Electronic Colloquium on Computational Complexity, Report No. 56 (2005)

The price of *Optimum* in Stackelberg games

A. C. Kaporis[*] [†‡] E. I. Politopoulou[*†§] P. G. Spirakis [*†§¶]

April 25, 2005

Abstract

Consider a system M of parallel machines, each with a strictly increasing and differentiable load dependent latency function. The users of such a system are of infinite number and act selfishly, routing their infinitesimally small portion of the total flow r they control, to machines of currently minimum delay. It is well known that such selfishness if modeled by a noncooperative game may yield a *Nash Equilibrium* on M with cost unboundedly worst than the overall *Optimum* one. We model such a system as a *Stackelberg* or *Leader-Followers* game, and present a simple algorithm that computes the *least* flow β_M (or "*price of optimum*") that must be controlled by a *Leader* in order to impose the overall optimum cost on M, as well as *Leader's* optimum strategy. The efficiency of our algorithm depends on the computation of the optimum and Nash assignment on such systems. Such assignments can be computed efficiently [4] for the classes of latency functions that we are interested in. Our motivation was [21] were the open question of computing β_M on a arbitrary system M was posed.

We were also greatly inspired from [7], in which the computation of β_M was a major issue. In that article systems with M/M/1 latency functions were studied, which are widely met in real world applications. Furthermore, $\beta_M(n)$ was computed explicitly for Stackelberg games with either $n = 1$ or a finite number n of *Followers*. It was demonstrated that $\beta_M(n)$ is nondecreasing on n, and as n increases it becomes harder for the *Leader* to impose the overall optimum. Most notably, it was conjectured that if $n \to \infty$ then it is not possible for the *Leader* to impose system's overall optimum. This comes into contrast to our theoretical results. As a by-product, we present a simple algorithm that computes the *Nash Equilibrium* for a system M with M/M/1 latency functions.

We should stress here that the model of parallel machines, despite its simplicity, incurs the worst *coordination ratio* as proved in [19].

1 Introduction

Selfish behavior in dynamic large scale networks such as Internet can be studied in the setting of a noncooperative game through the mathematical framework of Game Theory. Decisions in such networks are taken independently by their users [4, 7, 22, 17]. The users according to their

[*]Email addresses: {kaporis,politop}@ceid.upatras.gr, {politop,spirakis}@cti.gr

[†]Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece.

[‡]Partially supported by European Social Fund (ESF), Operational Program for Educational and Vacational Training II *(EPEAEK II)*, and particularly *Pythagoras*.

[§]Research Academic Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece.

[¶]Partially supported by Future and Emerging Technologies programme of the EU under EU contract 001907 "*Dynamically Evolving, Large Scale Information Systems (DELIS)*".

ISSN 1433-8092

own individual performance objectives make their choices bringing the network to fixed points, where nobody want to deviate from, called *Nash Equilibria* [15, 16]. In this context, the interested reader can find much of theoretic work in [7, 9, 20, 23, 10, 11, 12, 16, 17, 18]. Nash Equilibria are inefficient and exhibit at general suboptimal network performance. As a measure of how worse is the Nash equilibrium compared to the overall system's optimum, the notion of *coordination ratio* was introduced in [18, 11]. Their work have been extended and improved (*price of anarchy* here is another equivalent notion) in [3, 4, 6, 11, 19, 20, 22].

As it is mentionted above the selfish behavior may be modeled by a noncooperative game. Such a game could impose strategies that might induce an equilibrium closer to the overall optimum. These strategies are formulated through pricing mechanisms [5], algorithmic mechanisms [13, 14] and network design [23, 8]. The network administrator or designer can define prices, rules or even construct the network, in such a way that induces near optimal performance when the users selfishly use the system. Particulary interesting is the approach where the network manager takes part to the non-cooperative game. The manager has the ability to control centrally a part of the system resources, while the rest resources are used by the selfish users. This approach has been studied through *Stackelberg* or *Leader-Follower* games [2, 20, 7, 9, 24]. The advantage of this approach is that it might be easier to be deployed in large scale networks. This can be so, since there is no need to add extra components to the network or, to exchange information between the users of the network.

1.1 Motivation and contribution

Our work is motivated from [7, 20]. In [20] Roughgarden presented the `LLF` Stackelberg strategy for a *Leader*, on a noncooperative game with an infinite number of *Followers*, each routing its infinitesimal flow through machines of currently minimum delay (this setting corresponds to a model well known as *Flow Model* [22]).

