1 Introduction

We begin by discussing Dual Fitting and its application to an online scheduling algorithm for minimizing flow time on unrelated Machines. It will be shown that by allowing machines a \((1 + \epsilon)\) speedup, we can obtain a constant competitive ratio. Next, we look into multiplicative weights and experts learning.

2 Dual Fitting and Scheduling on Unrelated Machines

2.1 Preliminaries and Recap of Lecture Four

We begin by recalling the flow time minimization problem on unrelated machines from last lecture: Given a set of machines and a set of jobs, define: \(r_j \in \mathbb{Z}\), to be the release time of job \(j\), \(p_{i,j} \in \mathbb{Z}\), is the run time of job \(j\) on machine \(i\), \(C_j\) is the time at which job \(j\) is completed and the flow time of \(j\) is \(F_j = C_j - r_j\). Our goal is to find a schedule \(S\), s.t. \(\sum_{j \in J} F_j\) is minimized.

In the online setting, jobs arrive over time and no assumptions can be made about \(p_{i,j}\) or \(r_j\) before job \(j\) arrives. In addition, we make two additional requirements from the algorithm. Once a job is assigned to a machine, it cannot be migrated to another machine. Jobs may be paused and continued at a later date in the future. This is called preemption. In fact our algorithm will assign a job to machine immediately upon arrival. Such algorithms are called immediate dispatch algorithms.

We first observe that the algorithm only needs to specify which machine to assign a job. This is because given an assignment of jobs to a machine, we simply run the SRPT (Shortest Remaining Processing Time) algorithm on that machine, which is optimum (1-competitive).

Consider the following algorithm for a single machine system (Shortest Remaining Processing Time) which at any time works on the job with the least remaining processing (note that it only preempts a job upon the arrival on another job).

**Theorem 1** The SRPT algorithm is optimal for the online setting with only one machine.

**Proof:** We prove this by observing that if there are two jobs, \(j\) and \(j^*\), where \(j\) is executed at some slot before \(j^*\), while the remaining times \(p_{1,j^*} < p_{1,j}\). Then by executing \(j\) instead, and executing \(j^*\) in the time slot of \(j\) can only decrease the total flow time. By induction, over time slots, this implies the result. \(\square\)

**Theorem 2** There is no online algorithm with bounded competitive ratio.

**Proof:** Consider an instance \(I\) where we have three machines, and where all jobs run in unit time. We allow just two types of jobs, Type I jobs, which can only run on Machines 1 and 2, and Type II jobs which can only run on Machines 2 or 3. Our algorithm first begins by receiving 2 jobs of type I, and 2 jobs of type II at every unit of time \(t \in \{1, \ldots, T\}\). At time \(T+1\), there are at least \(T\) tasks waiting to be completed, at least half of which (wlog) are of task I. After time \(T\), for \(L\) time
steps, our algorithm receives 2 tasks of type I. Therefore, we have $T/2$ tasks backlogged for $L$ time steps, giving us the total flow of online $\geq LT/2$.

Let us now consider an offline algorithm. An offline algorithm would have pushed back all tasks of type II, executing only one at a time, while instead executing both tasks of type I as they came in. Therefore, once the second batch of type I tasks began coming in, they could be executed without any additional type I tasks in the backlog. Therefore all flow comes from type II tasks in this case, giving us $T^2$ flow, plus $L$ flow for the jobs arriving after time $T$.

As $L$ can be made arbitrarily large relative to $T$ ($L = T^2$ suffices), the algorithm has at least $\Omega(T)$ competitive ratio.

To get around this pessimistic guarantee, we consider the resource augmentation setting, where the online algorithm is allowed slightly more resources. This setting is often very useful in scheduling and other contexts, and allows one to obtain interesting algorithms despite the (sometimes) overly pessimistic nature of competitive analysis. In our setting, we allow the algorithm to use a $(1 + \epsilon)$-speed processor, but we compare its performance to an optimum offline solution on a speed 1 processor. If an algorithm is $\beta(\epsilon)$ competitive, we call it a $(1 + \epsilon)$-speed, $\beta$-competitive algorithm.

### 2.2 1+\epsilon Speed Machines and a Competitive Algorithm

We begin by stating a Theorem, which we shall later prove.

