Computational Game Theory **Spring Semester, 2009/10**

Lecture 2: March 1 2010

Lecturer: Yishay Mansour *Scribes: Allon Wagner and Elad Liebman*[1]

2.1 Quality of Equilibria

In this lecture we discuss the quality of equilibria. Let us assume we have a global function which describes the social gain of all the players ("the common welfare"), achieving an optimum value of OPT. However, in a game in which every player attemps to maximize its private gain, a system could reach an equilibrium in which the social gain is not OPT. We would like to compare the "quality" of an achieved *Nash equilibrium* (NE) to the "quality" of the global optimum (OPT) - how "far off" are we from maximizing the total gain for the entire system?

2.1.1 Job Scheduling

We will discuss a game of jobs (players) and machines. Each job must be allocated to one machine only. The cost for one job is the load on the machine to which it is assigned.

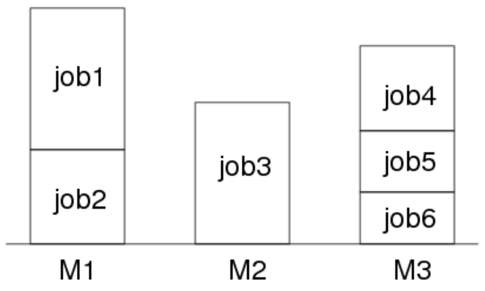

Figure 2.1: An example for the job scheduling game.

This is a special case of the routing problem. Assume there is a network of parallel lines from an origin S to a destination T as shown in figure 2.2. Several players want to send a

[1]These notes are based in part on old notes by Adi Adiv, Michal Rosen and Ricky Rosen (2006), and by Noa Bar-Yosef and Eitan Yaffe (2004).

particular amount of traffic from the source S to the destination T. The more traffic on a specific line, the longer the traffic delay (i.e. the cost).

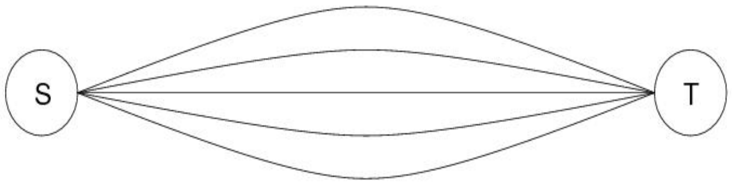

Figure 2.2: A reduction from the job scheduling problem to the routing problem

2.1.2 The Formal Model

- A group of $N = [1, \dots, n]$ jobs (players)
- A groupt of $M = [1, \dots, m]$ machines (we shall usually work under the assumption that $n \gg m$)
- Every machine M_j has speed $S_j \geq 0$
- Every job i has a weight value $w_i \geq 0$
- A *game*: Each player (job) $i \in N$ picks a machine $j \in M$

$$A = A_1 \times A_2 \times \ldots \times A_n \; where \; A_i = M$$

- The load on a machine M_j under the actions of the players $a \in A$ is defined as the sum of the weights of all the jobs that were allocated to it:

$$L_j(a) = \frac{\sum_{i:a_i=j} w_i}{S_j}$$

- The cost will simply be the load on the chosen machine:

$$COST_i(a) = C_i(a) = L_{a_i}(a) = \frac{\sum_{k:a_i=a_k} w_k}{S_{a_i}}$$

The social goal is to balance the load on the machines. We may consider two ways of measuring this:

- The *makespan* $MS(a) := \max_j L_j(a)$ is the value of the machine carrying the greatest load.
- The *social cost* $SC(a) := \sum_i w_i C_i(a)$ is the weighted sum of costs of all the players. It then follows that $SC(a) = \sum_{j \in M} L_j^2(a)$.

We will focus on the makespan.

Theorem 2.1. *Every job scheduling game has at least one pure equilibrium which is optimal.*

Proof. Intuition: We will define a total order relation for all the solution vectors, and show that every best response can only transfer us to a solution vector with a lower order, and the proof will follow.