An important question possed in [21] was the computation of the least portion β_M that a *Leader* must control in order to enforce the overall optimum cost on a system M. In Section 2 we present algorithm `OpTop` which computes β_M and its optimality proof in Section 3. Most importantly, we prove that `OpTop` is optimal for any system M with any class of latency functions for which we can compute efficiently Nash and optimum assignments. For example, see page 18 in [4].

On trying to compute β_M, we were greatly inspired from [7]. Their study was devoted to machines with M/M/1 latency functions, that are widely met in real world applications. An enlightening aspect of their work is that Korilis et al consider *Leader-Followers* noncooperative games such that, either the number n of *Followers*, or the routing scheme of each individual *Follower*, may vary. Thus, they investigate $\beta_M(n)$ on any system M such that $n = 1$, $n = n_0$ (with n_0 finite), and $n \to \infty$. As for routing scenarios, they consider cases such that (i) each *Follower* adjusts his routing strategy according to the decisions of the *Leader* in his desire to minimize his individual cost, and (ii) each *Follower* ships his flow through machines (links) of minimum currently delay (*simple-followers*). Concerning routing scenario (i) it is proved that $\beta_M(n)$ is nondecreasing on finite n (see Proposition 6.5) and as n is increasing it becomes harder for the *Leader* to impose the optimum on M. According to routing scenario (ii), they conjecture that it is *impossible* for a *Leader* to enforce the optimum on M as $n \to \infty$ ([7] see Section 8, last paragraph of page 24). These should be contrasted to our results, since in such a routing scenario, `Optop` always manages to impose the optimum on M, computing the *least* possible $\beta_M(n)$.

In Appendix, our theoretical analysis of `OpTop` is validated experimentally with an algorithm

implemented in C, that takes as input a 3-tuple of machines M (with linear or M/M/1 latencies) and a total of flow r. Finally, also in Appendix we present a simple polynomial algorithm for computing a *Nash Equilibrium* in a system M with M/M/1 latency functions working in the *Flow Model* (see also [4] for a nice presentation of the area).

1.2 Model - Stackelberg strategies

For this study the model and notation of [20] is used. We consider a set M of m machines, each with a latency function $\ell(\cdot) \geq 0$ continuous, differentiable and strictly increasing, that measures the load depended time that is required to complete a job. Jobs are assigned to M in a finite and positive rate r. Let the m-vector $X \in \mathcal{R}_+^m$ denote the assignment of jobs to the machines in M such that $\sum_{i=1}^m x_i = r$. The latency of machine $M_i \in M$ with load x_i is $\ell_i(x_i)$ and incurs cost $x_i\ell_i(x_i)$, convex on x_i. This instance is annotated (M, r). The *Cost* of an assignment $X \in \mathcal{R}_+^m$ on the (M, r) instance is $C(X) = \sum_{i=1}^m x_i\ell_i(x_i)$, measuring system's performance. The minimum cost is incurred by a unique assignment $O \in \mathcal{R}_+^m$, called the *Optimum* assignment. The unique assignment $N \in \mathcal{R}_+^m$ defines a *Nash equilibrium*, if no user can find a loaded machine with lower latency than any other loaded machine. That is, each machine $M_i \in M$ with load $n_i > 0$ experiences commmon latency L while any machine $M_j \in M$ with load $n_j = 0$ experiences latency $L_j \geq L$. The Nash assignment N causes cost $C(N)$ commonly referred to as *Social Cost* [12, 10, 11, 7, 9, 16, 17, 18]. The social cost $C(N)$ is higher than the optimal one $C(O)$, leading to a degradation in system performance. The last is quantified via the *Coordination Ratio*[10, 11, 3] or *Price of Anarchy (PoA)* [22], i.e. the worst-possible ratio between the social cost and optimal cost: $PoA = \frac{C(N)}{C(O)}$, and the goal is to minimize PoA [1]. To do so, a *hierarchical non cooperative Leader-Follower* or *Stackelberg game* is used [2, 20, 7, 9, 24]. In such a game, there is a set M of machines, jobs with flow rate r and a distinguished player or *Leader* who is responsible for assigning centrally a α portion of the rate r to the system so as to decrease the total social cost of the system. The rest of the players, called *Followers* are assigning selfishly the remaining $(1-\alpha)r$ flow in order to minimize their individual cost. The Leader assigns $S \in \mathcal{R}_+^m$ to M and the Followers react, inducing an assignment in Nash assignment T. The goal of the *Leader* is achieved if $C(S+T) = C(O)$.

