allows to handle the sequential tasks, and then shows how the parallel tasks can be treated as a whole to prove our main result:

THEOREM 1.1. (MAIN RESULT) EQUI∘EQUI *is an* $(2+\epsilon)$-*speed* $O(\kappa/\epsilon)$-*competitive algorithm for the non-clairvoyant scheduling problem with precedence constraints where* κ *is the maximum number of independent tasks in a job.*

2 Model and definition

We apologize in advance to the reader for the following tedious but rewarding paragraphs which are necessary to settle the problem formally throughout. Fortunately, Section 3 will demonstrate that only a much smaller and easier class of jobs needs to be considered for the analysis of competitiveness.

The problem. We consider a sequence of jobs $\{J_1, J_2, \ldots\}$ with release times $\{r_1, r_2, \ldots\}$. Following the terminology of [7], each job J_i consists in a set of tasks $\{J_{i,1}, \ldots, J_{i,m_i}\}$ with precedence constraints that the scheduler has to execute over p processors. Each task goes through different phases of different degree of parallelism and the scheduler has to decide on-the-fly the amount of processors to allot to each alive task as they appear in the system. The scheduler is *non-clairvoyant*, i.e., discovers the jobs at the time of their arrivals and the tasks at the time they become available; furthermore, it is unaware of the current degree of parallelism of each task (i.e., how they take advantage of more processing power) nor of the amount of work in each task; it is only informed that a task or a job is completed at the time of its completion. As in [2], we consider that the processors can be divided arbitrarily: fractional allocation is usually realized through time multiplexing in real systems.

Schedules. 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 the task J_{ij} at time t; (ρ_{ij}^t) are arbitrary non-negative real numbers, such that at any time t: $\sum_{ij} \rho_{ij}^t \leqslant p$.

The jobs. We extend the definition introduced by [2, 3, 11] as follows. Each job J_i consists of a directed acyclic graph (*DAG* for short) $(\{J_{i1}, \ldots, J_{i\,m_i}\}, \prec)$, where task J_{ij} is released as soon as all tasks J_{ik}, such that $J_{ik} \prec J_{ij}$, are completed. Job J_i is completed as soon as all its tasks are completed. Each task goes through a sequence of phases $J_{ij}^1, \ldots, J_{ij}^{q_{ij}}$ with different degrees of parallelism. Each phase J_{ij}^k consists in an amount of work w_{ij}^k and a *speed-up function* Γ_{ij}^k. At time t, during its k-th phase, each task J_{ij} progresses at a *rate* $\Gamma_{ij}^k(\rho_{ij}^t)$ which depends on the amount ρ_{ij}^t of processors allotted to J_{ij} by the scheduler, i.e., the amount of work accomplished between t and $t + dt$ in each task J_{ij} during its k-th phase is: $dw = \Gamma_{ij}^k(\rho_{ij}^t)dt$.

Given a schedule $\mathcal{S}_p$ of the jobs $\{J_1, J_2, \ldots\}$. A job or a task is *alive* as soon as it is released and until it is completed. Let c_{ij} denote the *completion time* of task J_{ij}. The *release time* r_{ij} of a task J_{ij} is: $r_{ij} = r_i$, the release time of Job J_i, if $J_{i,j}$ does not depend on any other task, i.e., if $J_{ik} \not\prec J_{ij}$ for all k; and $r_{ij} = \max\{c_{ik} : J_{ik} \prec J_{ij}\}$, otherwise. Let c_{ij}^k denote the completion time of the k-th phase of task J_{ij}: c_{ij}^k is the first time t' such that $w_{ij}^k = \int_{c_{ij}^{k-1}}^{t'} \Gamma_{ij}^k(\rho_{ij}^t)\,dt$ (with $c_{ij}^0 = r_{ij}$). Each task J_{ij} completes with its last phase, thus: $c_{ij} = c_{ij}^{q_{ij}}$. Job J_i is thus completed at time $c_i = \max_j c_{ij}$. A schedule is *valid* if all jobs eventually complete, i.e., if $c_i < \infty$ for all i. We denote by $W_{ij}(t)$ the *total work of task* J_{ij} *accomplished* at time t, i.e.:

---

[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.

for $t \leqslant c_{ij}$, $W_{ij}(t) = \sum_{k < \ell} w_{ij}^k + \int_{c_{ij}^{\ell-1}}^t \Gamma_{ij}^\ell(\rho_{ij}^t)dt$, where $\ell = \max\{k : c_{ij}^{k-1} \leqslant t\}$; and $W_{ij}(t) = w_{ij}$, for $t \geqslant c_{ij}$.

The model in [2, 3], where each job goes through a sequence of phases with different degree of parallelism, corresponds to the special case where each job consists in a single task. The model in [11], where each request consists in a set of jobs of the type of [2, 3], corresponds to the special case where each job consists in a graph of independent tasks.

Cost of a schedule. The *flowtime* F_i of a job J_i is the overall time J_i is alive in the system, *i.e.*, $F_i = c_i - r_i$. The *flowtime* of a schedule $\mathcal{S}_p$ is the sum of the flowtimes of the jobs: $\mathsf{Flowtime}(\mathcal{S}_p) = \sum_i F_i$. Our goal is to design a scheduler that minimizes the flowtime, which corresponds to the average response time of the system, which is a classic measure of quality of service. We denote by $\mathrm{OPT}_p = \inf\{\mathsf{Flowtime}(\mathcal{S}_p) : \text{valid schedule } \mathcal{S}_p\}$, the optimal cost on p processors.

Speed-up functions. As in [2, 3, 11], 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 reasonably verified 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 in [2, 3, 11], two types of speed-up functions will be of particular interest here:

- the *sequential* phases where $\Gamma(\rho) = 1$, for all $\rho \geqslant 0$ (the task progresses at a constant rate even if *no* processor is allotted to it, similarly to an idle period); and

- the *fully parallel* phases where $\Gamma(\rho) = \rho$, for all $\rho \geqslant 0$.