**Theorem 3** There is a $(1+\epsilon)$ speed $O(1/\epsilon)$ competitive algorithm for the total flow time.

We now describe the algorithm. Recall that on any machine, we can assume that the jobs assigned to it are executed in SRPT order. Consider a job $j$ at arrives at time $t$. For each machine $i$, consider the jobs in SRPT order.

We will define a cost function $Q$ that measures the increase in total flow of all jobs if $j$ is assigned to machine $i$. Let $Q_{i,j} = k_ip_{i,j} + p_{i,j} + \sum_{j'|p_{i,j'} < p_{i,j}} p_{i,j'}$. Here $k_i$ is the number of jobs $j'$ with remaining processing $p(i,j') \geq p_{i,j}$. So $k_ip_{i,j}$ is the cost incurred by delaying all these jobs. And $\sum_{j'|p_{i,j'} < p_{i,j}} p_{i,j'}$ is the delay of $j$, due to jobs which have lower time remaining.

The Greedy algorithm $A$ assigns job $j$, which arrives at time $t$, to the machine which increases flow the least, that is to machine $i^* = \arg\min_i Q_{i,j}$.

### 2.3 LP Formulation

Let us now consider an LP relaxation of our problem, which we will use only for the analysis of Greedy. Let $x_{i,j,t} = 1$ be a variable the denote the processing job $j$ receives on machine $i$ in time slot $t$. Consider the LP below. We have a constraint that for each machine, at each time point, at most 1 unit of work is done (we multiply this by $-1$ throughout to write it in a covering form). A job can only be worked at times $t > r_j$. Moreover, each job must be completed. This is captured by the second set of constraints. Note that this relaxation allows migrations and even executing a job on multiple machines at the same time. This can only improve the objective.

To write the objective function, we consider the fractional flow time of each job. Here we view $x_{i,j,t}/p_{i,j}$ fraction of $j$ completing at time $t$, and incurring at flow time of $(t - r_j)$. This gives the second term $\sum_{i,j,t>r_j} x_{i,j,t}(t - r_j)/p_{i,j}$ in the objective.

However, this term is not enough and the resulting LP can have a bad integrality gap. Consider a single job that has size $m$ on every machine. In any integral schedule it incurs flow time of $m$, 2
but the LP can split it across \( m \) machines, and finish it at time 1. To get around this, we note that \( F_j \) must be at least the processing time of each job, and thus \( F_j \sum_{i,t>r} x_{i,j,t} \). So the following LP is a relaxation within a factor of 2.

Writing out the Primal LP gets us:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i,j} \sum_{t>r} (x_{i,j,t} + x_{i,j,t}(t - r_j)/p_{i,j}) \\
\text{subject to} & \quad \sum_{j \in J} -x_{i,j,t} \geq -1 \quad \forall i,t \\
& \quad \sum_{j \in J \sum_{r_j<t}} x_{i,j,t}/p_{i,j} \geq 1 \quad \forall j \\
& \quad x_{i,j,t} \geq 0 \\
\end{align*}
\]

**Dual:** We now write the dual of this LP. Recall that for each constraint in our primal, we generate a dual variable. Therefore for our dual we have variables: \( \alpha_j, \beta_{i,t} \).

\[
\begin{align*}
\text{maximize} & \quad \sum_j \alpha_j - \sum_{i,t} \beta_{i,t} \\
\text{subject to} & \quad \alpha_j/p_{i,j} - \beta_{i,t} \leq (t - r_j)/p_{i,j} + 1 \quad \forall i,j,t > r_j \\
& \quad \alpha_j, \beta_{i,t} \geq 0 \\
\end{align*}
\]

As in previous applications of dual-fitting, we would like to set \( \alpha_j \) and \( \beta_{i,t} \) so that the dual value is high and this can be related to the performance of Greedy.

We first prove a small lemma:

**Lemma 4** \( \sum_j F_j = \sum_t n(t) \), where \( n(t) \) is the total number of jobs in the system at time \( t \).

**Proof:** By definition our total flow cost is equivalent to \( \sum_j F_j \). However, \( F_j = \text{Number of time steps that job } j \text{ is present in the system} \). Therefore, by a change of summation, total flow cost is equivalent to \( \sum_t n(t) \). \( \square \)

We will assign \( \alpha_j \) and \( \beta_{i,t} \) as follows.

