CSE586: Algorithms Under Uncertainty
Homework 1 Solution

Ashlesha Gupta (2021380) Divyajeet Singh (2021529)

Solution 1.

We are given a sequence of purchase options (d_1, p_1), (d_2, p_2), $\ldots$, (d_k, p_k), and options $(1, 1)$ and (∞, B). It is also given that the options follow economies of scale, i.e.

$$d_1 < d_2 < \cdots < d_k \implies \frac{p_1}{d_1} > \frac{p_2}{d_2} > \cdots > \frac{p_k}{d_k} \tag{1}$$

Claim 1. *It is okay to assume that* $p_{i-1} \leq \frac{p_i}{2}$ *for all* $2 \leq i \leq k$.

Proof. Assume there exist purchase options (d_{j-1}, p_{j-1}) and (d_j, p_j) for some $2 \leq j \leq k$ such that

$$p_{j-1} > \frac{p_j}{2} \tag{2}$$

$$d_{j-1} < d_j \tag{3}$$

If it snows for $d_{j-1} < d \leq d_{j+1}$ days, then we can either select the purchase option (d_j, p_j) once or the purchase option (d_{j-1}, p_{j-1}) some r times. By (3), $r > 1$. Let us assume we pick the second choice. Then, we pay

$$r \cdot p_{j-1} \geq r \cdot \frac{p_j}{2} \geq p_j \tag{4}$$

where the last inequality holds because $r \geq 2$. So, we end up paying more than the cost of the purchase option (d_j, p_j). So, we would never pick the purchase option (d_{j-1}, p_{j-1}).
If it snows for more than d_j days, then we can either select (d_j, p_j) some r_1 times or (d_{j-1}, p_{j-1}) some r_2 times. By (3), $r_1 \leq r_2$. In this case, we pick (d_j, p_j) r_1 times because by (1),

$$\frac{p_j}{d_j} > \frac{p_{j-1}}{d_{j-1}} \tag{5}$$

which means we get better value for money by picking (d_j, p_j) r_1 times.
Hence, it is safe to ignore (d_{j-1}, p_{j-1}) if (2) doesn't hold. □

Offline Optimal Algorithm

The offline ski-rental problem with multiple purchase options can be solved using dynamic programming. Assuming it snows for $T \leq |\sigma|$ days, for each day $1 \leq i \leq T$, we calculate the minimum cost of renting skiis for the first i days. The pseudocode for this algorithm is given in Algorithm 1.
The algorithm runs in $\mathcal{O}(T * k)$ time and $\mathcal{O}(T)$ space, meaning it is polynomial in the input size.

Algorithm 1 Offline Optimal Algorithm for modified the Ski-Rental Problem

```
1: procedure SKI-RENTAL($\sigma[1:N]$, $(d_i, p_i)[1:k]$):
2:     $T \leftarrow \sum_{i=1}^{N} \sigma[i]$                ▷ Total number of days it snows
3:     $C[0:T] \leftarrow \infty$
4:     $C[0] \leftarrow 0$
5:     for $i \leftarrow 1$ to $T$ do
6:         for $j \leftarrow 1$ to $i$ do
7:             $C[i] \leftarrow \min(C[i], C[i-d_j]+p_j)$
8:         end for
9:         $C[i] \leftarrow \min(C[i], C[i-1]+1)$                ▷ Rent for 1 day
10:    end for
11:    return $\min(C[T], B)$
12: end procedure
```

Competitive Online Algorithm

Now, we design an algorithm to solve the online version of the modified ski-rental porblem. Without loss of generality, we assume that the sequence of purchase options is sorted in increasing order of d_i's, and that $(d_1, p_1) = (1, 1)$ and $(d_k, p_k) = (\infty, B)$ (Note how it does not affect our algorithm or its analysis). Finally, we assume that (1) and Claim 1 hold.

Our algorithm, $\mathbf{\Lambda}$, is a generalization of the simple online ski-rental problem. In that version, we have two purchase options, $(1, 1)$ and (∞, B). The optimal strategy is to rent skiis (*use* $(1, 1)$) for $B - 1$ days and then buy them (*use* (∞, B)) on the last day, unless it stops snowing. This suggests that we use the purchase option (d_i, p_i) for $d = p_{i+1} - p_i$ days, before moving to the next purchase option. To generalize this, we need to divide the d days into intervals of d_i days. We rent this option till we exceed p_{i+1} days.

