1 A Deterministic Scheduling Algorithm

In this section we present a simple deterministic scheduling algorithm that does not use preemption. We show that this algorithm is 2-competitive with OPT.

First we note that without loss of generality we may consider TAPs where the cost ratio $\pi(\tau)/\sigma(\tau)$ for any task τ is in $[1, p]$; if $\pi(\tau)/\sigma(\tau) < 1$, i.e. the parallel implementation has lower work than the serial implementation, then the scheduler clearly should never use the serial implementation of this algorithm, so we can replace the serial implementation with the parallel implementation and hence get cost ratio 1, similarly, if $\pi(\tau)/\sigma(\tau) > p$ then the scheduler should never run the parallel task and we can replace the parallel implementation of the task with the serial implementation to get cost ratio p.

We say that a time is a ***verge*** time for our algorithm if at this time no processors are performing tasks and there is at least one ready task.

We propose Algorithm 1, which we call ***GR*** (GR is short for "greedy"), as a scheduling algorithm.

Algorithm 1 GR

while True **do**
  if verge time **then**
    do what OPT would do in the case where no tasks arrive later than this verge time in the TAP

GR is an incredibly simple algorithm, although there is some hidden complexity in it: the algorithm is consulting an oracle to know what OPT would do, albeit in the relatively simple special case of where all the tasks arrive at the same time. In Corollary **??** we exhibit an oracle that yields a schedule that achieves the same awake time as OPT for TAPs where all the tasks arrive at a single time, hence showing that GR actually can be computed. First we analyze the competitive ratio of GR assuming the existence of such an oracle.

Consider a TAP $\mathcal{T}$. Let ℓ be the number of verge times for $\mathcal{T}$; note that $\ell \leq n$ which in particular is finite. Let t_i be the i-th time that is a verge time, let q_i be the number of ready tasks for GR at time t_i. Let $T^{ALG}(q_1, \ldots, q_{\ell'})$ denote the awake time of a scheduling algorithm ALG on the truncation of the TAP $\mathcal{T}$ that only consists of tasks arriving at times before $t_{\ell'}$.

By construction we have

$$T^{OPT}(q) = T^{GR}(q). \tag{1}$$

We remark GR "locally" schedules optimally, which is why we refer to GR as "greedy".

An ALG-gap is an interval of time of non-zero length where for all times in the interior of the interval ALG has completed every task that has arrived thus far. Additionally for an interval to be an ALG-gap the interval must contain no other intervals which are also ALG-gaps (i.e. it is a "maximal" interval satisfying our conditions). We say that a TAP is ***ALG-gap-free*** if it contains no ALG-gaps.

Now we prove an important property of OPT.

Claim 1. *If there is a scheduling algorithm ALG that completes all tasks by time t_* then OPT finishes all tasks by time t_*.*

Proof. Say that ALG completes all tasks by time t_*. Let $t_0 < t_*$ be the most recent time that OPT has completed all tasks that arrive before time t_0. If OPT has not finished all tasks by time t_* then it was acting sub-optimally, as it could steal the strategy that ALG used on $[t_0, t_*]$ to achieve lower awake time. In particular, for any tasks that arrive in $[t_0, t_*]$ OPT could schedule them as ALG schedules them. We remark that OPT should not steal all of ALG. □

Note that as an immediate consequence of Claim 1 we have that any ALG-gap is a subset of an OPT-gap.

Decomposing TAPs into gap-free subsets of the TAP is very useful. Part of the reason for this is the following fact:

Claim 2. *If an algorithm ALG achieves competitive ratio r on ALG-gap-free TAPs, then ALG achieves competitive ratio r on arbitrary TAPs.*

Proof. We partition the tasks based on arrival time, splitting the tasks on the ALG-gaps. That is, we split the tasks into groups so that two tasks τ_i, τ_j are in the same group if and only if there are no gaps in between the arrival times of τ_i and τ_j. We can define an interval of time I_i for each of these ALG-gap-free subsets of the TAP, where I_i is defined so that all tasks in the i-th group start and finish at times contained in the interval I_i.

Let $T^{OPT}_{I_i}$ and $T^{ALG}_{I_i}$ denote the awake time of OPT and ALG on interval I_i. Because I_i is ALG-gap-free we have $T^{ALG} = \sum_i T^{ALG}_{I_i}$. Further, recall that by Claim 1 any ALG-gap is also an OPT-gap, so $T^{OPT} = \sum_i T^{OPT}_{I_i}$. Hence from our assumption that ALG is r-competitive on gap-free TAPs, such as the subset of the TAP on the interval I_i, we have $T^{ALG}_{I_i} \leq r \cdot T^{OPT}_{I_i}$ for all i. Summing we get $T^{ALG} \leq r \cdot T^{OPT}$, as desired.