1. \( \alpha_j = Q_{i^*,j} \), where \( i^* \) is the optimal machine \( i \) to place job \( j \) for Greedy. By definition, we see \( \sum_j \alpha_j = \text{Greedy Cost} \)

2. \( \beta_{i,t} = n(i,t) \) where \( n(i,t) = \text{number of jobs on machine } i \text{ at time } t \) for Greedy. We see \( \sum_{i,t} \beta_{i,t} = \sum_i \sum_t n(i,t) = \sum_i n(i) =, \) which by the above lemma is also exactly equal to the Greedy cost.

**Intuition for setting the dual variables:** While these choices may seem adhoc, this a natural a choice by consider the sensitivity analysis. In particular, if we perturb the right hand side of a primal constraint by \( \epsilon \), this changes the dual objective by \( \epsilon y \) where \( y \) is the corresponding dual variable. So \( y \) should measure the cost of this constraint. In our context, suppose we delete a time slot (i.e. change the rhs of one of the first constraints by 1 for some \( i,t \). Then, the total flow time will increase by \( n(i,t) \), the number of jobs on \( i \) at time \( t \). So this suggests setting \( \beta_{i,t} = n(i,t) \). Similarly, if we remove job \( j \) (set the right hand side of some constraint of second type by \(-1\)), then the cost \( Q_{i^*,j} \) due to \( j \) for other jobs will not be there. So, this suggests setting \( \alpha_j = Q_{i^*,j} \).
Lemma 5 These dual assignments are feasible. In other words we wish to show that:

\[
\forall i, j, t > r_j, \quad \alpha_j / p_{i,j} - \beta_{i,t} \leq (t - r_j) / p_{i,j} + 1
\]

or equivalently

\[
\forall i, j, t > r_j, \quad \alpha_j \leq \beta_{i,t} p_{i,j} + (t - r_j) + p_{i,j}.
\]

Proof: Let us fix a machine \(i\) and job \(j\). Note that we can assume that \(j\) is the final job, as if not \(\beta_{i,t}\) would only increase, and would only help the above inequality.

Sort the jobs that machine \(i\) has left at time \(t\) into two types of jobs, \(A_1\), where all jobs \(j_1, \ldots, j_r\) in \(A_1\) have processing time less than the processing time of \(j\), and \(A_2\) where all jobs \(j_{r+1}, \ldots, j_n\) have greater than or equal to the same processing time as \(j\). Let \(W_j\) denote the total remaining processing time of jobs smaller than \(p_j\), at the time when \(j\) arrives. Recall that \(Q_{i,j} = W_j + k_i p_{i,j} + p_{i,j}\).

Suppose at time \(t' > r_j\), machine \(i\) is executing job \(j_k\). When \(i = i^*\), we also need to consider the case when \(j_k = j\).

We have two cases:

1. Case 1 (\(j_k \in A_1\)): In this case note that \(n - r = k_i\). Moreover, as Greedy finished \((t - r_j)\) work on jobs of size less than \(p_j\), it holds that

\[
t - r_j = W_j - \sum_{j' \in A_1} p_{i,j'}
\]

We also have that \(b_{i,t} = n - k + 1\).

By the greedy choice of \(i^*\) where \(j\) was assigned, we have

\[
a_j \leq Q_{i,j} = W_j + k_i p_{i,j} + p_{i,j} = (t - r_j) + \sum_{j' \in A_1} p_{i,j'} + k_i p_{i,j} + p_{i,j}
\]

\[
\leq (t - r_j) + (r - k)p_{i,j} + (n - r)p_{i,j} + p_{i,j}
\]

\[
\leq (t - r_j) + p_{i,j} \beta_{i,t}
\]

In fact note that we have a slack of \(p_{i,j}\), and this is needed in the analysis for the case when \(i = i^*\).