For $a \in A$ we define:

$$L(a) = [(L_1(a), L_2(a), \ldots, L_m(a)]$$

We now sort by size: $\tilde{L}(a) = sort(L(a))$
(it should be noted that once we sort, different vectors do not necessarily have the same order of machines)

We define the order $a \preceq b$ iff $\tilde{L}(a) \leq_{\tilde{L}} \tilde{L}(b)$, where $\leq_{\tilde{L}}$ is the lexicographic order on the sorted vector (lexicographic order: $w \leq_L v$ if $w_i = v_i$ for $i = 0, ..., k$, and $w_{k+1} \leq v_{k+1}$) . Let $a^* \in A$ be such that for all $b \in A$, $a^* \preceq b$.

- a^* exists (since $\preceq$ is a total order).
- a^* is an optimal solution for MS (makespan), since the first coordinate in the sorted order is the most loaded machine, and for any other b, the first coordinate is at least as large.

Now we will also show that a^* is an equilibrium. Assume for contradiction that player j gains by moving from M_k to M_l resulting in a joint action b. The load on M_l after the move is smaller than M_k before the move. In addition, the load on M_k after the move is smaller than the load on M_k before the move. Lastly, note that $L_k(a) > L_l(a)$ (otherwise the job wouldn't have moved in the first place). Therefore, since a and b are identical in all machines except for M_k and M_l, it follows $\tilde{L}(a^*)\tilde{L}(b)$ and we have reached a contradiction to the minimality of a^*. □

Corollary 2.2. *If at any moment we allow one player to improve its cost by doing best response, we will eventually reach an equilibrium. It should be noted however that it will not necessarily be of an optimal cost.*

2.1.3 Measures for the Quality of an Equilibrium

- *Pure Nash Equilibrium* $PNE := \{a : \text{ a is a pure Nash equilibrium}\}$
- *Price of Anarchy* $PoA := \max_{a \in PNE} \frac{MS(a)}{OPT}$ - the worst ratio of an equilibrium to the optimal solution
- *Price of Stability* $PoS := \min_{a \in PNE} \frac{MS(a)}{OPT}$ - the best ratio of an equilibrium to the optimal solution

We've already established that for job scheduling $OPT \in PNE$ and therefore $PoS = 1$.

2.1.4 Two Identical Machines - an Example

As can be seen in Figure 2.3, at the right pure NE, the maximal load is 4. However, the maximal load of the *optimal solution*, which is also a NE is only 3. Therefore $PoA = \frac{4}{3}$. We will immediately show that this is the worst case.

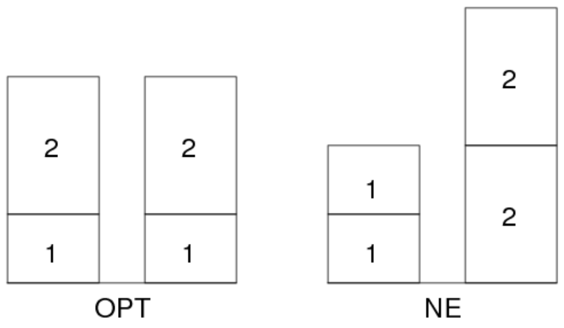

Figure 2.3: Two possible equilibria. It follows $PoA = \frac{4}{3}$

Theorem 2.3. *The price of anarchy for a job scheduling game with identical machines is* $PoA \leq (2 - \frac{2}{m+1})$

Proof. let $a \in PNE$ be a pure equilibrium. Let j^* be the machine with the heaviest load ($L_{j^*}(a) = MS(a)$). Let i^* be the lowest-weighing job on j^*.
Machine j^* must be assigned with at least one more job - otherwise it would be an optimal assignment. It follows then that i^* is at most $\frac{1}{2}L_{j^*}(a) = \frac{1}{2}MS(a)$. There is no machine $j \neq j^*$

for which $L_j(a) < L_{j^*}(a) - w_{i^*}$, otherwise, job i^* would move to machine j, thus contradicting the fact that we're in a state of equilibrium. Hence:

$$\forall j.\ L_j(a) \geq L_{j^*}(a) - w_{i^*} \geq \frac{1}{2} L_{j^*}(a)$$

OPT has to be at least the average load on each machine, therefore

$$OPT \geq \frac{\sum w_i}{m} = \frac{\sum L_j}{m} \Rightarrow$$

Therefore,

$$OPT \geq \frac{MS(a) + (m-1)\frac{1}{2}MS(a)}{m} = \frac{m+1}{2m} MS(a) \Rightarrow \frac{MS(a)}{OPT} \leq 2 - \frac{2}{m+1}$$