2 Algorithm `OpTop`

In this section we present an efficient algorithm that computes the least flow that should be controlled by the leader in order to induce the overall optimum on a given system. Next we give an example where we run this algorithm on a set of 5 machines with M/M/1 latency functions. Let the optimum $O := \langle o_1, \ldots, o_m \rangle$ and the Nash assignment $N := \langle n_1, \ldots, n_m \rangle$ of flow r on to system $M = \{M_1, \ldots, M_m\}$. Intuitively, `OpTop` initially loads $s_i = o_i$ to each machine $M_i \in M$ with $n_i < o_i$, that is, to all machines not appealing to the selfish users. Then it discards all these not appealing machines. The remaining flow is assigned recursively by `OpTop` in exactly the same fashion to the simplified subsystem of machines. It terminates as soon as it encounters a simplified system with all of its machines optimally loaded (see also Example 1).

[1] Notice that in a general setting may exist a set A of Nash equilibria, then PoA is defined with respect to worst one, i.e. $PoA = \max_{N \in A} \frac{C(N)}{C(O)}$.

Algorithm: OpTop $(M,\ r)$
Input: Machines $M = \{M_1, \ldots, M_m\}$, flow r
Output: A portion β of flow r
begin:
  Compute Optimum assignment $O := \langle o_i : M_i \in M \rangle$ of the flow r on machines M
  $r_0 = r$; β = CompOpTop$(M,\ r,\ r_0,\ O)$;
end;
Procedure: CompOpTop $(M,\ r,\ r_0,\ O)$
Input: Machines M, flow r, initial flow r_0, Optimum assignment $O := \langle o_i : M_i \in M \rangle$
Output: A portion β of initial flow r_0
begin:
  Compute the Nash assignment $N := \langle n_i : M_i \in M \rangle$ of flow r on M;
  If $(N \equiv O)$ **return** $\beta = (r_0 - r)/r_0$;
    else $(M, r, O) \leftarrow$ Simplify(M, r, N, O); **return** CompOpTop(M, r, r_0, O);
  end if;
end;
Procedure: Simplify(M, r, N, O)
Input: Machines M, flow r
    Nash assignment $N := \langle n_i : M_i \in M \rangle$, Optimum assignment $O := \langle o_i : M_i \in M \rangle$
Output: Reduced machines M, Reduced flow r, Reduced Optimum $O := \langle o_i : M_i \in M \rangle$
begin:
  for $i = 1$ **to** size(M) **do:**
    If $o_i > n_i$ **then**
      $r \leftarrow r - o_i$; $M \leftarrow M \backslash \{M_i\}$; $O := O \setminus \langle o_i \rangle$;
    end if;
  end for;
  return (M, r, O);
end;

The numerical stability of OpTop depends solely on the finite precision accuracy of the computation of Nash and Optimum assignments on a given system M. Such computations can be performed efficiently with arbitrary digits precision for all classes of machines considered in [4]. In particular, for linear latency functions efficient computations of Nash and Optimum assignments can be found in [20]. Also, for the case of M/M/1, an algorithm that runs in $O(m^2)$ time and computes a Nash assignment is given in Appendix and for the corresponding Optimum see [7]. Therefore, for such latency classes OpTop runs in $O(m^3)$. In the following example, we run the OpTop algorithm presented above, for a set of 5 machines with M/M/1 latency functions.

Example 1 *Suppose a set M of 5 machines with latency functions $\ell_1(x_1) = \frac{1}{0.6-x_1}$, $\ell_2(x_2) = \frac{1}{0.24-x_2}$, $\ell_3(x_3) = \frac{1}{0.3-x_3}$, $\ell_4(x_4) = \frac{1}{0.2-x_4}$, $\ell_5(x_5) = \frac{1}{0.4-x_5}$. The total flow of the system is $r = 1$. We run the algorithm* OpTop *on the above set M and get the minimum portion of the flow r that should be controlled centrally to induce the optimum performance of the system.*

The Optimum assignment on M is $O = \langle o_1, \ldots, o_5 \rangle = \langle 0.4017, 0.1146, 0.1598, 0.0855, 0.2381 \rangle$ and the Nash one is $N = \langle n_1, \ldots, n_5 \rangle = \langle 0.452, 0.092, 0.152, 0.052, 0.252 \rangle$ (Figure 1(a) in Appendix). Observe that $o_2 > n_2, o_3 > n_3, o_4 > n_4$ and OpTop assigns *optimally* flow to machines M_2, M_3, M_4 equal to $o_2 + o_3 + o_4 = 0.3599$. Then OpTop simplifies the system $M = \{M_1, \ldots, M_5\}$ to the subsystem $M' = M \backslash \{M_2, M_3, M_4\} = \{M_1, M_5\}$. The remaining flow r' that is going to be scheduled on $M' = \{M_1, M_5\}$ is $r' = r - (o_2 + o_3 + o_4) = 0.6401$. OpTop computes the Nash assignment