2. Case 2 (\(j_k \in A_2\)): Here, as the jobs smaller than \(p_{i,j}\) and as \(k_i - \beta_{i,t}\) of the jobs at least as large as \(j\) has been finished, we have that

\[
(t - r_j) \geq W_j + (k_i - \beta_{i,t})p_{i,j}
\]

As before this gives

\[
a_j \leq Q_{i,j} = W_j + k_i p_{i,j} + p_{i,j} \leq (t - r_j) + \beta_{i,t} p_{i,j} + p_{i,j}
\]

\[\square\]
2.4 Using Resource Augmentation

While $\alpha$ and $\beta$ form a feasible solution for the dual, we note that the dual objective for this setting of values is zero, and this does not seem useful at all.

We now use that we are considering the resource augmentation setting. So the key insight will be the following. As we need to compare Greedy, with an offline algorithm running at speed $1/(1+\epsilon)$, we consider the dual for the LP where the speed $1/(1+\epsilon)$. Let us consider the dual LP, $D'$ for this setting. This is equivalent to the one below, and the only difference is that $p_{i,j}$ now becomes $(1+\epsilon)p_{i,j}$, will become $1/(1+\epsilon)$, as only $1/(1+\epsilon)$ unit of work can be done per time step.

The new dual LP $D'$:

$$\text{maximize } \sum_j \alpha'_{j} - \sum_{i,t} \beta'_{i,t}$$

subject to  $\alpha'_{j}/((1+\epsilon)p_{i,j}) - \beta'_{i,t} \leq (t - r_j)/(p_{i,j}(1+\epsilon)) + 1 \ \forall i, j, t > r_j$

$\alpha'_{j}, \beta'_{i,t} \geq 0$

Consider now the solution $\alpha' = \alpha$, and $\beta' = \beta/(1+\epsilon)$. Consider as $\alpha$ and $\beta$ form a feasible solution for our original LP, $\alpha'$ and $\beta'$ necessarily form a feasible solution for our modified LP, which can be seen by multiplying the constraint by $(1+\epsilon)$.

Note crucially that the dual objective for $D'$ now becomes

$$\sum_j \alpha'_{j} - \sum_{i,t} \beta'_{i,t} = \sum_j \alpha_j = \frac{1}{1+\epsilon} \sum_i \sum_t \beta'_{i,t} = \frac{\epsilon}{1+\epsilon} \sum_i \sum_t \beta'_{i,t} = \epsilon$$

which implies by dual fitting, we have a $1 + 1/\epsilon$ approximation wrt to the dual LP, and hence a $2 + 2/\epsilon$ competitive ratio.

3 Experts Learning, Online Prediction, Regret Minimization

Consider the online learning problem, where we have $n$ experts. At each time step $t$, an expert gives us a binary recommendation, and we must make a prediction based on this advice. At the end of time $t$, nature reveals the true outcome, and we learn whether we made and error or not. Our goal is to minimize the number of errors.

3.1 Prediction Problem with Perfect experts

Suppose there is an expert who is always correct (but the online algorithm of course does not know this). The following is easily seen.

Consider the algorithm that eliminates experts who made mistakes at the end of each round. To make its prediction is uses the majority recommendation.

**Theorem 6** This algorithm makes at most $\log_2(n)$ mistakes.

**Proof:** Whenever we make a mistake, at least half the remaining experts are eliminated.  \(\square\)
3.2 The Non-Perfect Expert’s Problem

Imagine a scenario where all of our experts are prone to mistakes. We wish our error to be roughly the error of the best expert. We propose the following algorithm for choosing experts, and prove an error bound.

**Weighted Majority:** For each expert, we maintain a weight $w_i(t)$ at time $t$. Let $w_i(0) = 1$. Take the weighted majority vote at each time point. If an expert was incorrect, update $w_i(t+1) = (1-\epsilon)w_i(t)$. Here we need $\epsilon \leq 1/2$ for technical reasons.

**Theorem 7** The number of mistakes $M$ that Weighted Majority makes is bounded by $(2 + \epsilon)m_j + 2\log(n)/\epsilon$ for every $j$, where $m_j$ is the error of expert $j$.

**Proof:** Define $\phi(t) = \sum_i w_i(t)$. Clearly, $\phi(0) = n$. We make the following two observations.

1. For all $i,t$ we trivially have $w_i(t) \leq \phi(t)$ and thus $(1-\epsilon)m_i(t) \leq \phi(t)$.

2. If online makes a mistake at time $t$, then
   
   $\phi(t+1) \leq (1-\epsilon/2)\phi(t)$.
   