□

2.1.5 Tightness of the Upper Bound

We will now show this upper bound is tight (for every m) by giving an example in which PoA is $(2 - \frac{2}{m+1})$. Consider the following game: m machines, 2 jobs of size 1, and $m(m-1)$

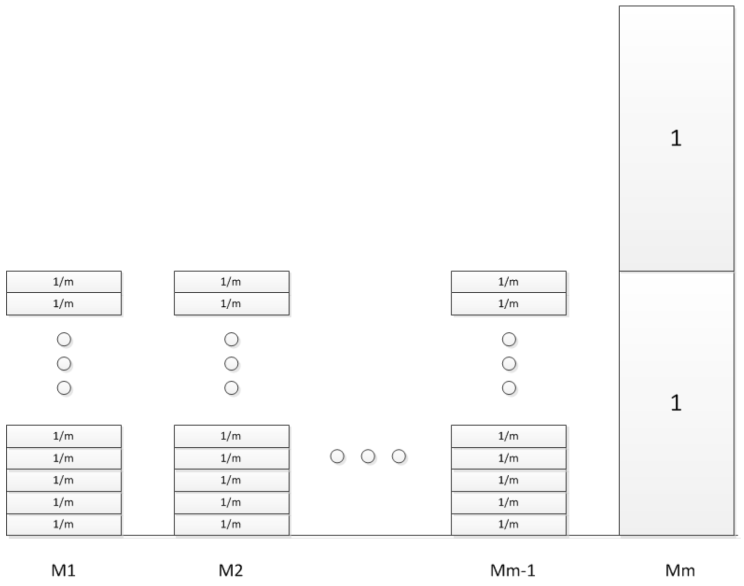

Figure 2.4: A game in which $PoA = (2 - \frac{2}{m+1})$

jobs of size $\frac{1}{m}$, as shown in Figure 2.4. One can easily verify that this is a NE with a cost of 2. The optimal configuration is obtained by scheduling the two "heavy" users (with $w = 1$) on two separate machines and dividing the other users among the rest of the machines. In this configuration we get a cost of: $C = OPT = 1 + \frac{1}{m}$. Thus, the PoA is $(2 - \frac{2}{m+1})$.

2.2 Related Machines

In this section, we will provide a tight upper bound on the PoA for JS games of related machines. First note the following observation:

Note 2.4. $OPT \geq \frac{\sum_i w_i}{\sum_j s_j}$

This is trivial, since dividing the jobs so that each machine carries an equal load is the best result we can hope for. Next we give a simple upper bound on the PoA.

Note 2.5. *There is a trivial upper bound: $PoA \leq m$*

Proof. Assume $s_1 \geq s_2 \geq \ldots \geq s_m$.
If all the jobs will pick the fastest machine L_1 then: $L_1^{all} = \frac{\sum w_i}{s_1} \leq m \cdot OPT$. The inequality follows from Note (2.4) and s_1 being the highest speed. If $a \in PNE$, every job i chooses not to move to machine M_1. Should it move, its new cost would necessarily be at most L_1^{all}. Since it chooses not to, we decude that its current cost is $c_i(a) \leq L_1^{all}$. We conclude $\forall a.\ MS(a) \leq L_1^{all}$ which means $PoA \leq m$. □

We will show a more involved tight upper bound of $O(\frac{\ln m}{\ln \ln m})$ for the price of anarchy. We will begin by demonstrating that for every m, the PoA might obtain that value. I.e, the bound is tight.

Example 2.6 (Lower Bound). Assume a fixed m. Let us have $k+1$ sets of machines with indices $l = 0, \ldots, k$. Let there be N_l machines in set l. The speed of each machine in group j will be $s_j := 2^j$. We will define:

- $N_k := \sqrt{m}$
- $\forall j < k.\ N_j := (j+1)N_{j+1}$
- And it follows that $N_0 := k!N_K$

Let us choose k in the following manner: we want $m = \sum_{l=0}^{k} N_l$. Note that each set is at least twice as large as the following set, and therefore $N_0 \sim \sum_{l=0}^{k} N_l$. From Stirling's approximation, it follows that $k \sim \frac{\ln N}{\ln \ln N}$ will satisfy this condition. We will describe a "high priced" equilibrium: Each machine in N_j has j jobs of size 2^j each. Thus the load on each

machine j is exactly j, and it follows $MS = k$.
We denote this assignment by a.