We say that a job J_i is PS if each of the phases of its tasks is either sequential or parallel. An instance is PS if all of its jobs are PS. For any task J_{ij} of a PS job J_i, we define $\mathsf{seq}(J_{ij})$ and $\mathsf{par}(J_{ij})$ as the overall sequential and parallel works in the task respectively: $\mathsf{seq}(J_{ij}) = \sum_{k\,:\,k\text{-th phase of }J_{ij}\text{ is sequential}} w_{ij}^k$ and $\mathsf{par}(J_{ij}) = \sum_{k\,:\,k\text{-th phase of }J_{ij}\text{ is fully parallel}} w_{ij}^k$.

Chains and antichains. Given a job J_i, a *chain* ξ of dependences is a sequence of tasks $J_{ij_1} \prec \cdots \prec J_{ij_k}$. We denote by $\mathsf{seq}(\xi) = \sum_{\ell=1}^k \mathsf{seq}(J_{ij_\ell})$ the overall sequential work to be done along the chain ξ. We denote by $\mathsf{seq}(J_i) = \max\{\mathsf{seq}(\xi) : \xi$ is a chain of dependences in $J_i\}$. Since each sequential task is executed at a rate independent of the amount of processing power it recieves, $\mathsf{seq}(J_i)$ is a lower bound on the flowtime of J_i in any valid schedule.

Two tasks J_{ij} and J_{ik} in J_i are *independent* (*i.e.*, their release and completion times are unrelated) if $J_{ij} \not\prec^* J_{ik}$ and $J_{ik} \not\prec^* J_{ij}$, where $\prec^*$ denotes the transitive closure of $\prec$. An *antichain* of length k in J_i is a set of k pairwise independent distinct tasks. We denote by $\kappa(J_i)$ the maximum length of an antichain in J_i. $\kappa(J_i)$ is an upper bound on the maximum number of tasks in J_i that can be scheduled simultaneously in any valid schedule.

Non-clairvoyant scheduling with precedence constraints. As in [2, 3, 11], we consider that the scheduler knows nothing about the progress of each tasks and is only informed that a job or a task is completed *at the time of its completion*; in particular, it is not aware of the different phases that each task goes through (neither of the amount of work nor of the speed-up function). Furthermore, tasks are released as soon as they become available without noticing the scheduler of the precedence constraints: if two tasks complete as other tasks are released, the scheduler is unable to guess which spawns which. In particular, the order in which the tasks of a given job are released depends heavily on the computed schedule, and the scheduler cannot even reconstruct the DAG *a posteriori* in general. It is only aware at all time of the IDs of the current alive jobs and of their alive tasks.

Competitiveness and resource augmentation. We say that a given scheduler A_p is *c-competitive* if it computes a schedule $A_p(S)$ whose flowtime is at most c times the optimal (off-line) *clairvoyant* flowtime (that is aware of the characteristics of the phases of each task and of the DAG of each job), *i.e.*, such that $\mathsf{Flowtime}(A_p(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances $J = \{J_1, J_2, \ldots\}$. Due to the overwhelming advantage granted to the optimum which knows all the hidden characteristics of the jobs, [2] shows that no algorithm is $o(\sqrt{n})$-competitive for n jobs even if each job consists of a single task. It is thus necessary to limit the power of the optimum by reducing its resources for obtaining relevant informations on an non-clairvoyant algorithm. We say that a scheduler A_p is *s-speed c-competitive* if it computes a schedule $A_{sp}(J)$ on sp processors whose flowtime is at most c times the optimal flowtime on p processors only, *i.e.*, such that $\mathsf{Flowtime}(A_{sp}(J)) \leqslant c \cdot \mathrm{OPT}_p(J)$ for all instances J. This analysis technique, which provides interesting insights on the relative performances of different algorithms that could not be compared directly to the optimum cost, is known as *resource augmentation* (see for instance [8]).

3 Reduction to PS 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 infinitely many infinitesimal sequential or fully parallel phases. Edmonds's proof relies implicitly on the *monotonicity* of the analyzed algorithm, EQUI, in the sense that increasing the work in one phase of a job can only increase the flowtime of the computed schedule. As shown in [11, Theorem 3], it turns out that this assumption is unnecessary and holds for *any* algorithm independently of its monotonicity. Furthermore, Edmonds proves that one can reduce the analysis to *streamlined instances* of infinitesimal sequential or parallel phases, *i.e.*, instances that come with an optimal schedule in which every parallel phase gets all the processors as soon as it is available (in particular, only one parallel phase is executed at any time, while the other alive jobs are idle, *i.e.*, in a sequential phase).

We extend this result to the setting with precedence constraints. Moreover, we improve Edmonds's result on two aspects: we reduce the analysis of non-clairvoyant algorithms to their competitiveness on PS instances where jobs are composed of a *finite* sequence of positive sequential or fully parallel work, for which there exists a *weakly streamlined* optimal schedule, *i.e.*, a schedule in which at any time at most one task among all the alive tasks in a parallel phase is scheduled; and this reduction holds for non-monotonic algorithm as well. It follows that for *any* non-clairvoyant scheduling problem, it is enough to analyse the competitiveness of a non-clairvoyant algorithm on streamlined instances. As shown in [2], this allows a huge simplification of the analysis because of the very simple form of the optimal schedule. As already noticed in [11], sequential and parallel phases are both unrealistic in practice (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, and Theorem 3.1 guarantees that these two extreme(ly simple) regimes are *sufficiently general* to cover the range of all possible non-decreasing sublinear functions. Furthermore, intuitively, streamlined instances are fundamentally the hardest cases for any non-clairvoyant algorithm for the following reason: at any time the algorithm should give all the processors to one single job, but it does not have a clue about which one to elect.

Consider an instance J and 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 on p processors. Theorem 3 in [11] shows that one can remap the phases

within each task J_{ij} such that:

1. every phase in the new instance (J'_{ij}) is either sequential or parallel;

2. the precedence constraints between the tasks are preserved;

3. the schedule computed by the non-clairvoyant scheduler A_{sp} on the new tasks (J'_{ij}) is exactly the same as on the tasks (J_{ij}); and 4) substituting each J_{ij} by J'_{ij} in $\mathcal{O}_p$ yields a valid schedule for the tasks (J'_{ij}).