$N' = \langle n'_1, n'_5 \rangle = \langle 0.4199, 0.2199 \rangle$ on $M' = \{M_1, M_5\}$ of the remaining flow $r' = 0.6401$. The overall optimum assignment O reduces to $O' = \langle o_1, o_5 \rangle = \langle 0.4017, 0.2381 \rangle$ with respect to subsystem $M' = \{M_1, M_5\}$ (Figure 1(b) in Appendix). Observe $M_5 \in M'$ has Nash load $n'_5 < o_5$ so M' is further simplified to $M'' = \{M_1\}$ and the remaining flow is $r'' = r' - o_5 = 0.4017$ (while M_5 receives its optimal load o_5). Once more, `OpTop` computes the Nash assignment $N'' = \langle n''_1 \rangle = \langle 0.4017 \rangle$ on $M'' = \{M_1\}$ (Figure 1(c) in Appendix) and the optimum one reduces with respect to M'' as $O'' = \langle o_1 \rangle = \langle 0.4017 \rangle$. That is, `OpTop` has encountered a subsystem $M'' = \{M_1\}$ such that $O'' = \langle 0.4017 \rangle \equiv N''$ end terminates returning the least portion $\beta_M = \frac{r - r''}{r} = 0.5983$ needed to be centrally controlled in order to enforce the overall Optimum O on M.

3 Optimality of algorithm `OpTop`

In the following Sections 3.1 and 3.2 we prove our main theorem:

Theorem 1 *Consider a system of parallel machines $M = \{M_1, \ldots, M_m\}$, with latency function $\ell_i(\cdot)$ per machine $M_i \in M$ differentiable and strictly increasing. Algorithm* `OpTop` *computes the least portion β_M of total flow r that a Leader must control to induce overall optimum cost on M.*

3.1 Useful machinery

We denote the corresponding *Nash* and *Optimum* assignments of flow r to system M as $N = \langle n_1, \ldots, n_m \rangle$ with $\sum_{i=1}^m n_i = r$, and $O = \langle o_1, \ldots, o_m \rangle$ with $\sum_{i=1}^m o_i = r$. We give a more useful definition for the Nash assignment N.

Definition 1 *An assignment $N = \langle n_1, \ldots, n_m \rangle$ of total flow $\sum_{i=1}^m n_i = r$ on the system of machines $M = \{M_1, \ldots, M_m\}$ is called* Nash Equilibrium *if there exists a constant $L^N > 0$ such that for each machine $M_i \in M$, if $n_i > 0$ then $\ell_i(n_i) = L^N$, otherwise $\ell_i(n_i) \geq L^N$.*

We denote as *Stackelberg* strategy S an assignment $S = \langle s_1, \ldots, s_m \rangle$ of flow $\sum_{i=1}^m s_i = \beta r,\ \beta \in [0,\ 1]$, on system M. Given S, we denote the *induced Nash* assignment as $T = \langle t_1, \ldots, t_m \rangle$ with $\sum_{i=1}^m t_i = (1 - \beta)r$.

Definition 2 *Given Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ with $\sum_{i=1}^m s_i = \beta r, \beta \in [0,\ 1]$, the assignment $T = \langle t_1, \ldots, t_m \rangle$ of the remaining flow $\sum_{i=1}^m t_i = (1-\beta)r$ on system $M = \{M_1, \ldots, M_m\}$ is an* Induced Nash Equilibrium *if there exists a constant $L^S > 0$ such that for each $M_i \in M$, if $t_i > 0$ then $\ell_i(t_i + s_i) = L^S$, otherwise $\ell_i(t_i + s_i) = \ell_i(0 + s_i) \geq L^S$.*

The *Cost* of an assignment $X = \langle x_1, \ldots, x_m \rangle$ on M equals $C(X) = \sum_{i=1}^m x_i \ell_i(x_i)$. Then, strategy S induces Nash assignment T with cost $C(S + T) = \sum_{i=1}^m (s_i + t_i)\ell_i(s_i + t_i)$.