□

By Claim 2, in order to bound GR's competitive ratio, it suffices to consider TAPs without GR-gaps. Note however that a TAP without GR-gaps could still have OPT-gaps.

We conclude our analysis of the competitive ratio of GR in Proposition 1 with an inductive argument on the number of OPT-gaps in the TAP. First we establish the base case for the argument: we consider GR's competitive ratio on a TAP without OPT-gaps.

Claim 3. *GR is 2-competitive on any OPT-gap-free TAP.*

Proof. Since the TAP is OPT-gap-free we must have

$$T^{OPT}(q_1, \ldots, q_\ell) \geq T^{SGR}(q_1, \ldots, q_{\ell-1}). \quad (2)$$

Because GR finishes all q_i tasks that arrive at time t_i by time t_{i+1} we can actually always decompose $T^{GR}(q_1, \ldots, q_\ell)$ as

$$T^{GR}(q_1, \ldots, q_\ell) = \sum_{i=1}^{\ell} T^{GR}(q_i). \quad (3)$$

By Equation (3), and Equation (1) we thus have

$$T^{GR}(q_1, \ldots, q_\ell) = T^{GR}(q_1, \ldots, q_{\ell-1}) + T^{OPT}(q_\ell). \quad (4)$$

Hence by Equation (2) and Equation (4) we have

$$\begin{aligned} T^{GR}(q_1, \ldots, q_\ell) &\leq T^{OPT}(q_1, \ldots, q_\ell) + T^{OPT}(q_\ell) \\ &\leq 2T^{OPT}(q_1, \ldots, q_\ell), \end{aligned}$$

as desired. □

Proposition 1. *GR is 2-competitive.*

Proof. The proof is by strong induction on the number of OPT-gaps. The base case of our induction is established in Claim 3, which says that if there are 0 OPT-gaps then GR is 2-competitive.

Consider a TAP that has more than 0 OPT gaps; say that its first OPT-gap starts at time t_*. Let j be the largest index such that verge time $t_j < t_*$.

Using our inductive hypothesis we have:

$$\begin{aligned} &T^{OPT}(q_1, \ldots, q_\ell) \\ &\geq T^{OPT}(q_1, \ldots, q_j) + T^{OPT}(q_{j+1}, \ldots, q_\ell) \\ &\geq \frac{1}{2}\left(T^{GR}(q_1, \ldots, q_j) + T^{GR}(q_{j+1}, \ldots, q_\ell)\right) \\ &= \frac{1}{2}T^{GR}(q_1, \ldots, q_\ell). \end{aligned}$$

□

Now we demonstrate the existence of an oracle computing OPT on TAPs where all tasks arrive at a single point in time. First, we need a lemma:

Lemma 1 (Cake frosting lemma)**.** *you can frost a cake*

Proof. Of course by Claim **??** $T^{OPT}(q) \leq T^{SGR}(q) = T(q)$. We now claim that OPT can do no better than this. This is obvious if $q \leq p$: if OPT schedules any task in serial then OPT has awake time at least 1, and if OPT schedules everything in parallel then OPT has awake time at least qk/p; for $q \leq p$ we have $T(q) = \min(1, qk/p)$, so OPT can do no better than SGR here. It is intuitively obvious that if there are $q > p$ tasks, then scheduling about $\lfloor q/p \rfloor$ tasks in serial to each processor like SGR does is the right strategy; proving this is somewhat intricate however.

Say OPT schedules x of the q tasks in serial, and schedules the remaining $q - x$ in parallel. Say that an optimal assignment of the tasks, under the constraint that x of the tasks are scheduled in serial and $q - x$ are scheduled in parallel, achieves awake time M.

We claim that there must exist an optimal assignment of tasks such that each processor is assigned either $\lfloor x/p \rfloor$ or $\lceil x/p \rceil$ serial tasks. To prove this, we start from some optimal assignment, and modify it in such a way as to make the configuration "closer" to our desired configuration. Let $n_\sigma(\rho_i)$ be the number of serial tasks scheduled on processor ρ_i.

To formalize a notion of "closeness" we define a potential function ϕ of the assignment S of tasks:

$$\phi(S) = \sum_i \min(|n_\sigma(\rho_i) - \lfloor x/p \rfloor|, |n_\sigma(\rho_i) - \lceil x/p \rceil|).$$