LEMMA 3.1. (REMAPPING TO PS INSTANCES, THEOREM 3 IN [11]) *There exists a collection of* PS *jobs* $J'_1, \ldots, J'_n$ *such that:*

- *precedence constraints are preserved: for all* i, j, k, $J'_{ij} \prec J'_{ik} \Leftrightarrow J_{ij} \prec J_{ik}$;

- $\mathcal{O}_p[J'/J]$ *is a valid schedule of* $J'_1, \ldots, J'_n$; *in particular,* $\mathsf{Flowtime}(\mathcal{O}_p[J'/J]) \leqslant \mathsf{Flowtime}(\mathcal{O}_p)$;

- $A_{sp}(J') = A_{sp}(J)[J'/J]$,

where $\mathcal{S}[J'/J]$ *denotes the schedule obtained by scheduling task* J'_{ij} *instead of* J_{ij} *in a schedule* $\mathcal{S}$.

J' is a PS instance. We free in $\mathcal{O}_p[J'/J]$ all the processors allocated to sequential phases, since sequential phases progress at the same rate even if it receives no processor, the flowtime is unchanged. We rearrange the allocations of the parallel phases of the tasks in J' in $\mathcal{O}_p[J'/J]$ as follows: in every interval of time in which several tasks in a parallel phase are scheduled in $\mathcal{O}_p[J'/J]$, we reallocate all the processors within this time interval to these tasks in an arbitrary Round Robin order; this can only decrease the flowtime of the schedule since each phase will complete no later than in $\mathcal{O}_p[J'/J]$. We thus obtain a valid schedule $\mathcal{O}'_p$ of J', in which at any time if a parallel phase is scheduled, it receives all the processors. Since $\mathsf{Flowtime}(\mathcal{O}_p[J'/J]) \leqslant \mathsf{Flowtime}(\mathcal{O}_p)$, it follows that:

LEMMA 3.2. (A WEAKLY STREAMLINED SCHEDULE FOR J') *There exists a weakly streamlined schedule* $\mathcal{O}'_p$ *of* $J'_1, \ldots, J'_n$ *such that* $\mathsf{Flowtime}(\mathcal{O}'_p) \leqslant \mathsf{Flowtime}(\mathcal{O}_p)$.

By remapping the phases again, we can furthermore assume that the schedule computed by A_{sp} is *always late with respect to the schedule* $\mathcal{O}'_p$, *i.e.*, at any time the amount of completed work of each task J'_{ij} in A_{sp} is smaller or equal than in $\mathcal{O}'_p$: $W^A_{ij}(t) \leqslant W^{\mathcal{O}'}_{ij}(t)$, for all i, j, t.

LEMMA 3.3. (A_{sp} IS LATE) *There exists a collection of* PS *jobs* $J_1'', \ldots, J_n''$ *such that:*

- *precedence constraints are preserved: for all* i, j, k, $J_{ij}'' \prec J_{ik}'' \Leftrightarrow J_{ij} \prec J_{ik}$;

- $\mathcal{O}_p'[J''/J']$ *is a valid schedule of* $J_1'', \ldots, J_n''$;

- $A_{sp}(J'') = A_{sp}(J)[J''/J]$;

- A_{sp} *is always late with respect to* $\mathcal{O}_p'$: *for all task* J_{ij} *and all time* t, $W_{ij}^A(t) \leqslant W_{ij}^{\mathcal{O}'}(t)$, *where* $W_{ij}^{\mathcal{S}}(t)$ *denotes the completed work of task* J_{ij}'' *in schedule* $\mathcal{S}$ *up to time* t.

Proof. The idea consists in replacing the first phases of each task by a unique sequential phase until the first date between the completion time of the task in A_{sp}, and the time from which $\mathcal{O}_p'$ will never be late again with respect to A_{sp} on this task. This ensures that during this newly created first phase, the task progresses at the same rate in $\mathcal{O}_p'$ and in A_{sp} independently of the amount of processors allotted to it. We then just need to ensure that each task is released at least as early in $\mathcal{O}_p'$ as in A_{sp}, which is done inductively on the DAG structure of each job.

THEOREM 3.1. *For any non-clairvoyant algorithm A, if for all* PS *instances* $J = \{J_1, \ldots, J_n\}$ *and all weakly streamlined schedules* $\mathcal{O}_p$ *of* J *such that* A_{sp} *is late with respect to* $\mathcal{O}_p$, *we have* Flowtime$(A_{sp}(J)) \leqslant c \cdot$ Flowtime$(\mathcal{O}_p)$, *then A is s-speed c-competitive for all instances of jobs whose tasks go through phases with arbitrary non-decreasing sublinear speed-up functions.*

Proof. Consider an arbitrary instance $J = \{J_1, \ldots, J_n\}$. Consider an arbitrary small $\epsilon > 0$ and $\mathcal{O}_p$ a valid schedule of J on p processors such that Flowtime$(\mathcal{O}_p) \leqslant$ OPT$_p(J) + \epsilon$ (note that we do not need that an optimal schedule exists). Let J'' be the PS instance and $\mathcal{O}_p'$ the weakly streamlined schedule given by Lemmas 3.2 and 3.3 from J, A_{sp} and $\mathcal{O}_p$. Since $A_{sp}(J'') = A_{sp}(J)[J''/J]$, Flowtime$(A_{sp}(J)) =$ Flowtime$(A_{sp}(J''))$. But $A_{sp}(J'') \leqslant c \cdot$ Flowtime$(\mathcal{O}_p')$ by hypothesis. Thus, Flowtime$(A_{sp}(J)) \leqslant c \cdot$ Flowtime$(\mathcal{O}_p') \leqslant c \cdot$ Flowtime$(\mathcal{O}_p) \leqslant c$ OPT$_p(J) + c\epsilon$. Decreasing ϵ to zero completes the proof.

We shall from now on consider only PS instances and weakly streamlined optimal schedules with respect to which the algorithm is always late.