Definition 3 *Machine $M_i \in M$ is called* over-loaded *(or* under-loaded*) if $n_i > o_i$ (or $n_i < o_i$), otherwise is called* optimum-loaded*, $i = 1, \ldots, m$.*

Definition 4 *Machine $M_i \in M$ (or load $s_i \in S$) is called* frozen *if Stackelberg strategy S assigns to it load $s_i \geq n_i,\ i = 1, \ldots, m$.*

Luckily, by the selfish assignment N of the users, all machines may end up optimum-loaded. In this way, $N \equiv O$ and the cost $C(N)$ of the system is minimized, that is $C(N) = C(O)$. In general

$N \not\equiv O$, since the selfish users prefer and thus over-load fast machines, while dislike and under-load slower ones, increasing the cost $C(N) > C(O)$. The crucial role of strategy S is to wisely pre-assign load $s_i \geq 0$ to each machine $M_i \in M$. This is successful to the extent that the induced selfish assignment T made by the users will assign an additional load $t_i \geq 0$ to each M_i, yielding the nice property $s_i + t_i = o_i$ for each $i = 1, \ldots, m$. Intuitively, strategy S *biasses* the initial Nash assignment N to the induced one T, in a way that $S + T \equiv O$, minimizing the induced overall cost $C(S + T) = C(O)$ of system M. It is convenient to state the following easy proposition.

Proposition 1 *Consider the machines in M with latency functions $\ell_j(\cdot)$ $j = 1, \ldots, m$. Let the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of total flow r on to M. If $N' = \langle n'_1, \ldots, n'_m \rangle$ is the Nash assignment of total flow $r' \leq r$ on to M, then for each machine $M_i \in M$ it holds: $n'_i \leq n_i$.*

Proof. Since N is a Nash assignment of the flow r on M, by Definition 1, $\exists\ L^N > 0$, such that for each machine $M_i \in M$ if $n_i > 0$ then $\ell_i(n_i) = L^N$, otherwise $\ell_i(n_i) = \ell_i(0) \geq L^N$. Let $M^{N^+} = \{M_i \in M : n_i > 0\}$ and $M^{N^-} = \{M_i \in M : n_i = 0\}$. Similarly for N', let $L^{N'} > 0$ the corresponding constant, and $M^{N'^+} = \{M_i \in M : n'_i > 0\}$ and $M^{N'^-} = \{M_i \in M : n'_i = 0\}$. To reach a contradiction, suppose that $\exists\ M_{i_0} \in M^{N'^+}$ such that $n'_{i_0} > n_{i_0}$.

Case 1: If $M_{i_0} \in M^{N^-}$ then $\ell_{i_0}(n'_{i_0}) = L^{N'} > \ell_{i_0}(n_{i_0}) = \ell_{i_0}(0) \geq L^N$, since each $\ell_i(\cdot)$ is strictly increasing and $n'_{i_0} > n_{i_0} = 0$. Then, each machine $M_i \in M^{N^+}$ must have load $n'_i > n_i$ under N', otherwise it will experience latency $\ell_i(n'_i) \leq \ell_i(n_i) = L^N < L^{N'}$ which is impossible, since N' is a Nash equilibrium. Therefore, we reach a contradiction since we get $r' \geq \sum_{M_i \in M^{N^+}} n'_i > \sum_{M_i \in M^{N^+}} n_i = r$.

Case 2: If $M_{i_0} \in M^{N^+}$ then $\ell_{i_0}(n'_{i_0}) = L^{N'} > \ell_{i_0}(n_{i_0}) = L^N$. Therefore, each machine $M_i \in M^{N^+}$ must receive load $n'_i > n_i$ under N', otherwise each $M_i \in M^{N^+}$ will experience latency $\ell_i(n'_i) \leq \ell_i(n_i) = L^N < L^{N'}$. That is, in the same fashion, we reach a contradiction. ∎

Theorem 2 describes each Stackelberg strategy S inducing Nash assignment T with cost $C(S+T) = C(N)$. In other words, Theorem 2 describes exactly all those useless strategies that induce cost indifferent form $C(N)$. Then, it is useless for `OpTop` to employ such a strategy S when trying to escape from a particular Nash equilibrium N with $C(N) >> C(O)$.

Theorem 2 *Consider the machines in M with latency functions $\ell_j(\cdot)$ $j = 1, \ldots, m$. Let the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the total flow r to M. Suppose that for a Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ with $\sum_{i=1}^m s_i = \beta r, \beta \in [0,\ 1]$, it holds $s_j \leq n_j$, $j = 1, \ldots, m$. Given S, let the induced Nash assignment $T = \langle t_1, \ldots, t_m \rangle$ of the remaining flow $(1-\beta)r$. Then it holds $n_j = s_j + t_j$ for each $M_j \in M$, in other words, $N \equiv S + T$.*