Claim 2.7. *The assignment a is a pure NE*

Proof. Let us consider a job i on machine in set N_j. The load that i observes in a is j. First, let us consider moving i to a machine in a set of a smaller index N_{j-t} $(t \geq 1)$. The current load on the new machine is $j - t$. With job i the load will be:

$$(j-t) + \frac{2^j}{2^{j-t}} = j - t + 2^t > j$$

i does not want to move.
Now, let us consider moving i to a machine in a set of a bigger index N_{j+t}. The new load on that machine will be

$$(j+t) + \frac{2^j}{2^{j+t}} > j$$

Again, i does not want to move. □

Note, however, that there are exactly $j \cdot N_j$ jobs on set N_j, each of size 2^j. We can place each of them on a seperate machine in group N_{j-1} (observe that we have exactly $j \cdot N_j$ machines in that group), causing load 2 on those machines. There is no problem with the group N_0 since there were no jobs on it in a - the original PNE.[2] We have shown $OPT \leq 2$. Therefore:

Corollary 2.8. $PoA \sim \frac{\ln m}{\ln \ln m}$

Now we will bound the price of anarchy from above.

Theorem 2.9. *For every JS game of related machines, $PoA = O(\frac{\ln m}{\ln \ln m})$*

Proof. Let a be the worst PNE. We will define $c := \lfloor \frac{MS(a)}{OPT} \rfloor$. We will show $(c-1)! \leq m$, and since $c \leq PoA \leq c+1$ it follows by Stirling's approximation that $PoA \sim \frac{\ln m}{\ln \ln m}$.
Let us sort the machines according to their speed $s_1 \geq s_2 \geq \ldots \geq s_m$.
Let J_k be the maximal prefix-set $J_k = \{1, \ldots, l\}$ *s.t.* $\forall j \in J_k$. $L_j(a) \geq k \cdot OPT$. *(see Figure 2.5)*
Observe that J_0 contains all machines. From Lemma 2.10 and Lemma 2.12 we deduce $m = |J_0| \geq (c-1)!$, and the theorem follows. □

Our current goal is to provide a lower bound for the size of the sets J_k. We start with the first set J_{c-1}.

Lemma 2.10. $|J_{c-1}| \geq 1$

[2] Observe that our proposed assignment is not optimal, however, since we do not place any jobs on the fastest machines N_k.

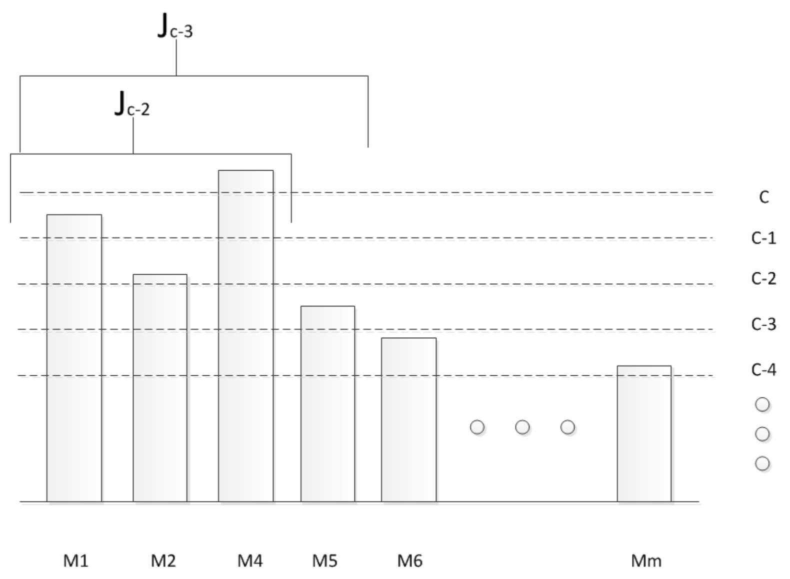

Figure 2.5: our definition of J_k. Note the machines are sorted by speed.