4 Competitiveness of α-scatterer Equi-schedulers

We consider a special class of algorithms which are a composition of a *job scheduler* and a *task scheduler*. The job scheduler allocates processing power to each alive jobs. The task scheduler is called within each job to divide among its alive tasks the processing power it has received from the job scheduler. This approach is a generalization of the algorithms developed in [12, 11] for minimizing the flowtime of jobs that consist in sets of independent tasks. According to the section above, we only need to consider PS instances. A non-clairvoyant scheduler cannot avoid wasting resources by not allocating all the processors to the alive tasks in a parallel phase. The job scheduler may waste resources if it allocates processor to a job in which no alive task is in a parallel phase. The task scheduler may also waste processors if it allocate some to a task in a sequential phase. Since the job and the task schedulers are non-clairvoyant, wasting resources is unavoidable at both levels. In [2], Edmonds shows essentially that when all jobs consist in a chain of tasks, the job scheduler EQUI, which divides evenly the processors among the alive jobs, does not waste more than half of the processors. The principle of his analysis is that the amount of parallel work scheduled by EQUI self-stabilizes around a positive fraction of the processing power and EQUI is thus $(2 + \epsilon)$-speed $O(1/\epsilon)$-competitive. In [11], it is shown that as soon as dependencies are introduced between the tasks, sequential works that could be optimally scheduled together can get scattered over large intervals of time and end up increasing drastically the flowtime by the game of the dependencies. We show here that if the task scheduler does not scatter too much the sequential work, then it does not waste too much resources, and the composition of both EQUI and the task scheduler ends up not wasting more than half of the resources as well, *i.e.*, is $(2 + \epsilon)$-speed $O(1/\epsilon)$-competitive whatever how complicated the precedence constraints are! We refer as EQUI-*scheduler* any composition of the job scheduler EQUI with a task scheduler.

4.1 α-scatterer task schedulers.
We say that a task scheduler does not scatter too much the sequential work if the sum over time of the proportion of the processors it receives from the job scheduler that ends up on sequential phases, can be bounded by some constant times the largest amount of sequential work along a chain in the corresponding job, seq(J_i).

Formally, consider a task scheduler A and assume it receives from the job scheduler ρ_i^t processors at each time t to schedule a PS job J_i, released at time 0.

DEFINITION 4.1. (WASTE) *We denote by ω_i^t the amount of processors which are not alloted to tasks in a parallel phase at time t in job J_i: $\omega_i^t = \rho_i^t - \pi_i^t$, where $\pi_i^t = \sum_{j \in Par_i^t} \rho_{ij}^t$ with $Par_i^t = \{j : task\ J_{ij}$ is alive and in a parallel phase at time $t\}$.*

We denote by waste(A, ρ, J_i) *the sum over time of the proportion of the processors that are not allotted by A to an alive task in a parallel phase:* waste$(A, \rho, J_i) = \int_0^\infty \frac{\omega_i^t}{\rho_i^t} dt$.

DEFINITION 4.2. (α-SCATTERER) *A task scheduler A is an α-scatterer if for any piecewise constant function $\rho : t \mapsto \rho^t$, and for any PS job J released at time 0:* waste$(A, \rho, J) \leqslant \alpha \cdot$ seq(J).[2] α *is referred as the* scattering coefficient *of the task scheduler A.*

Intuitively, the more sequential phases are scheduled at the same time, the better, because each of them has to share the processors with others and then contributes less to the waste of A. A task scheduler is thus inefficient if it schedules only one sequential phase at a time, *i.e.*, if it *scatters* the sequential phases far apart from each other. An α-scatterer will *under no circumstances* scatter the sequential phases of the tasks in such a way that it ends up giving away more than a proportion α times the largest chain of sequential work in a job, of the processors it receives from the job scheduler.

PROPOSITION 4.1. (OPTIMAL SCATTERER) EQUI, *as a task scheduler, is a $\frac{\kappa(J)+1}{2}$-scatterer.[3] Furthermore, for any task scheduler A, any piecewise constant function ρ and any integer k, there exists a job J such that $\kappa(J) = k$ and* waste$(A, \rho, J) \geqslant \frac{k+1}{2} \cdot$ seq(J).

Proof. Take an arbitrary piecewise constant function ρ and a job J released at time 0, consisting in a DAG of tasks whose the size of the largest anti-chain is k. Let ν_t be the total number of alive tasks, and λ_t be the number of alive tasks in a sequential phase, at time t. The tasks scheduler EQUI evenly divides the ρ^t processors among the ν_t alive tasks; the tasks in a sequential phase receive then a total of $\frac{\lambda_t}{\nu_t} \cdot \rho^t$ processors. It follows that: waste$(EQUI, \rho, J) = \int_0^\infty (\frac{\lambda_t}{\nu_t} \cdot \rho^t) \cdot \frac{1}{\rho^t} dt = \int_0^\infty \frac{\lambda_t}{\nu_t} dt$.

According to Dilworth's theorem, since the size of the largest antichain in the DAG representing J is k, one can partition the tasks in J into k disjoint chains $\xi_1, \ldots, \xi_k$, *i.e.*, such that in each ξ_j all the tasks have to be scheduled in a fixed total order given by $\prec^*$, the transitive closure of $\prec$. Since EQUI schedules each task

[2]Recall that seq(J) is the largest total amount of sequential work along a chain of tasks in job J.

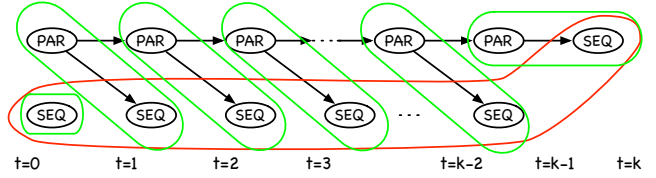[3]Recall that $\kappa(J)$ is the maximum number of independent tasks in job J.

Figure 1: The evil comb – an antichain of size k in red and a partition into k chains in green.

as soon as it is available, we can *w.l.o.g.* assume that ξ_1 has a task alive at any time until the completion of J.