So, $\mathbf{\Lambda}$ uses purchase option (d_1, p_1) until it exceeds p_2 days. Then, it uses the option (d_2, p_2) until it exceeds p_3 days. This continues until we reach (d_k, p_k) (which is the option to buy the skiis forever), or it stops snowing. Here is a formalization of the behaviour of $\mathbf{\Lambda}$. Each purchase option (d_i, p_i) is used for

$$T_i = \frac{p_{i+1} - p_i + c_i}{d_i} \tag{6}$$

times for all $1 \leq i < k$. Here, $c_i \geq 0$ is the number of days by which we exceed the p_{i+1} days limit. Once we reach/exceed $p_k = B$ days, we use the option (∞, B) once.

Competitive Analysis for $\mathbf{\Lambda}$

The above proposed algorithm can be analyzed by simply establishing an upper bound on the overall cost paid by it. We then compare this to the cost paid by an offline optimal algorithm, $\mathbf{OPT}$, to find the competitive ratio. As mentioned above, we use each purchase option (d_i, p_i) for T_i times. $\mathbf{\Lambda}$ pays a cost of p_i for each of the T_i times.

From (6), we can see that $c_i \bmod d_i \equiv 0$ implies that the interval $p_{i+1} - p_i$ is exactly divisible into T_i chunks of d_i days each. We also can estimate an upper bound on c_i.

$$c_i \leq d_i \tag{7}$$

This is because we buy in chunks of d_i days, so we cannot exceed p_{i+1} days by more than d_i days. If the interval $p_{i+1} - p_i$ is exactly divisible by d_i, then we must use the option (d_i, p_i) once more to exceed the

p_{i+1} days limit. Following this, option (d_{i+1}, p_{i+1}) can be used. Otherwise, we exceed it by less than d_i days. So,

$$T_i = \frac{p_{i+1} - p_i + c_i}{d_i} \le \frac{p_{i+1} - p_i + d_i}{d_i} \tag{8}$$

Before proceeding with the two cases for the analysis, we make the following claim.

Claim 2. *For all purchase options (d_i, p_j), including $(1, 1)$ and (∞, B),*

$$\frac{p_i}{d_j} \le 1$$

Proof. It is easy to see that the equality holds for $(d_1, p_1) = (1, 1)$. For all other purchase options, we follow economies of scale, given by (1). So, we have

$$\frac{p_1}{d_1} = 1 > \frac{p_2}{d_2} > \cdots > \frac{p_{k-1}}{d_{k-1}} > \frac{p_k}{d_k} = 0 \tag{9}$$

where the last equality follows as $d_k = \infty$. So, the claim holds. □

Case 1: It snows for $D \ge p_k$ days

This is the simpler case. Since it snows for more than or equal to p_k days, $\mathbf{\Lambda}$ moves through all purchase options. So, the cost paid by $\mathbf{\Lambda}$ is (by (8) and Claim 2)

$$\begin{aligned}
\mathbf{\Lambda}(\sigma) &= \sum_{i=1}^{k-1} \left[\frac{p_{i+1} - p_i + c_i}{d_i} \cdot p_i \right] + p_k & (10)\\
&\le p_k + \sum_{i=1}^{k-1} \frac{p_i}{d_i} \cdot (p_{i+1} - p_i + d_i) & (11)\\
&= B + \sum_{i=1}^{k-1} \frac{p_i}{d_i} \cdot p_{i+1} - \sum_{i=1}^{k-1} \frac{p_i}{d_i} \cdot p_i + \sum_{i=1}^{k-1} p_i & (12)\\
&\le B + \sum_{i=1}^{k-1} p_{i+1} - \sum_{i=1}^{k-1} p_i + \sum_{i=1}^{k-1} p_i & (13)\\
&= B + \sum_{i=1}^{k-1} p_{i+1} & (14)
\end{aligned}$$