Proof. Since N is a Nash equilibrium on the machines in M with $\sum_{i=1}^m n_i = r$, then there exists a constant $L^N > 0$, such that for each machine $M_j \in M$ that receives load $n_j > 0$ it holds $\ell_j(n_j) = L^N$. That is, all loaded machines incur the same latency L^N to the system M. Consider an arbitrary Stackelberg strategy S, assigning load $s_j \leq n_j$ to each $M_j \in M$ with $\sum_{i=1}^m s_i = \beta r, \beta \in [0,\ 1]$. Then, the initial system of machines M is transformed by S to the equivalent system M^S, such that each machine $M_j^S \in M^S$ with load x_j now experiences latency $\ell_j^S(x_j) = \ell_j(x_j + s_j)$. Since for each $M_j \in M$ it holds $s_j \leq n_j$ then $\exists\ t_j \geq 0$ such that $t_j = n_j - s_j$ and also $\sum_{i=1}^m t_i = (1-\beta)r$. Let $T = \langle t_1, \ldots, t_m \rangle$ this assignment on M^S. Obviously, for the same constant $L^N > 0$ as above, it

holds: $\ell_j^S(t_j) = \ell_j\left((n_j - s_j) + s_j\right) = \ell_j(n_j) = L^N$, for each $M_j \in M$ with $t_j > 0$. This means that T is a Nash equilibrium on system M^S and also $S + T \equiv N$. ∎

In view of the negative result of Theorem 2, a natural question concerns the properties that a Stackelberg strategy must have in order to induce cost $\neq C(N)$. We answer this question on Theorem 3 and its generalization Lemma 1 below, and before this, we give a convenient definition.

Definition 5 *Each Stackelberg strategy S that satisfies Theorem 2 is called* useless-strategy*, otherwise is called* useful-strategy*.*

Theorem 3 states that any machine $M_i \in M$ receiving load $s_i \geq n_i$ by a strategy S (while there is no machine $M_j \in M$ with load $s_j < n_j$) will become non appealing for the subsequent selfish assignment T of the users. That is, for each $M_i \in M$ assigned load $s_i \geq n_i$, its induced load by the Nash assignment T equals $t_i = 0$. Intuitively, in the induced Nash equilibrium T, the dictated load $s_i \geq n_i$ by strategy S to machine M_i will remain "*frozen*" to $s_i,\ i \in \{1, \ldots, m\}$.

Theorem 3 *Let the Nash assignment $N = \langle n_1, \ldots, n_m\rangle$ of the total load r on system M. Suppose that for a Stackelberg strategy $S = \langle s_1, \ldots, s_m\rangle$ with $\sum_{i=1}^m s_i = \beta r$ we have either $s_j \geq n_j$ or $s_j = 0, j = 1, \ldots, m$. Then for the induced Nash assignment $T = \langle t_1, \ldots, t_m\rangle$ of the remaining load $(1-\beta)r$ we have that $t_j = 0$ for each $M_j \in M$ such that $s_j \geq n_j,\ j = 1, \ldots, m$.*

Proof. By Definition 1, since N is a Nash equilibrium on M, $\exists\ L^N > 0$ such that for each $M_i \in M$, if $n_i > 0$ then $\ell_i(n_i) = L^N$, otherwise $\ell_i(n_i) \geq L^N$. Fix a Stackelberg strategy S on M, such that for each machine $M_i \in M$, either $s_i \geq n_i$ or $s_i = 0$. Let $M^{S^+} = \{M_i \in M : s_i \geq n_i\}$ and $M^{S^-} = \{M_i \in M : s_i = 0\}$, and notice that $M = M^{S^+} \cup M^{S^-}$ and $M^{S^+} \cap M^{S^-} = \emptyset$. Each $M_i \in M^{S^+}$ receiving induced load $t_i \geq 0$ now experiences latency

$$\ell_i^{S^+}(t_i) = \ell_i(t_i + s_i) \geq \ell_i(s_i) \geq \ell_i(n_i) \geq L^N. \tag{1}$$

On the other hand, each $M_j \in M^{S^-}$ receiving induced load $t_j \geq 0$ experiences the *same* (since $s_j = 0$) as the *initial* (that is, without applying strategy S) latency

$$\ell_j^{S^-}(t_j) = \ell_j(t_j + s_j) = \ell_j(t_j). \tag{2}$$

In the sequel, the induced Nash assignment T by strategy S assigns the remaining flow on M

$$r - \sum_{i=1}^{m} s_i = r - \sum_{M_j \in M^{S^+}} s_j \leq \sum_{M_j \in M^{S^-}} n_j. \tag{3}$$