Now, let $\mathbf{1}_{\xi_j}(t) = 1$ if and only if there is an alive task in a sequential phase in ξ_j at time t, and $\mathbf{1}_{\xi_j}(t) = 0$ otherwise (note that since ξ_j is a chain there cannot be two alive tasks in ξ_i at the same time t). Since the k chains cover all the tasks, we can rewrite: waste$(EQUI, \rho, J) = \int_0^\infty \frac{\sum_{j=1}^k \mathbf{1}_{\xi_j}(t)}{\nu_t} dt = \sum_{j=1}^k \int_0^\infty \frac{\mathbf{1}_{\xi_j}(t)}{\nu_t} dt \leqslant \int_0^\infty \mathbf{1}_{\xi_1}(t) dt + \sum_{j=2}^k \int_0^\infty \frac{\mathbf{1}_{\xi_j}(t)}{2} dt$, since any task in any chain ξ_j, $j \geqslant 2$, is always executed together with one other task in ξ_1 at least. It follows that: waste$(EQUI, \rho, J) \leqslant$ seq$(\xi_1) + \frac{k-1}{2} \max\{seq(\xi_2), \ldots,$ seq$(\xi_k)\} \leqslant \frac{k+1}{2} \cdot$ seq(J), by definition of seq(J). EQUI is thus a $\frac{\kappa(J)+1}{2}$-scatterer for any job J.

Optimality. Consider an arbitrary task scheduler A receiving ρ^t processors from the job scheduler. Consider a job consisting in a binary comb (see Fig. 1) with $k-1$ leaves and an extra isolated node, in which internal nodes consist in a single parallel phase, and the leaves and the isolated node consist in a sequential phase. At any time, A has to schedule two tasks, one internal node task and one isolated node or leaf task. Since A is non-clairvoyant, we can set which is which afterwards, once A made its choices. At time 1, let ρ_1 and ρ_2, $\rho_1 \geqslant \rho_2$, the total processing power given by A to each of the two first tasks: we say that the task which received ρ_1 processing power was a sequential phase of work 1 and that the other task was a parallel phase of work ρ_2; this later parallel task will spawn the next two tasks. This process continues until time $k - 1$, where the parallel task spawns a unique sequential task of work 1. Finally, A wasted at least half of the processing power until time $k - 1$ where it has to waste all it receives for one extra time unit, since it receives no parallel work. The total waste is thus: waste$(A, \rho, J) \geqslant \frac{k-1}{2} + 1 = \frac{k+1}{2}$.

4.2 Competitiveness of α-scatterer Equi-schedulers.

The main result of this section is that as soon as one uses an EQUI-scheduler based on a α-scatterer task scheduler, the competitive ratio is

bounded by a constant *independent* of the number of requests, namely $O(\alpha/\epsilon)$, as soon as it receives $2 + \epsilon$ times more resources than the optimal it is compared to. If we see an s-speed $O(1)$-competitive algorithm as a system that can handle a charge of $1/s$ without diverging (*i.e.*, which starts being overwhelmed at a charge $1/s$), our result means that introducing precedence constraints does not decrease the maximum load the system can handle. One could believe a priori that the precedence constraints and the interaction of the different jobs may conduct the scheduler to waste an higher proportion of resources on sequential work that would propagate over time and thus lower the maximum load the system can handle; it turns out that using an α-scheduler guarantees that the scattering of the sequential work due to the precedence constraints remains local to each job and does not interfere with other jobs in a dramatic way.

The key is to show that the effect of the precedence constraints on an α-scatterer is essentially to stretch the sequential work by a factor at most α while the parallel work remains unchanged. It follows that the self-stabilization process exhibited by [2] still occurs and the parallel work ends up receiving a positive fraction of the processing power which guarantees that the system will not diverge, as soon as it receives at least $(2 + \epsilon)$ times more resources than the optimal it is compared to.

Now, consider an arbitrary EQUI-scheduler on sp processors, EQUI∘A, based on an α-scatterer task scheduler A. Consider a PS instance $J = \{J_1, \ldots, J_n\}$, together with a weakly streamlined schedule $\mathcal{O}_p$ given by Lemma 3.3 such that EQUI∘$A_{sp}(J)$ is always late with respect to $\mathcal{O}_p$. We show that Flowtime(EQUI∘$A_{sp}(J)$) $\leqslant O(\frac{\alpha}{\epsilon})$ Flowtime($\mathcal{O}_p$) as soon as $s = 2 + \epsilon$ for some $\epsilon > 0$. Scaling the parallel work by $1/p$ in J, we assume *w.l.o.g.* that $p = 1$, and remove p from the notations.

Linearization of the tasks within a job. This step is the key step that allows to get rid of the precedence constraints within each job and to count the progress of the job with only two variables: one that counts for the progress of the total sequential work and one that counts for the progress of the total parallel work.

Before we proceed we define the following operation. Consider a job J_i and a time interval $[t, t']$ such that all tasks in J_i remain in the same phase and the amount of processors allotted by EQUI∘$A_s(J)$ to each of them remains constant during $[t, t']$. The operation *push-PAR-to-the-right* consists in rescheduling the parallel work phases within the time interval in a Round Robin manner as far as possible to the right (see Fig. 2). Note that we obtain a valid schedule since the sequential
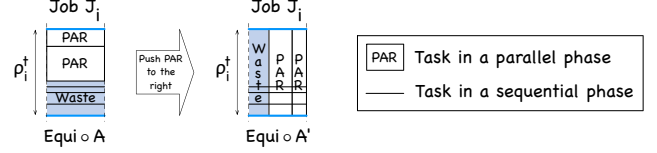

Figure 2: Push the parallel work to right within the area allotted to a job in EQUI∘$A_s(J)$.

phases are unaffected by this change.

Furthermore, this operation is *waste-conservative*. Consider the waste of A for job J_i given the amount of processor ρ_i^t allotted by the job scheduler EQUI$_s$: waste(EQUI∘$A_s(J), J_i$) $= \int_0^\infty (1 - \pi_i^t/\rho_i^t) dt$, where π_i^t denotes the amount of processors given by A to the tasks of J_i in parallel phases at time t. The waste may only change during the time interval concerned by the operation. If T denotes the length of the time interval, ρ the amount of processors J_i receives from EQUI$_s$, and π the amount of these processors that are allotted to J_i's alive tasks in a parallel phase, the contribution to the total waste for job J_i of this time interval is $(1 - \pi/\rho) \cdot T$ in the former schedule and $1 \cdot (T - \pi T/\rho)$ in the later. It follows that the waste in both schedule for J_i are identical.