Proof. Assume by contradiction $|J_{c-1}| = 0$. This implies that $L_1(a) < (c-1)OPT$. But then there exists a machine $j > 1$ that achieves the MS: $L_j(a) \geq c \cdot OPT$. Pick an arbitrary job i running on L_j and transfer it to machine L_1. OPT has to put job i on some machine, and machine 1 is the fastest, so $\frac{w_i}{s_1} \leq OPT$. We conclude that L_1's new load is:

$$L_1^{new} = L_1(a) + \frac{w_i}{s_1} < (c-1)OPT + OPT \leq c \cdot OPT$$

Job i lowered its cost, so a cannot be NE. □

Let S_{k+1} be the set of jobs running on machines in J_{k+1}, i.e.,

$$S_{k+1} := \{i : \ a_i \in J_{k+1}\}$$

Lemma 2.11. *In every optimal assignment, every $i \in S_{k+1}$ will run on a machine in J_k.*

Proof. If $i \in S_{k+1}$ then $c_i(a) \geq (k+1)OPT$. Let q be the fastest machine which is not in J_k: $L_q(a) < k \cdot OPT$. Note that $w_i > s_q \cdot OPT$. Otherwise, we can transfer job i to machine q: $L_q^{new} = L_q(a) + \frac{w_i}{s_q} \leq L_q(a) + OPT < (k+1)OPT$. Job i improved its gain, but we know that a is a NE! Therefore, $w_i > S_q \cdot OPT$.

Now, assume by contradiction that there is an optimal assignment of jobs to machines in which i runs on a machine $p \notin J_k$. Then $\frac{w_i}{s_p} > \frac{s_q}{s_p} \cdot OPT \geq OPT$. The last inequality follows from q being the fastest machine not in J_k. But this means that $L_p(a) > OPT$, which is a contradiction to the optimality of a. $\square$

Next, we bound the size of J_k as a function of J_{k+1}.

Lemma 2.12. $|J_k| \geq (k+1)|J_{k+1}|$

Proof. Let $W_{k+1} := \sum_{i \in S_{k+1}} w_i$. Since S_{k+1} are exactly the jobs running on machines in J_{k+1} it follows that

$$W_{k+1} = \sum_{j \in J_{k+1}} s_j \cdot L_j(a) \geq (k+1)OPT \sum_{j \in J_{k+1}} s_j$$

By Lemma 2.11, in an optimal assignment all the jobs in S_{k+1} are placed on J_k and thus

$$OPT \sum_{j \in J_k} s_j \geq W_{k+1}$$

It follows that $OPT \sum_{j \in J_k} s_j \geq W_{k+1} \geq (k+1)OPT \sum_{j \in J_{k+1}} s_j$.

Let us define $s := min_{j \in J_{k+1}} s_j$. Obviously, since the J_k are prefixes, $\forall j \in J_k \setminus J_{k+1}$ we have that $s \geq s_j$. And from the previous inequality:

$$OPT(\sum_{j \in J_k \setminus J_{k+1}} s_j) + OPT(\sum_{j \in J_{k+1}} s_j) \geq (k+1)OPT(\sum_{j \in J_{k+1}} s_j)$$

$$OPT \cdot s|J_k \setminus J_{k+1}| + OPT(\sum_{j \in J_{k+1}} s_j) \geq (k+1)OPT(\sum_{j \in J_{k+1}} s_j)$$

$$OPT \cdot s(|J_k| - |J_{k+1}|) \geq k \cdot OPT(\sum_{j \in J_{k+1}} s_j) \geq k \cdot OPT \cdot s|J_{k+1}|$$

$$|J_k| \geq (k+1)|J_{k+1}| \quad \square$$

As previously stated, the combination of Lemmata 2.10 and 2.12 proves Theorem 2.9.

2.3 Unrelated Machines

In an unrelated machine model, job i's weight is dependant on the machine j on which it's placed. (Think, for instance, that we have several servers, each optimized for a specific kind of tasks. If we place a task on a matching server, it will be processed quickly. If we place it on a non-matching server it will take more time).

Formally, when job i is placed on machine j, it adds $w_{i,j}$ to the machine's load.

Remark. Related machines are a special case of unrelated machines with $w_{i,j} := \frac{w_i}{s_j}$.

Let us first demonstrate that now the PoA is not bounded.

Example 2.13. Assume we have 2 machines and 2 jobs with the following weight vectors: Job 1: $(1, \epsilon)$ Job 2: $(\epsilon, 1)$ (*see Figure 2.6*).
In case (1), no job would like to move, becuase on the new machine the load it will see is $1 + \epsilon$! $PoA = \frac{1}{\epsilon}$ and thus becomes arbitrarily big.