Now, all we need is a way to estimate the sum of all costs p_j for $1 \le j \le k-1$. We can do this by using Claim 1, according to which $p_{j-1} \le \frac{p_j}{2}$ for all $2 \le j \le k$. So,

$$\begin{aligned}
\sum_{i=1}^{k-1} p_{i+1} &= p_2 + p_3 + \cdots + p_{k-1} + p_k & (15)\\
&\le p_2 + p_3 + \cdots + \frac{p_k}{2} + p_k & (16)\\
&\le p_2 + p_3 + \cdots + \frac{p_k}{4} + \frac{p_k}{2} + p_k & (17)\\
&\le \frac{p_k}{2^{k-2}} + \frac{p_k}{2^{k-3}} + \cdots + \frac{p_k}{4} + \frac{p_k}{2} + p_k & (18)\\
&\le p_k \sum_{i=0}^{\infty} \frac{1}{2^i} = 2p_k = 2B & (19)
\end{aligned}$$

Hence,

$$\mathbf{\Lambda}(\sigma) \leq B + 2B = 3B \tag{20}$$

Therefore, our algorithm pays at most $3B$ in this case. Since it snows for more than B days, $\mathbf{OPT}(\sigma) = B$ since it simply buys the skiis on the first day.

$$\frac{\mathbf{\Lambda}(\sigma)}{\mathbf{OPT}(\sigma)} \leq \frac{3B}{B} = 3 \tag{21}$$

Case 2: It snows for $D < p_k$ days

We follow a similar argument. Suppose it stops snowing after $p_j \leq D < p_{j+1}$ days. Then, $\mathbf{\Lambda}$ pays a total of

$$\begin{aligned}
\mathbf{\Lambda}(\sigma) &\leq \sum_{i=1}^{j} \frac{p_i}{d_i} \cdot (p_{i+1} - p_i + d_i) && (22)\\
&\leq \sum_{i=1}^{j} p_{i+1} - \sum_{i=1}^{j} p_i + \sum_{i=1}^{j} p_i && (23)\\
&= \sum_{i=1}^{j} p_{i+1} = p_2 + p_3 + \cdots + p_j + p_{j+1} && (24)\\
&\leq \frac{p_{j+1}}{2^{j-1}} + \frac{p_{j+1}}{2^{j-2}} + \cdots + \frac{p_{j+1}}{4} + \frac{p_{j+1}}{2} + p_{j+1} && (25)\\
&\leq p_{j+1} \sum_{i=0}^{\infty} \frac{1}{2^i} = 2p_{j+1} && (26)
\end{aligned}$$

Hence, $\mathbf{\Lambda}(\sigma) \leq 2p_{j+1}$. Of course, $1 \leq \mathbf{OPT}(\sigma) \leq p_{j+1}$. However, we can show that $\mathbf{OPT}(\sigma) \leq \frac{p_{j+1}}{2}$. By Claim 2, there exists some purchase option (d_l, p_l) such that $d_l \geq D$ and $p_l \leq \frac{p_{j+1}}{2}$ following (1), as

$$p_j < D \implies \frac{p_j}{D} < 1 \implies \frac{p_{j+1}}{2D} < 1 \tag{27}$$

So, $\mathbf{OPT}$ uses either the option (d_l, p_l), or a combination of options (d_i, p_i) for $1 \leq i < l$. If it chooses the option (d_l, p_l), then as stated, $\mathbf{OPT}(\sigma) = p_l \leq \frac{p_{j+1}}{2}$. Otherwise, it uses a combination of options whose total cost is at most p_l. So, $\mathbf{OPT}(\sigma) \leq p_l \leq \frac{p_{j+1}}{2}$. Therefore, we finally have

$$\frac{\mathbf{\Lambda}(\sigma)}{\mathbf{OPT}(\sigma)} \leq \frac{2p_{j+1}}{\frac{p_{j+1}}{2}} = 4 \tag{28}$$

(21) and (28) collectively prove that $\mathbf{\Lambda}$ is $\mathbf{4}$-competitive.

Solution 2.

Since this problem is an extension of the preceding problem, we also proceed in a very similar fashion, both for the offline optimal and for the competitive online algorithm.