DEFINITION 4.3. (LINEARIZATION OF A SCHEDULE) *Let $\delta > 0$ be some arbitrary small constant. The δ-linearization of the schedule EQUI∘$A_s(J)$ with respect to $\mathcal{O}$ is the schedule EQUI∘$A'_s(J)$ on s processors obtained as follows. We apply the push-PAR-to-the-right operation to EQUI∘$A_s(J)$ within each maximal time interval of length $\leqslant \delta$, such that:*

1. *the amount of processors allotted to each alive task does not change in EQUI∘$A_s(J)$ and neither in $\mathcal{O}$;*

2. *each alive task remains in the same phase in both schedules; and*

3. *if $\mathcal{O}$ schedules a task J_{ij} and if this task is scheduled in EQUI∘$A_s(J)$ as well during this period of time, this task is the last scheduled in the Round Robin order applied by the push-PAR-to-the-right operation.*

By construction, the linearized schedule EQUI∘$A'(J)$ is a valid schedule of J, verifying:

- EQUI∘$A'_s(J)$ is always late with respect to $\mathcal{O}$;

- Flowtime(EQUI∘$A'_s(J)$) $\leqslant$ Flowtime(EQUI∘$A_s(J)$) $\leqslant$ Flowtime(EQUI∘$A'_s(J)$) $+ n\delta$ (since the last sequential phase of each job may be released and thus completed δ units of time earlier in the new schedule, but not before);

- for all job J_i, $\mathsf{waste}(\textsc{Equi} \circ A'_s(J), \rho_i, J_i) = \mathsf{waste}(\textsc{Equi} \circ A_s(J), \rho_i, J_i)$.

We are now left with showing that:

$$\mathsf{Flowtime}(\textsc{Equi} \circ A'_s(J)) \leqslant O(\alpha/\epsilon) \, \mathsf{Flowtime}(\mathcal{O}),$$

decreasing δ to zero will conclude the result.

Parametrization. Thanks to the δ-linearization, we are now back to the setting of [2] where the execution of each job consists in an alternation of executions of parallel and sequential work.

Let, at time t, in $\textsc{Equi} \circ A'_s(J)$:

- n_t, be the number of alive jobs;

- ℓ_t, be the number of alive jobs in which no parallel work is scheduled; and

- m_t, be the number of alive jobs in which some parallel work is scheduled.

By definition, $\mathsf{Flowtime}(\textsc{Equi} \circ A'_s(J)) = \int_0^\infty n_t \, dt = \int_0^\infty (\ell_t + m_t) \, dt$.

Bounding $\int_0^\infty \ell_t \, dt$. Note that as opposed to [2], $\int_0^\infty \ell_t \, dt$ is not directly related to the total sequential work in the instance. But, since ℓ_t counts, for each job, the periods of time in which the job wastes all the resources it receives in $\textsc{Equi} \circ A'_s(J)$, we have: $\int_0^\infty \ell_t \, dt = \sum_i \mathsf{waste}(\textsc{Equi} \circ A'_s(J), J_i)$. But, $\textsc{Equi} \circ A'_s(J)$ was obtained from $\textsc{Equi} \circ A_s(J)$ by a waste-conservative process. So, $\int_0^\infty \ell_t \, dt = \sum_i \mathsf{waste}(\textsc{Equi} \circ A_s(J), J_i) \leqslant \sum_i \alpha \, \mathsf{seq}(J_i)$, since A is an α-scatterer. But, the optimal flowtime verifies: $\mathrm{OPT}(J) \geqslant \sum_i \mathsf{seq}(J_i)$, so, $\int_0^\infty \ell_t \, dt \leqslant \alpha \, \mathrm{OPT}(J)$.

Bounding $\int_0^\infty m_t \, dt$. Thanks to the δ-linearization, the proof now follows essentially the steps of [2]. More precisely, we will conduct the same calculations as in [2] over variables that have the same names. The key to Edmonds's result in [2] is that when job consists in a single chain of tasks, the parallel work tends to self-stabilize as follows: at any time t, the m_t jobs executing some parallel work receive each $s/(m_t + \ell_t)$ processors and thus the total parallel work progresses at a rate $s m_t/(m_t + \ell_t)$, while the weakly streamlined schedule $\mathcal{O}$ executes the parallel work on one processor at a rate $\leqslant 1$. Thus in an ideal steady state, when $s m_t/(m_t + \ell_t) < 1$, the parallel work accumulates in $\textsc{Equi}$ with respect to $\mathcal{O}$, and the proportion $s m_t/(m_t + \ell_t)$ tends to increase; and when $s m_t/(m_t + \ell_t) > 1$, $\textsc{Equi}$ progresses faster than $\mathcal{O}$ and $s m_t/(m_t + \ell_t)$ tends to decreases. It follows that in an ideal steady state, m_t self-stabilizes around the value $s m_t/(m_t + \ell_t) \sim 1$, *i.e.*, $m_t \sim \frac{\ell_t}{s-1}$. It turns out that if $s < 2$, the steady state cannot be reached fast enough,

but as soon as $s = 2 + \epsilon$, Edmonds shows with the help of a potential function that $\int_0^\infty m_t \, dt$ can be bounded from above by some combination of the delay imposed to $\textsc{Equi}$ by the sequential phases, $\int_0^\infty \ell_t \, dt$, and by the time spent by the optimum on parallel phases. The following consists in adapting this scheme to our setting. We propose as well some minor simplifications on Edmonds's proof.

Thanks to the δ-linearization, we are able to define variables similar to the case where jobs consist in a single chain of tasks. For $T \leqslant t$, let m_t^T be the number of alive jobs executing some parallel work dw between t and $t + dt$ in $\textsc{Equi} \circ A'_s(J)$, that has been done before time T in $\mathcal{O}$. This variable counts how late $\textsc{Equi} \circ A'_s(J)$ is with respect to $\mathcal{O}$.

Let M_t be the number of alive jobs at time t in $\mathcal{O}$ such that at least one of its alive tasks is in a parallel phase. Note that as opposed to [2], M_t does not count the number of parallel phases scheduled at time t and can take values larger than 1, since $\mathcal{O}$ is only weakly streamlined and there may be unscheduled alive parallel phases in addition to the task scheduled at time t in $\mathcal{O}$. Let ϕ be the function that associates to each time T where some parallel work dw is scheduled in $\mathcal{O}$ during a time interval $[T, T + dT)$, the time $\phi(T)$ ($\geqslant T$) at which this exact same parallel work dw is executed in $\textsc{Equi} \circ A'_s(J)$, during time interval $[\phi(T), \phi(T + dT))$.