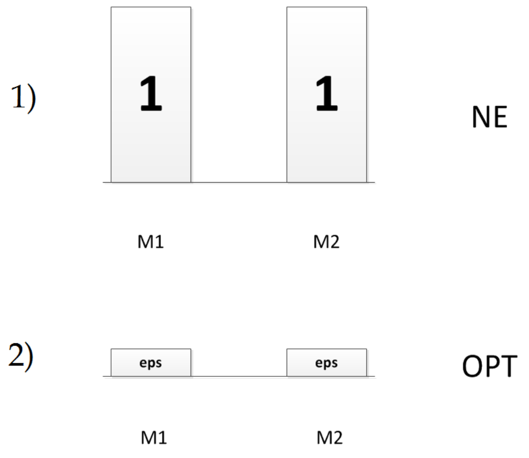

Figure 2.6: The PoA is unbound for unrelated machines

We will now present a new notion: a *coalition*. In Example (2.13) the high-priced equilibrium relied on the fact that only one player could try and change his action in every point in time. Since his condition would only have worsened if he changed his action, he chose not to. Now, we allow a group of players to join and change their actions simultaneously.

Definition Strong Equilibrium [Aumman 1956]

- A *coalition* is a group of players: $\Gamma \subseteq N$.
- A *deviation* is a joint action of all the members of the coalition: $\alpha \in \times_{i \in \Gamma} A_i$.
- An action a is *not robust against coalition* Γ if

$$\exists \alpha \in \times_{i \in \Gamma} A_i.\ \forall i \in \Gamma.\ u_i(\alpha, a_{-\Gamma}) < u_i(a)$$

  namely, there is a deviation from which $\underline{\text{all}}$ the coalition members benefit.

Strong EQ: $a \in A$ is robust against any coalition Γ.

Remark. Specifically, a is robust against a coalition $|\Gamma| = 1$, which makes it a PNE.

Note that Example 2.13 above is not robust against a coalition of size 2.

Remark. Strong EQ doesn't necessarily exist even if there is a pure NE. Consider, for instace, the prisoner's dilemma. There is exactly one pure NE (both confess) but it is not strong: if the two prisoners unite, they can both improve their gain (by deciding that they will both remain silent).

Definition Strong Price of Anarchy

$$SPoA := max_{a \in SE} \frac{c(a)}{OPT}$$

where c is the cost function, and SE is the group of strong equilibria.

Note 2.14. *$SE \subseteq NE$ and therefore $SPoA \leq PoA$*

Example 2.15. It is possible that all the coalition members improve, but other players are worse off. *(see Figure 2.7)*

Theorem 2.16. *For every JS game, $SPoA \leq m$*

Proof. Let $a \in A$ be a SE. $Wlog$, assume the machines are sorted by their load:

$$L_1(a) \geq L_2(a) \geq \ldots \geq L_m(a)$$

If $L_m(a) > OPT$ then we will simply take a coalition $\Gamma = N$ (a coalition of all the players). All the players will switch their action to the OPT action ($\alpha_i = opt_i$), and will all see a strictly smaller load than before. a is not SE.

Otherwise, $L_m \leq OPT$. We will show that $L_k(a) \leq L_{k+1}(a) + OPT$, and conclude by induction that $L_1(a) \leq m \cdot OPT$. Since $SPoA = \frac{L_1(a)}{OPT}$ in our case, we have proven the claim.

It remains to be shown that $L_k(a) \leq L_{k+1}(a) + OPT$. Assume by contradiction $L_k(a) > L_{k+1}(a) + OPT$. We will build a coalition of jobs $\Gamma = \{i : \ a_i \in \{1, \ldots, k\}\}$ and choose $\alpha_i := opt_i$ (i.e., each job chooses the machine it chooses in the optimal solution). Let us examine the new state of a player $i \in \Gamma$:

- if $\alpha_i \in \{1, \ldots, k\}$ then i's condition improved: we move all the jobs on machines $\{1, \ldots, k\}$ according to the opt solution, so the new load on every machine $\{1, \ldots, k\}$ is at most OPT. (And might be strictly less since the coalition, which is the set of jobs we might place on every machine $\{1, \ldots, k\}$, is a subset of all the jobs).