Offline Optimal Algorithm

This version of the problem can also be solved optimally using dynamic programming. The DP state is same as above - the minimum cost of renting skiis for the first i days for each $1 \leq i \leq N$, where N is the total

length of the input seqeunce.
We need to ensure that this algorithm does not unnecessarily make a purchase on a day that it does not snow. This can be easily handled by simply skipping to purchase such days. Note that it is ideal to not purchase on any day where $\sigma_i = 0$ because buying on a later day where $\sigma_{i^+} = 1$ will enable skiing for more days on which it snows.
The pseudocode for this algorithm is given in Algorithm 2.

Algorithm 2 Offline Optimal Algorithm for the modified Ski-Rental Problem with non-monotonic input

```
1:  procedure SKI-RENTAL-NON-MONOTONIC($\sigma[1:N]$, $(d_i, p_i)[1:k]$):
2:      $C[0:N] \leftarrow \infty$
3:      $C[0] \leftarrow 0$
4:      for $i \leftarrow 1$ to $N$ do
5:          if $\sigma[i] = 0$ then
6:              $C[i] = C[i-1]$
7:              continue
8:          end if
9:          for $j \leftarrow 1$ to $i$ do
10:             $C[i] \leftarrow \min(C[i], C[i-d_j] + p_j)$
11:         end for
12:         $C[i] \leftarrow \min(C[i], C[i-1] + 1)$                    ▷ Rent for 1 day
13:     end for
14:     return $\min(C[N], B)$
15: end procedure
```

Similar to the offline optimal in Solution-1, this algorithm runs in $\mathcal{O}(|\sigma| * k)$ time and $\mathcal{O}(|\sigma|)$ space, meaning it is polynomial in the input size.

Competitive Online Algorithm

We design an algorithm to solve the online version of this problem, following the same assumptions as Problem-1. There, the optimal strategy is to use each purchase option (d_i, p_i) until the limit of p_{i+1} days is exceeded. Here, the input sequence is not necessarily monotonic. So, we propose the following algorithm.
Our algorithm, $\mathbf{\Gamma}$, that solves this problem is an extension of $\mathbf{\Lambda}$ from the Solution-1. We can try running $\mathbf{\Lambda}$ repeatedly, before making a final purchase of $(d_k, p_k) = (\infty, B)$. The final purchase is made if some run of $\mathbf{\Lambda}$ ends with the last purchase option (d_k, p_k), or if at any point we have paid more than B. Let us denote the i^{th} *run/cycle* of $\mathbf{\Lambda}$ by $\mathbf{\Lambda}_i$.
We proceed with $\mathbf{\Lambda}_1$. After the use of each option (d_i, p_i) ends, we check if it was snowing on the last day (when we still had the skiis). If yes, continue $\mathbf{\Lambda}_1$ if it snows the next day; else, we do not make a purchase. If it wasn't snowing on the last day, start a new run, $\mathbf{\Lambda}_2$ on the next snowy day. If at any point we finish the run $\mathbf{\Lambda}_j$ where $j \geq 1$, i.e., we end up buying the skiis forever, we terminate the algorithm. Otherwise, this process continues until some run $\mathbf{\Lambda}_j$ uses the option (d_k, p_k), or we have paid more than B already.

Competitive Analysis for $\mathbf{\Gamma}$

We now consider the following cases.

Case 1: It snows more for $T \geq B$ days

In such a case, $\mathbf{\Gamma}$ will also buy the skiis forever, either before or on the B^{th} day. Let's say the cost it paid is some x, which covers at most $B-1$ days. Then, any renting option it chooses will make the total cost paid by it to at least B. The last purchase option it chooses cannot be (∞, B), else it would have bought the skiis forever. So, the purchase option, y that takes the total cost above B can be at most p_{k-1}, which is upper bounded by $\frac{p_k}{2} = \frac{B}{2}$.
So, our algorithm ends up paying $B - 1 + \frac{B}{2}$. Moreover, if it snows for any more days (if the T days are not over), $\mathbf{\Gamma}$ will simply buy the skiis forever. So, the total cost paid by $\mathbf{\Gamma}$ is at most

$$\mathbf{\Gamma}(\sigma) \leq B - 1 + \frac{B}{2} + B \leq \frac{5B}{2} \tag{29}$$