   This follows as the incorrect experts at time $t$ have at least half the total weight $\sum_i w_i(t)$, and their weight is reduced by $(1-\epsilon)$. So,
   
   $\phi(t+1) = \sum_i w_i(t+1) \leq (1-\epsilon/2)\sum_i w_i(t) + \frac{1}{2}\sum_i w_i(t) \leq (1-\epsilon/2)\phi(t)$.

Putting these together, we can infer that

$$(1-\epsilon)m_i(T) \leq \phi(T) \leq (1-\epsilon/2)^M\phi(0) = (1-\epsilon/2)^Mn$$

Take the logarithm on both sides, and using that $-\epsilon - \epsilon^2 \leq \log(1-\epsilon) \leq -\epsilon$ for $\epsilon < 1/2$, we get

$m_i(T)\log(1-\epsilon) \leq M\log(1-\epsilon/2) + \log(n) \leq (-\epsilon - \epsilon^2)m_i(T) \leq M(-\epsilon/2) + \log(n)$

Rearranging gives,

$$M \leq (2 + 2\epsilon)m_i(T) + 2\log(n)/\epsilon$$

Notice that if $T$ is large enough, the effect of the $\log(n)$ is relatively small.

3.3 Randomized Experts Problem

It turns that the $2 + 2\epsilon$ factor above can be reduced to $(1 + \epsilon)$ dependence by turning this into a randomized algorithm. As we shall see later on, this will have numerous consequences.

We propose the following change from Section 3.3. Rather than taking the weighted majority, take the advice of expert $i$ with probability $p_i(t) = w_i(t)/\phi(t)$. We update the probabilities similarly as before upon mistakes.

We now consider the expected number of mistakes $E[M]$. We assume that nature knows our distribution $p^*$ while determining the outcome at time $t$, but it does not know our coin toss (otherwise we essentially get the deterministic setting).
**Theorem 8** For each expert $i$, $E[M] \leq (1 + \epsilon)m_i(T) + \log(n)/\epsilon$

**Proof:** Let $m_t$ denote the mistake vector for experts based on nature’s outcome at time $t$ (that is $m_t[i] = 1$ if expert $i$ made a mistake at time $t$). Therefore, the expected number of mistakes at time $t$ is $p_t \cdot m_t$. Moreover,

$$\phi(t + 1) = \sum_i w_i(t + 1) = \sum_i w_i(t) \cdot (1 - \epsilon m_i(t)) = \sum_i w_i(t) - \epsilon \sum_i w_i(t)m_i(t) = \phi(t)(1 - \epsilon p_t \cdot m_t)$$

The last step uses that $p_t(i) = w_i(t)/\phi(t)$.

Note that the weights evolve deterministically as they only depend on the mistakes of experts, which is independent on the randomness used by our online algorithm.

Let cost($t$) = $p_t m_t$ denote the algorithms expected cost at time $t$. Then, $\phi(t + 1) = \phi(t)(1 - \epsilon \text{cost}(t))$. This gives

$$\phi(T) = \phi(0) \prod_{t=1}^{T} (1 - \epsilon \text{cost}(t)) \leq ne^{-\epsilon \sum_{t=1}^{T} \text{cost}(t)} = ne^{-\epsilon E[M]}$$

Now as $(1 - \epsilon)m_i(T) \leq \phi(T) \leq ne^{-\epsilon E[M]}$, taking the log of both sides we have: $m_i(T) \log(1 - \epsilon) \leq -\epsilon E[M] + \log(n)$ and hence

$$m_i(T)(-\epsilon - \epsilon^2) \leq E[M](-\epsilon) + \log(n)$$

which gives $E[M] \leq (1 + \epsilon)m_i(T) + \log(n)/\epsilon$.

If $M^*$ is the loss of the best expert, the quantity $M - M^*$ is referred to as the regret, as thus expected regret can be bounded as

$$E[M] - M^* \leq \epsilon M^* + \log(n)/\epsilon$$

Setting $\epsilon = \sqrt{\log n/M^*}$, this implies a regret of $O(\sqrt{M^* \log n})$. Usually, $M^*$ can be estimated on the fly using standard doubling tricks.

**References**

