CS 684 Algorithmic Game Theory	Scribe: Peter Richtárik
Instructor: Eva Tardos	August 29, 2005

1 Discrete (atomic) Load Balancing Game

Consider the following model. There are n *users* and m *machines (servers)*. Every user wants to impose a "permanent" load (*job*) on one of the machines. Let p_i be the *job size* of user i. We assume that user i can be served only by machines $j \in S_i \subset \{1, \dots, m\}$. A reason for this might be physical proximity or availability of resources, for instance.

Let A be a *feasible assignment* of jobs to machines (also called just *assignment* or *solution*), i.e. for every job i there is a unique machine $j \in S_i$ such that $(i, j) \in A$. For a fixed assignment A, let

$$L_j := \sum_{i:(i,j)\in A} p_i$$

be the total *load* on machine j. We assume that the *response time* of machine j for every job processed on it is a function of the total load L_j on this machine. Let's denote it $r_j(\cdot)$.

If we view the users as independent selfish players seeking to minimize response time of their jobs, we have a game-theoretic model.

Example. This is an example of a discrete load balancing game with 4 users and 3 machines. Let $S_1 = \{1\}$, $S_2 = \{1, 2\}$, $S_3 = \{1, 2\}$ and $S_4 = \{1, 2, 3\}$. The red dashed lines in the figure below denote a feasible assignment of jobs to machines.

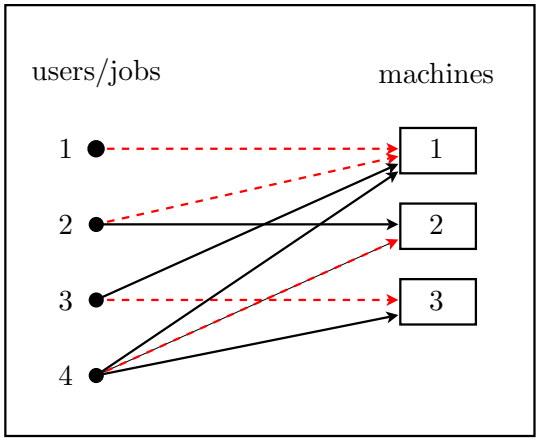

Figure 1: A simple load balancing game setting with 4 users/jobs and 3 machines.

Assuming unit jobs sizes ($p_i = 1$ for $i = 1, 2, 3, 4$), this assignment creates a load of 2 on the first machine and load 1 on machine 2 and 3. If we assume that $r_i(L) = iL$, then the response times of machines $1, 2$ and 3 are $2, 2$ and 3, respectively.

1.1 Formulation of Nash equilibria

A feasible assignment A is a Nash equilibrium if no user i (assume $(i,j) \in A$) has unilateral incentive to put his job on a machine $k \neq j$ where $k \in S_i$. In other words, assignment A is at Nash if and only if for all i (assuming $(i,j) \in A$) and all $k \in S_i$:

$$r_j(L_j) \leq r_k(L_k + p_i)$$

Example. Consider a simple situation with 2 users, 2 machines and unit job lengths, i.e. $(p_1 = p_2 = 1)$. Let $S_1 = S_2 = \{1,2\}$ so that both machines are accessible by both users. Lastly, let $r_1(L) = 2$ and $r_2(L) = L$, i.e. the response time of the first machine is constant and does not depend on the load whereas the response time of the second machine is linear and equals to its load. Out of the four possible assignments, only $\{(1,1),(2,1)\}$ is not at Nash. Indeed, in this solution, response time of both is 2. Hence both would gain from switching (alone) to machine 2 as their response time there would only be 1. It can be easily verified that the remaining three assignments are at Nash.

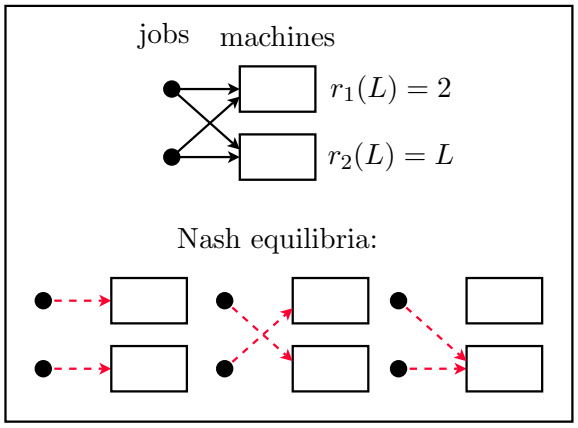

Figure 2: Nash equilibria in a simple load balancing game.