Since it snows for more than B days, $\mathbf{OPT}(\sigma) = B$ since it simply buys the skiis on the first day with its prior oofflline knowledge.

$$\frac{\mathbf{\Gamma}(\sigma)}{\mathbf{OPT}(\sigma)} \leq \frac{\frac{5B}{2}}{B} = 2.5 \tag{30}$$

Case 2: It snows for $T < B$ days

(30) and (**??**) collectively prove that $\mathbf{\Gamma}$ is $\mathcal{O}(k)$-competitive.

Solution 3.

Given algorithm (say $\mathbf{\Lambda}$) accesses an element in the list and swaps it with the element just before it. We try to establish a lower bound on the competitive ratio of $\mathbf{\Lambda}$ by constructing an adversary that forces $\mathbf{\Lambda}$ to access the last element of the list at every element request.
Assume the given list is $L[1:n] = [l_1, l_2, \ldots, l_{n-1}, l_n]$. Then the adversary constructs an online input sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots \rangle$ of length $|\sigma|$ as follows

- $\sigma_1 = l_n$. Then $\mathbf{\Lambda}$ swaps $L[n]$ with $L[n-1]$. Note how this brings l_{n-1} to $L[n]$.
- $\sigma_2 = l_{n-1}$, which is located at $L[n]$. Then $\mathbf{\Lambda}$ swaps $L[n]$ with $L[n-1]$, which brings l_n to $L[n]$.
- $\sigma_3 = l_n$. Repeat.

σ will keep alternating between l_n and l_{n-1}. This makes the cost of each request n. However, an offline optimal algorithm (say $\mathbf{OPT}$) would notice the bring l_n and l_{n-1} to the front of the list, at $L[2]$ and $L[1]$ respectively, on the first two requests. $\mathbf{OPT}$ would then pay a cost of 1 for each request.

$$\mathbf{\Lambda}(\sigma) = n \cdot |\sigma| \tag{31}$$

$$\mathbf{OPT}(\sigma) = 2n + 1 \cdot \frac{|\sigma| - 2}{2} + 2 \cdot \frac{|\sigma| - 2}{2} \tag{32}$$

$$= 2n + \frac{3}{2} \cdot (|\sigma| - 2) \tag{33}$$

So, the competitive ratio of $\mathbf{\Lambda}$ with respect to $\mathbf{OPT}$ is

$$\frac{\mathbf{\Lambda}(\sigma)}{\mathbf{OPT}(\sigma)} = \frac{n \cdot |\sigma|}{2n + 1.5 \cdot (|\sigma| - 2)} \tag{34}$$

$$\approx \frac{n \cdot |\sigma|}{2n + 1.5\,|\sigma|} \tag{35}$$

$$= n \cdot \frac{|\sigma|}{2n + 1.5\,|\sigma|} = \Omega(n) \tag{36}$$

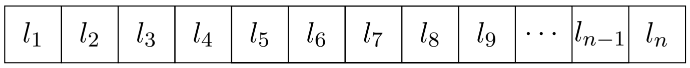

Figure 1: A sample list $L[1 : n]$

It is easy to see that in the limit that $|\sigma|$ increases, the competitive ratio approaches n. Hence, the competitive ratio of $\mathbf{\Lambda}$ is lower bounded by n. $\square$

Solution 4.

Given algorithm (say $\mathbf{\Lambda}$) accesses an element in the list and moves it half-way to the front, i.e. if the element is at position k from the front, then it is moved to position $\lfloor \frac{k}{2} \rfloor$, assuming 0-based indexing. We try to establish an upper bound on the performance of $\mathbf{\Lambda}$ with respect to an offline optimal algorithm, $\mathbf{OPT}$.