LEMMA 4.1. *As in [2], the variables verify the following relationships: for all time $T \leqslant t$,*

1. *$m_T^T = m_T$,*

2. *$m_t^T \leqslant n_T$,*

3. *If $M_T > 0$:*
   - *$m_t^{T+dT} = m_t^T$ when $t \notin [\phi(T), \phi(T + dT)]$ and*
   - *$m_t^{T+dT} = m_t^T + 1$ when $t \in [\phi(T), \phi(T + dT)]$;*

4. *If $M_T = 0$, then: $m_t^T = m_t$, for all $t \geqslant T$.*

Proof. The first point is because $\textsc{Equi} \circ A'_s(J)$ is late with respect to $\mathcal{O}$. The second point is because each *job* is released in $\mathcal{O}$ at the same time as in $\textsc{Equi} \circ A'_s(J)$ so that if a job is counted in m_t^T, it is alive at time T in $\textsc{Equi} \circ A'_s(J)$ as well and is counted in n_T. The last points hold by definition of m_t^T and $\phi(T)$.

Following [2], let $F_T = \int_0^T m_t \, dt$ be the contribution to the cost of $\textsc{Equi} \circ A'_s(J)$ due to parallel work up to time T, and $\hat{F}_T = \int_T^\infty f_t(m_t^T, \ell_t) \, dt$, with $f_t(m, \ell) = \frac{m^2 - \ell^2}{n_t}$, a potential function that counts, as we will see later, the total cost induced by the accumulated tardiness of $\textsc{Equi} \circ A'_s(J)$ at time T.

The proof consists in demonstrating that $F_T + \hat{F}_T$ is a continuous function whose derivative is bounded by a function of ℓ_t and M_t only. It follows that its limit $F_\infty + \hat{F}_\infty = F_\infty = \int_0^\infty m_t \, dt$ is bounded in terms of known lower bounds on OPT.

LEMMA 4.2. *If $s > 2$, for all time T,*

$$F_T + \hat{F}_T \leqslant \int_0^T \frac{(s+2)\ell_t + M_t}{s-2} \, dt.$$

Proof. At time 0, EQUI $\circ A'_s(J)$ is not late and that it is also the case at the end of the schedule; thus, $F_0 = \hat{F}_0 = \hat{F}_\infty = 0$.

Continuity. F_T is clearly continuous. $\hat{F}_T$ is continuous as well, even at the dates where a new phase ends or begins in $\mathcal{O}$ or in EQUI $\circ A'_s(J)$. Indeed, since m_t^T and ℓ_t are uniformly bounded, the variation induced by the variation of the bounds of integration are continuous. Now, m_t^T and m_t^{T+dT} may differ only by 1 only on an interval of infinitesimal size corresponding to the infinitesimal time interval during which EQUI $\circ A'_s(J)$ executes the infinitesimal parallel work done in $\mathcal{O}$ between T and $T + dT$, if any. It follows that $F_T + \hat{F}_T$ is continuous.

Bounding the derivative. Furthermore, $F_T + \hat{F}_T$ is differentiable except on a finite number of dates corresponding to the phase changes in $\mathcal{O}$. Clearly, $dF_T = m_T dT$. Now,

$$d\hat{F}_T = \underbrace{\int_{T+dT}^\infty f_t(m_t^{T+dT}, \ell_t) - f_t(m_t^T, \ell_t) \, dt}_{(1)}$$
$$\underbrace{- \int_T^{T+dT} f_t(m_T, \ell_t) \, dt}_{(2)}.$$

We now bound (1) and (2) independently. (1) corresponds to the increase of the cost of EQUI $\circ A'_s(J)$ due to jobs falling behind. If $M_T = 0$, no job falls behind, and (1) = 0. If $M_T > 0$:

$$(1) = \int_{\phi(T)}^{\phi(T+dT)} f_t(m_t^T + 1, \ell_t) - f_t(m_t^T, \ell_t) \, dt$$
$$= \int_{\phi(T)}^{\phi(T+dT)} \frac{s(2m_t^T + 1)}{(s-2)n_t} dt.$$

But the length of the interval $[\phi(T), \phi(T+dT)]$ is exactly $\frac{n_{\phi(T)} dT}{s}$ since the number of processors alloted to each job at time $\phi(T)$ is $\frac{s}{n_{\phi(T)}}$ and if a parallel phase is executed in a job in EQUI $\circ A'_s(J)$, this phase receives all the processors allotted to the job (thanks to the

δ-linearization), thus:

$$(1) = \frac{s(2m_t^T + 1)}{(s-2)n_{\phi(T)}} \frac{n_{\phi(T)} dT}{s} = \frac{(2m_t^T + 1)}{(s-2)} dT$$
$$\leqslant \frac{(2n_T + 1)}{(s-2)} dT,$$

since $m_{\phi(T)}^T$ is at most n_T. We can thus bound (1) by $\frac{2(m_T + \ell_T) + M_T}{(s-2)}$.

(2) corresponds to the amount of tardiness that EQUI $\circ A'_s(J)$ is making up for:

$$(2) = -f_T(m_T^T, \ell_T) dT = -f_T(m_T, \ell_T) dT$$
$$= -\frac{s(m_T^T + \ell_T)(m_T^T - \ell_T)}{(s-2)n_T} dT = -\frac{s(m_T - \ell_T)}{(s-2)} dT.$$

It follows that:

$$d(F_T + \hat{F}_T) \leqslant \left(m_T + \frac{2(m_T + \ell_T) + M_T}{(s-2)} - \frac{s(m_T - \ell_T)}{(s-2)} \right) dT$$
$$= \frac{(s+2)\ell_T + M_T}{s-2} dT.$$

Since $F_0 = \hat{F}_0 = 0$, we conclude that $F_T + \hat{F}_T \leqslant \int_0^T \frac{(s+2)\ell_t + M_t}{s-2} dt$.

We can now conclude with the proof of our main theorem 4.1.