Having in mind (2) and (3), the crucial observation is that *even* if all the remaining flow that appears in LHS of (3) is assigned selfishly *only* on subsystem M^{S^-}, it is impossible the common latency L^{S^-} experienced by each loaded machine in M^{S^-} to become $L^{S^-} > L^N$, so that at least one machine in M^{S^+} to become appealing for the selfish players. More formally, let T^{S^-} be the *partial* Nash assignment that corresponds to assigning the flow that appears in LHS of (3) *only* on to the subsystem M^{S^-}. By Definition 1, $\exists\ L^{S^-} > 0$ such that for each loaded machine $M_j \in M^{S^-}$ with load $0 < t_j^{S^-} \leq n_j$ (here RHS inequality stems from Proposition 1 and the RHS of (3)) it holds

$$\ell_j^{S^-}(t_j^{S^-}) = \ell_j(t_j^{S^-}) = L^{S^-} \leq \ell_j(n_j) = L^N. \tag{4}$$

By (1) and (4) it follows that no machine $M_j \in M^{S^+}$ is appealing for the *overall* induced Nash assignment T. ∎

A crucial limitation of Theorem 3 is that it does not rule out the existence of a strategy S assigning load $s_j < n_j$ to some machine $M_j \in M$ in a way that at least one machine in M^{S^+} to become appealing for the selfish users. Lemma 1 rules out such a possibility. Intuitively, Lemma 1 states that each assignment of load $s_j \geq n_j$ made by strategy S to each machine $M_j \in M^{S^+}$ remains *unaffected* by its induced Nash load (i.e. T induces load $t_j = 0$ on M_j), *irrespectively* of any assignment of load $s_i < n_i$ made by S to any other machine $M_i \neq M_j$.

Lemma 1 *Let the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the total load r on system M. Suppose that for a Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ with $\sum_{i=1}^{m} s_i = \beta r, \beta \in [0,\ 1]$, we have either $s_j \geq n_j$ or $s_j < n_j, j = 1, \ldots, m$. Then for the induced Nash assignment $T = \langle t_1, \ldots, t_m \rangle$ of the remaining load $(1-\beta)r$ we have that $t_j = 0$ for each machine $M_j \in M$ such that $s_j \geq n_j,\ j = 1, \ldots, m$.*

Proof. By Definition 1, since N is a Nash equilibrium on M, $\exists\ L^N > 0$ such that for each machine $M_i \in M$, if $n_i > 0$ then $\ell_i(n_i) = L^N$, otherwise $\ell_i(n_i) \geq L^N$. Consider an arbitrary strategy S and let $M^{S^+} = \{M_i \in M : s_i \geq n_i\}$ and $M^{S^-} = \{M_i \in M : s_i < n_i\}$. Similarly as in (1), each $M_i \in M^{S^+}$ receiving induced load $t_i \geq 0$ now experiences latency

$$\ell_i^{S^+}(t_i) = \ell_i(t_i + s_i) \geq \ell_i(s_i) \geq \ell_i(n_i) \geq L^N. \tag{5}$$

However, here we do not have the nice fact as in (2) for the machine latencies in M^{S^-} (since now $s_j \neq 0$). We can circumvent this as follows. The induced Nash assignment T assigns on system M the remaining flow that equals

$$r^S = r - \sum_{M_i \in\ M^{S^+} \cup M^{S^-}}^{m} s_i \leq r^{S^-} = r - \sum_{M_i \in M^{S^+}} s_i \leq \sum_{M_i \in M^{S^-}} n_i, \tag{6}$$

where the rightmost inequality stems from the fact that

$$r^{S^+} = \sum_{M_i \in M^{S^+}} s_i \geq \sum_{M_i \in M^{S^+}} n_i. \tag{7}$$

Now, we prove that *even* if the flow r^{S^-} in (6) is scheduled selfishly *only* on the subsystem M^{S^-}, then all machines with load > 0 in it, would not experience common latency $L^{S^-} > L^N$ so that at least one machine in M^{S^+} to become appealing for scheduling any excess of flow. Let N^{S^-} the *partial* Nash assignment (that is, without previously assigning S on to subsystem M^{S^-}) when scheduling flow r^{S^-} appearing in (6) *only* on to subsystem M^{S^-}. Also, applying strategy S on to subsystem M^{S^-}, let T^{S^-} the *induced partial* Nash assignment (that is, by assigning previously S on to subsystem M^{S^-}) when scheduling the remaining of r^{S^-} appearing in (6) *only* on to subsystem M^{S^-}. Let $n_i^{S^-}$ (or $t_i^{S^-}$) denote the load assigned by N^{S^-} (or T^{S^-}) to each $M_i \in M^{S^-}$. Then, we have the following two cases.