Claim 3. *$\mathbf{\Lambda}$ is 4-competitive.*

Proof. We use potential function analysis to prove the claim. Assuming both $\mathbf{\Lambda}$ and $\mathbf{OPT}$ start with the same list $L[1 : n]$, we define the potential function $\Phi(i)$ as twice the number of inversions in $\mathbf{\Lambda}$'s list as compared to $\mathbf{OPT}$'s list after the i^{th} request.
Under this assumption, $\Phi(0) = 0$ and $\Phi(i) \geq 0$ for all $i \geq 0$. Let $\Delta_i = \Phi(i) - \Phi(i-1)$ represent the change in potential after the i^{th} request. We prove that the cost paid by $\mathbf{\Lambda}$ on the i^{th} access is

$$\mathbf{\Lambda}(\sigma_i) \leq 4 \cdot \mathbf{OPT}(\sigma_i) - \Delta_i \tag{37}$$

Let x be the number of elements in $L\left[\frac{k}{2} : k\right]$ of $\mathbf{\Lambda}$ and to the left of σ_i in $\mathbf{OPT}$'s list. Let y be the number of elements in $L\left[\frac{k}{2} : k\right]$ of $\mathbf{\Lambda}$ but to the right of σ_i in $\mathbf{OPT}$'s list. It is easy to observe that

$$\mathbf{\Lambda}(\sigma_i) = \frac{k}{2} + x + y + 1 = k + 1 = 2 \cdot (x + y) + 1 \tag{38}$$

$$\mathbf{OPT}(\sigma_i) \geq x + 1 - r + \delta \tag{39}$$

0 where δ takes care of any additional exchanges done by $\mathbf{OPT}$. Then we track the change in potential after the i^{th} request. If $\mathbf{OPT}$ does not make any other exchanges on the i^{th} request, then the number of inversions in $\mathbf{\Lambda}$'s list increases by x and decreases by y.

$$\Delta_i = 2x - 2y + \delta \tag{40}$$

$$\implies \mathbf{\Lambda}(\sigma_i) + \Delta_i = 2x + 2y + 2x - 2y + 1 + \delta \tag{41}$$

$$= 4x + 1 + \delta \leq \mathbf{4} \cdot \mathbf{OPT}(\sigma_i) \tag{42}$$

If $\mathbf{OPT}$ moves $L[k]$ to the left, then the number of inversions decreases by at most x and increases by at most $\frac{k}{2}$. In this case,

$$\Delta_i = 2x - 2y + \delta - x + \frac{k}{2} \tag{43}$$

$$= 2x - 2y - x + (x + y) + \delta = 2x - y + \delta \tag{44}$$

$$\implies \mathbf{\Lambda}(\sigma_i) + \Delta_i = 2x + 2y + 2x - y + 1 + \delta \tag{45}$$

$$= 4x - y + 1 + \delta \leq \mathbf{4} \cdot \mathbf{OPT}(\sigma_i) \tag{46}$$

where the last inequality follows because $y > 0$, so the inequality holds even after ignoring y.
If $\mathbf{OPT}$ moves $L[k]$ to the right, the number of inversions may increase by some ρ but then $\mathbf{OPT}$ also pays a cost of ρ for the exchange. So they cancel out in the inequality. (42) and (46) collectively prove the inequality in (37).
Adding up the inequality (37) for all accesses $i = 1, 2, \ldots, |\sigma|$, we get

$$\begin{aligned}
\mathbf{\Lambda}(\sigma) = \sum_{i=1}^{|\sigma|} \mathbf{\Lambda}(\sigma_i) &\le \sum_{i=1}^{|\sigma|} 4 \cdot \mathbf{OPT}(\sigma_i) - \Delta_i && (47)\\
&= 4 \cdot \sum_{i=1}^{|\sigma|} \mathbf{OPT}(\sigma_i) - \sum_{i=1}^{|\sigma|} \Delta_i && (48)\\
&= 4 \cdot \mathbf{OPT}(\sigma) - \Phi(|\sigma|) + \Phi(0) && (49)\\
&= 4 \cdot \mathbf{OPT}(\sigma) - \Phi(|\sigma|) && (50)
\end{aligned}$$

where the last quality follows because $\Phi(0) = 0$. (50) proves the claim since $\Phi(|\sigma|) \ge 0$. $\square$

Solution 5.