THEOREM 4.1. *If A is an α-scatterer task scheduler, EQUI$\circ A$ is an $(2 + \epsilon)$-speed $O(\alpha/\epsilon)$-competitive algorithm for the non-clairvoyant scheduling problem with precedence constraint.*

Proof. For $s = 2 + \epsilon$ and arbitrarily small $\delta > 0$,

$$\mathsf{Flowtime}(\text{EQUI} \circ A_s(J))$$
$$\leqslant \mathsf{Flowtime}(\text{EQUI} \circ A'_s(J)) + n\delta$$
$$= n\delta + \int_0^\infty \ell_t + m_t \, dt$$
$$\leqslant n\delta + \int_0^\infty \ell_t + \frac{(s+2)\ell_t + M_t}{s-2} dt,$$

by Lemma 4.2 and since $\hat{F}_\infty = 0$. But $\mathsf{Flowtime}(\mathcal{O}) \geqslant \int_0^\infty M_t \, dt$, and $\int_0^\infty \ell_t \, dt \leqslant \alpha \sum_i \mathsf{seq}(J_i) \leqslant \alpha \, \mathsf{Flowtime}(\mathcal{O})$ since A is α-scatterer. Furthermore, by definition of $\mathcal{O}$, $\mathsf{Flowtime}(\mathcal{O}) \leqslant \text{OPT}(J) + \eta$ for some arbitrarily small $\eta > 0$. It follows that

$$\mathsf{Flowtime}(\text{EQUI} \circ A_s(J)) \leqslant \tfrac{2s\alpha + 1}{s-2}(\text{OPT}(J) + \eta) + n\delta.$$

Taking δ and η to 0, yields the claimed competitive ratio: $\mathsf{Flowtime}(\text{EQUI} \circ A_{2+\epsilon}(J)) \leqslant (2\alpha + \frac{4\alpha+1}{\epsilon}) \cdot \text{OPT}(J)$.

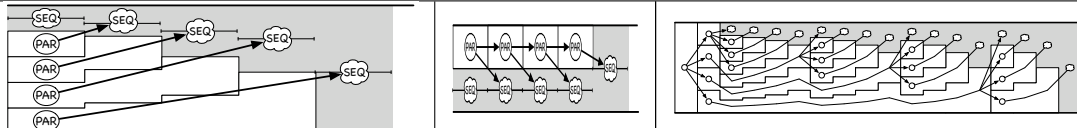Combining with Proposition 4.1, we obtain our main result, Theorem 1.1.

Precedence DAG	$\leqslant \kappa$ independent chains	IN-forest with $\leqslant \kappa$ leaves	OUT-forest with $\leqslant \kappa$ leaves	d-regular complete OUT-tree with $\leqslant \kappa$ leaves	d-regular Series-Parallel-DAG with $\leqslant \kappa$ tasks
Upper bounds	EQUI is a H_κ-scatterer		EQUI is a $\frac{\kappa+1}{2}$-scatterer and one can compute the exact scattering coefficient of EQUI for any OUT-tree	EQUI is a $\Theta(\kappa \frac{\ln\ln\kappa}{\ln\kappa})$-scatterer and the algorithm SPLIT[a] is a $\kappa^{\frac{\ln\ln d}{\ln d}}$-scatterer ⎯⎯⎯⎯⎯⎯⎯ [a]SPLIT is the task scheduler that divides recursively and evenly between the children the processing power given to their ancestor at the time the ancestor spawns the children.	
Lower bounds for Equi-schedulers	no EQUI-scheduler is s-speed α-competitive for $\alpha < \frac{\log_s(\kappa)+1}{2}$		no α-scatterer for $\alpha < \frac{\kappa+1}{2}$		
Lower bounds for any non-clairvoyant algorithm	no s-speed c-competitive for $c < \frac{\ln\kappa}{2\ln\ln\kappa}$ and $s = o(\frac{\ln\kappa}{\ln\ln\kappa})$ (from [11])				
Examples of bad instances					

Table 1: Upper and lower bounds on classic precedence structures.

5 Concluding remark

Our analysis relies on the notion of α-scatterer task scheduler. This tool allows to analyze conveniently the competitiveness of non-clairvoyant algorithms in Edmonds's job model without going through the whole process of the analysis. Table 1 sums up the upper and lower bounds obtained by our method on classic precedence structures which could not be included due to space constraints. Note that it is unclear to us if the scattering coefficient is the right concept nor if it is a graph-related notion, although it is a simple powerful tool to deal with precedences in the non-clairvoyant setting.

Moreover, it seems that Edmonds's analysis of EQUI, based on a potential function, might have a much wider extend than simply the field of non-clairvoyant algorithms since once the variables m_t^T and ℓ_t are defined, the underlying scheduling problem vanishes. We believe it might be worth trying to understand what is the exact problem it solves in order to exploit its whole potential.

References

[1] Yossi Azar and Leah Epstein. On-line scheduling with precedence constraints. In *Scandinavian Workshop on Algorithm Theory*, pages 164–174, 2000.

[2] J. Edmonds. Scheduling in the dark. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 179–188, New York, NY, USA, 1999. ACM Press.

[3] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. *J. Scheduling*, 6(3):231–250, 2003.

[4] J. Edmonds, S. Datta, and P. Dymond. TCP is competitive against a limited adversary. In *Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 174–183, 2003.

[5] J. Edmonds and K. Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.

[6] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal online scheduling of parallel jobs with dependencies. *J. of Combinatorial Optimization*, 1:393–411, 1998.

[7] Yuxiong He, Wen Jing Hsu, and Charles E. Leiserson. Provably efficient online non-clairvoyant adaptive scheduling. In *Proc. of IEEE Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.

[8] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. of the ACM*, 47:214–221, 2000.

[9] Keqin Li. Average-case performance analysis of scheduling random parallel tasks with precedence constraints on mesh connected multicomputer systems. *J. Parallel Distrib. Comput.*, 66(8):1090–1102, 2006.

[10] R. Motwani, S. Philipps, and E. Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130:17–47, 1994.

[11] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *Proc. of 15th European Symposium on Algorithms (ESA)*, volume LNCS 4698, pages 741–753, 2007.

[12] J. Robert and N. Schabanel. Pull-based data broadcast with dependencies: Be fair to users, not to items. In *Proc. of 18th Symp. on Discrete Algorithms (SODA)*, pages 238–247, 2007.