- if $\alpha_i \in \{k+1, \ldots, m\}$: note that we added to machine α_i at most OPT. Therefore, $L_{\alpha_i}^{new} \leq L_{\alpha_i}(a) + OPT \leq L_{k+1}(a) + OPT < L_k(a)$. Again, job i has improved its gain.

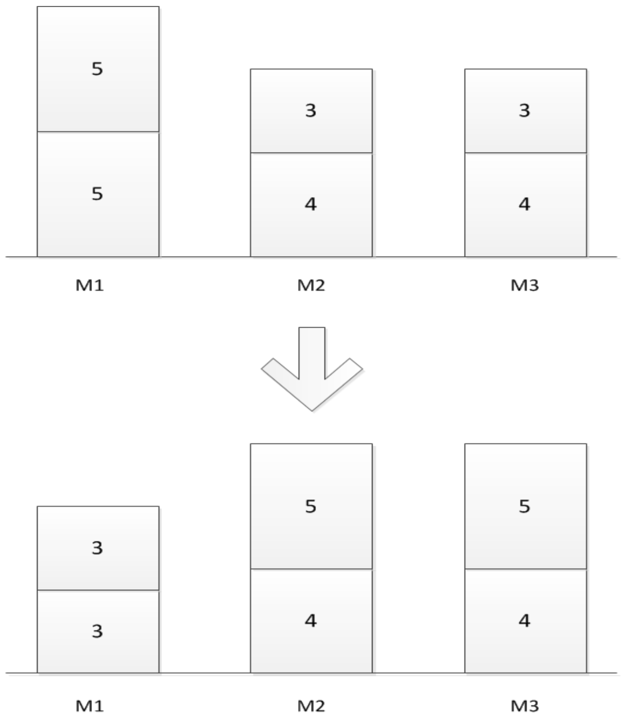

Figure 2.7: coalition example

We conclude that all the players in the coalition have improved their gain, hence a is not a SE. □

The next claim shows that the bound on the $SPoA$ is tight.

Claim 2.17. *For every m there exists a JS setting s.t. $SPoA = m$*

Proof. For a fixed m, choose $n = m$ to be the number of jobs. We will choose $w_{i,j}$ according to the following matrix $W = (w_{i,j})$ (Rows are jobs, columns are machines. Entries not explicitly stated are ∞):

$$W = \begin{bmatrix} 1 & & & & & 1 \\ 1 & 2 & & & & \\ & 1 & 3 & & & \\ & & 1 & \ddots & & \\ & & & 1 & m-1 & \\ & & & & 1 & m \end{bmatrix}$$

Namely, $w_{i,i} = i$; $w_{i+1,i} = 1$; $w_{1,n} = 1$; and all other $w_{i,j} = \infty$.

Obviously, the red solution is an optimal assignment with load 1. We will show that the blue assignment is a SE, which means that $SPoA = m$.

Assume that there exists a coalition Γ whose members can improve their cost. Let i be the smallest index in Γ. We will show that i cannot improve its cost. Clearly, $i > 1$, because job 1 is already placed on a machine with load 1, which is the minimal cost for job 1. It follows that there exists a job $i-1 \notin \Gamma$. This job places a load $i-1$ on machine $i-1$. Let us analyze the options of job i: The only possible way it can improve its gain is by moving to machine $i-1$. But then it will see load i on this machine (weight $i-1$ put by job $i-1$ in addition to its own weight), which means it did not improve its gain. □

Remember that our definition for SE necessitates *each* member of the coalition to *strictly* improve its gain. We might have conceived an alternative definition: we could have required *at least* one member of the coalition to improve its gain (and the others to be unharmed). However, such an alternative definition leads to non-intuitive results. For instance, according to the "classic" definition, the state in Figure (2.8) is a SE. If we switch to the other definition however, there is no SE since every two jobs placed on one machine will form a coalition.

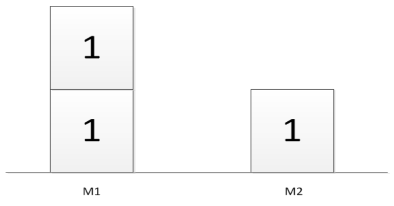

Figure 2.8: After we change the definition of a coalition, any two jobs placed on the same machine would form one, in this specific example.