We proceed with the proof in a similar fashion as that of online load balancing with restricted assignments where the size of each job is 1. Let m be the number of machines. Then, we need to prove that the greedy algorithm, $\mathbf{\Lambda}$, is still $\mathcal{O}(\log m)$-competitive. Let $\mathbf{OPT}(\sigma) = \lambda^*$ be the offline optimal solution. A very obvious fact is that the size of each job

$$p_i \le \lambda^* \text{ for } 1 \le i \le |\sigma| \tag{51}$$

We denote the load on machine x at some time t by $L_t(x)$. By way of contradiction, let us assume that the greedy algorithm is not $\mathcal{O}(\log m)$-competitive. Let use denote a value slightly larger than 2 by

$$\mathit{2}^+ = 2 + \frac{c}{\lambda^*} \quad \text{for some } c < \lambda^* \tag{52}$$

since $\frac{c}{\lambda^*} < 1$ by definition (of c). Consider the machine m_1 with the current highest load. Then,

$$L_{\text{curr}}(m_1) \ge (\mathit{2}^+ \log m + \mathit{2}^+) \cdot \lambda^* \tag{53}$$

Let J_1 be the set of the minimum number of jobs last assigned to m_1 such that their total load is more than $2\lambda^*$, i.e. the total load of J_1 is $2\lambda^* + c$ where $c < \lambda^*$. Since λ^* is the maximum load on any machine in $\mathbf{OPT}$, there must exist at least one more machine in $\mathbf{OPT}$ (and hence in $\mathbf{\Lambda}$), say m_2, where some k_1 many jobs from J_1 must be assigned such that their total load is at least λ^*. Let's call this subset J_2.
Consider the time τ when the first job from J_2 arrived and was assigned to m_1 by $\mathbf{\Lambda}$. This must have happened as $L_\tau(m_1) \le L_\tau(m_2)$. But $L_\tau(m_1) \ge L_{\text{curr}} - (2\lambda^* + c)$, i.e. $L_\tau(m_2) \ge L_{\text{curr}} - (2\lambda^* + c)$. Therefore, we conclude that at least two machines with load at least $L_{\text{curr}} - (2\lambda^* + c)$ exist in λ.
We now repeat the same argument by considering the jobs responsible for increasing the load on both machines m_1 and m_2 from at least $L - 2 \cdot (2\lambda^* + c)$ to at least $L - (2\lambda^* + c)$. This is always possible by assuming a sufficiently large load on machines m_1 and m_2. This gives that there are now some k_2 many jobs, having a total load of at least $2 \cdot (2\lambda^* + c)$. This means that $\mathbf{OPT}$ needs at least four machines to handle this load, otherwise its answer would be more than λ^*.

Each time we repeat this process, we have shown that the number of machines doubles. We can repeat this argument for some r times until

$$L_{\text{curr}} - r \cdot (2\lambda^* + c) \leq 2\lambda^* + c \tag{54}$$

$$\implies r \geq \frac{L_{\text{curr}} - 2\lambda^* - c}{2\lambda^* + c} \tag{55}$$

Now, by (53), we have

$$r > \frac{(2^+ \log m + 2^+) \cdot \lambda^* - (2\lambda^* + c)}{2\lambda^* + c} \tag{56}$$

$$= \frac{\lambda^*}{2\lambda^* + c}\left[\left(2 + \frac{c}{\lambda *}\right)\log m + \left(2 + \frac{c}{\lambda *}\right)\right] - 1 \tag{57}$$

$$= \frac{(2\lambda^* + c)\log m + (2\lambda^* + c)}{2\lambda^* + c} - 1 \tag{58}$$

$$= \log m + 1 - 1 = \log m \tag{59}$$

The above set of equations mean that there are at least more $2^r = m$ machines, which is a contradiction. Hence, the most heavily loaded machine must also have a load of at most $(2^+ \log m + 2^+) \cdot \lambda^*$. Since $\frac{c}{\lambda *} < 1$, we can also say that the load will be at most $(3 \log m + 3) \cdot \lambda^*$. $\square$

References

1. Lecture Notes, CSE586 (Monsoon 2023, Dr. Syamantak Das)

2. Lecture Notes, Online Algorithms (2011, Yossi Azar)