Note that $\phi(S)$ is non-negative. We desire a configuration with $\phi(S) = 0$. Consider a configuration of tasks achieving awake time M with $\phi(S) > 0$. We first apply the following procedure to the configuration: while the difference between the maximum work assigned to a processor and the minimum work assigned to a processor is more than 1 swap 1 unit of work from a processor with the maximum amount of work to a processor with the minimum amount of work. This swap cannot increase the range of works assigned to processors, and the process must eventually terminate. Now, while $\phi(S) > 0$ there must exists ρ_i, ρ_j with $n_\sigma(\rho_i) < \lfloor x/p \rfloor$ and $n_\sigma(\rho_j) > \lfloor x/p \rfloor$ or there exists ρ_i, ρ_j with $n_\sigma(\rho_i) < \lceil x/p \rceil$ and $n_\sigma(\rho_j) > \lceil x/p \rceil$. By construction the range of the works is at most 1; hence ρ_i has at least 1 unit of parallel work to have work within 1 of ρ_j despite having at least 2 fewer serial tasks than ρ_j. Then ρ_i gives this 1 unit of parallel work to ρ_j, and in exchange ρ_j gives ρ_i a serial task. This swapping operation decreases $\phi(S)$ by exactly 1, and does not change the amount of work assigned to each processor, which importantly means that the swap does not increase the range of

the works. Hence, we can repeat this swapping process to eventually achieve a configuration with awake time M where each processor has $\lfloor x/p \rfloor$ or $\lceil x/p \rceil$ serial tasks.

If $x \bmod p = 0$ then the awake time of OPT is clearly at least

$$x/p + (q-x)k/p. \tag{5}$$

Note that if x increases by p then (5) changes by $1-k<0$. That is, (5) is monotonically decreasing in x, and is thus minimized by setting $x = p\lfloor q/p \rfloor$ and hence getting $q - x = q \bmod p$. This gives awake time $\lfloor q/p \rfloor + (q \bmod p)k/p \geq T(q)$.

Now we consider the case where $x \bmod p \neq 0$. Here $(p-(x \bmod p))/k$ tasks can be added in serial without increasing the work at all. Thus the awake time is at least

$$\lceil x/p \rceil + \frac{k}{p}\left(\max\left(0, q - x - \frac{p-(x \bmod p)}{k}\right)\right). \tag{6}$$

Consider when the max in (6) yields 0. For this to happen we must have

$$q - x \leq \frac{p - x \bmod p}{k}.$$

As $p - x \bmod p \leq p$ we get the following bound on $q-x$:

$$q - x \leq p/k,$$

which must be met in order for the max to yield 0. Clearly as $q \geq x \geq q - p/k$, $\lfloor x/p \rfloor \geq \lfloor q/p \rfloor - 1$; we claim that this inequality is actually strict. Imagine that $\lfloor x/p \rfloor = \lfloor q/p \rfloor - 1$. Because the max expression yields 0, meaning that the parallel tasks add nothing to the awake time, if all processors only have $\lfloor q/p \rfloor - 1$ or $\lfloor q/p \rfloor < q/p$ serial tasks, then the total work is at most $\lfloor q/p \rfloor < q/p$ which is impossible: the total work in the system must be at least q and it can be distributed in the best case perfectly equally which makes q/p as a lower bound on the time to complete q tasks. Hence $\lfloor x/p \rfloor = \lfloor q/p \rfloor$. But then the awake time is at least $\lceil x/p \rceil = \lfloor q/p \rfloor + 1 \geq T(q)$.

Now we consider the case when the max in (6) yields some positive number. Note that if x increases by p (but x is still sufficiently small so that the max yields a positive number) then (6) changes by $1-k<0$. Further, if x increases by 1 (but x is still sufficiently small so that the max yields a positive number) without $\lceil x/p \rceil$ changing, then (6) changes by $(k/p)(1/k - 1) < 0$. That is, (6) is monotonically decreasing in x. Hence we still have that the awake time is at least $T(q)$.

We have considered all cases, and shown that no matter what choice of x OPT makes, and no matter how OPT schedules given that choice of x, OPT must incur awake time at least $T(q)$, as desired. □

Algorithm 2 OPT Oracle

```
minAwakeTime ← ∞
for I ∈ {0,1}^n do
    if I_i = 1 then
        Schedule task i in parallel when it arrives
    else
        Schedule task i in serial when it arrives
    In particular, first schedule serial tasks to minimize awake time
    And then sprinkle parallel tasks on top
    if awakeTime_I ≤ minAwakeTime then
        minAwakeTime ← awakeTime_I
```

Corollary 1. *OPT Oracle is an oracle for OPT.*

Proof. □

References

[1] Benjamin Berg, Mor Harchol-Balter, Benjamin Moseley, Weina Wang, and Justin Whitehouse. Optimal resource allocation for elastic and inelastic jobs. *SPAA*, pages 75–87, 07 2020.

[2] Davide Bilò, Luciano Gualà, Stefano Leucci, Guido Proietti, and Giacomo Scornavacca. Cutting bamboo down to size. *FUN*, 2020.

[3] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1242–1253, 01 2012.

[4] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Eric Torng. Competitively scheduling tasks with intermediate parallelizability. *ACM Transactions on Parallel Computing*, 3:1–19, 07 2016.