1.2 Existence of Nash

It will be shown later that the discrete load balancing game has always a Nash equilibrium. The argument is basically as follows: start with any assignment and let the players, one-at-a-time, switch machines if they are unhappy. If we can find a function (of assignment as an argument) which decreases with every switch, then clearly the process will terminate (cycling is prevented by monotonicity of the function and there is a finite number of feasible assignments) reaching a state where no player has further incentive to switch.

1.3 Quality of Nash

There are a few natural quality/cost measures to consider:

- worst response time:

$$\max_{j:L_j>0} r_j(L_j)$$

- average/total response time (version 1):

$$\sum_{i:(i,j)\in A} r_j(L_j)$$

- average/total response time (version 2):

$$\sum_{(i,j)\in A} p_i r_j(L_j) = \sum_j \sum_{i:(i,j)\in A} p_i r_j(L_j) = \sum_j L_j r_j(L_j)$$

Theorem 1 *With respect to the worst response time cost function, there are discrete load balancing games with arbitrarily bad Nash equilibria.*

Proof: We will first give an example where the optimal assignment (a Nash) has cost 1 and exhibit a Nash with cost 3. Consider the situation as shown in the diagram below. Let the response time of all machines be equal to their load, i.e. let $r_j(L) = L$ for all machines j. The red dashed assignment (where all the jobs "go to the left") is optimal with a cost of 1 since all jobs are assigned to different machines and hence all response times are 1. This assignment is clearly a Nash. Consider an alternative assignment with all jobs going to the right (black solid arrows). Here 3 jobs experience a response time of 3 and hence the cost of the assignment is 3. This assignment is also clearly a Nash.

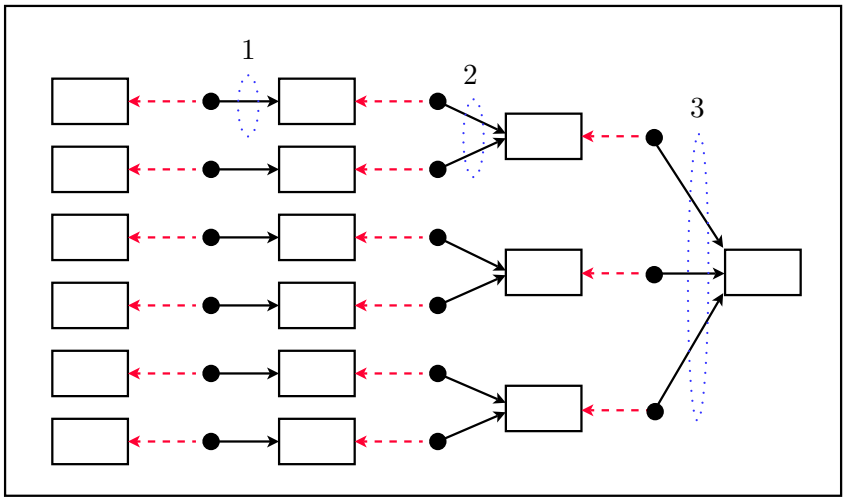

Figure 3: Existence of arbitrarily bad Nash. The optimal (red dashed arrows) assignment (a Nash) has cost 1, while the black solid assignment (also a Nash) has cost 3.

Without changing the structure of our simple example, it is possible to construct an arbitrarily bad Nash by making the response time functions more steep. On the other hand, if we want

to keep linear response times, our setting can be naturally expanded to exhibit arbitrarily bad Nash equilibrium. Let's show how we can do the latter. Let $n > 3$ be any positive integer. It is straightforward to construct a situation with an optimal cost 1 and a Nash with cost n. In analogy with the above picture, we will need to arrange machines in $n+1$ columns, from left to right, with $n!$ machines in the first column, $n!/1!$ machines in the second, $n!/2!$ in the third, $n!/(k-1)!$ in the k-th column and just $n!/n! = 1$ machine in the last column. The structure of jobs and the sets S_j stays identical. It is clear that the all-jobs-to-the-left assignment is optimal, with cost equal to 1. The all-jobs-to-the-right assignment is a Nash, and has cost equal to n since the n rightmost jobs are processed on just 1 machine. ■