Case 1: Suppose that for each machine $M_i \in M^{S^-}$ it holds $s_i \leq n_i^{S^-}$. Then we can apply Theorem 2 when assigning the remaining of r^{S^-} on subsystem M^{S^-}. In this way, for each machine $M_i \in M^{S^-}$ it holds $s_i + t_i^{S^-} = n_i^{S^-}$. Furthermore, from Inequality (6) we realize that

$$r^{S^-} = \sum_{M_i \in M^{S^-}} n_i^{S^-} \leq \sum_{M_i \in M^{S^-}} n_i. \tag{8}$$

Applying Proposition 1, we conclude that for each loaded machine $M_i \in M^{S^-}$ it holds $\ell_i(n_i^{S^-}) \leq \ell_i(n_i) = L^N$ and using (5) the lemma is proved.

Case 2: Suppose that there exists at least one machine $M_i \in M^{S^-}$ such that $s_i > n_i^{S^-}$. Then we can construct T^{S^-} as follows. For each machine $M_i \in M^{S^-}$ such that $s_i \leq n_i^{S^-}$ we set [2] $t_i^{S^-} \leq n_i^{S^-} - s_i$, otherwise we set $t_i^{S^-} = 0$. Clearly, each machine $M_i \in M^{S^-}$ with $t_i^{S^-} > 0$ experiences a common latency

$$L^{S^-} = \ell_i^{S^-}(t_i^{S^-}) \leq \ell_i((n_i^{S^-} - s_i) + s_i) = \ell_i(n_i^{S^-}) \leq \ell_i(n_i) = L^N,$$

and the lemma follows. On the other hand, each machine $M_i \in M^{S^-}$ with $t_i^{S^-} = 0$ already has load s_j due to S such that $n_j^{S^-} < s_j < n_j$. Therefore,

$$L^{S^-} \leq \ell_j^{S^-}(t_j^{S^-}) = \ell_j^{S^-}(0) = \ell_j(s_j) < \ell_j(n_j) = L^N,$$

and the lemma follows. ∎

In Section 3.2 we apply Theorem 3, Lemma 1 and Proposition 2 to discard the machines with frozen load $s_j = o_j \geq n_j$ and simplify the initial game. Clearly, such machines will never be affected by the induced selfish play of the users on the rest of machines. Therefore, using Proposition 2, we focus on the remaining machines with load under S that equals $s_i < n_i$, which may be affected by the selfish users, trying to find a subsequent partial Stackelberg strategy on them that will induce the optimum cost.

Proposition 2 *Let the system $M = \{M_1, \ldots, M_m\}$ and the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the total load r on M. Fix a Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ such that either $s_j \geq n_j$ or $s_j < n_j, j = 1, \ldots, m$. Let the subset of frozen machines $M^{S^+} = \{M_j \in M : \ s_j \geq n_j\}$, $j = 1, \ldots, m$, and their frozen load $r^{S^+} = \sum_{M_j \in M^{S^+}} s_j$. Then the initial Stackelberg game of flow r on M can be simplified to scheduling the remaining unfrozen flow $r^{S^-} = r - r^{S^+}$ to the remaining unfrozen subsystem of machines $M^{S^-} = M \setminus M^{S^+}$.*

3.2 The optimal evolution of `OpTop`

3.2.1 Phase 1: `OpTop` loads optimally all initially under-loaded machines.

During Phase $i \geq 1$, let $N^i = \langle n_1^i, \ldots, n_m^i \rangle$ denote the Nash assignment of flow r^i (where r^1 equals the initial total flow r) on to subsystem of machines M^i (where M^1 is the initial system M). Also, let $O = \langle o_1, \ldots, o_m \rangle$ denote the overall optimum assignment of flow r onto system M (notice that O is not parameterized with respect to the ith Phase). We introduce the following partition (according to Definition 3) of the system $M^1 \equiv M$ of machines, during Phase 1

$$M^{1^-} = \{M_j \in M^1 : \ n_j^1 < o_j\} \text{ and } M^{1^+} = \{M_j \in M^1 : \ n_j^1 \geq o_j\}. \tag{9}$$

- According to Theorem 3 and Lemma 1, if during Phase 1 a Stackelberg strategy $S^1 = \langle s_1^1, \ldots, s_m^1 \rangle$ assigns load s_j^1 such that $o_j < n_j^1 < s_j^1$ to at least one over-loaded machine $M_j \in M^{1^+}$ (see Definition 3) then M_j will remain frozen to an unfavorably high value $s_j^1 > o_j$,

[2] See on Appendix the validity of this inequality.

irrespectively of any load s_i^1 strategy S^1 may assign to any other machine $M_i \neq M_j$. Therefore, M_j will never reduce its load to the optimum value o_j, and thus the system M will never converge to its overall optimum assignment O.

- In the same fashion, applying Lemma 1, if during PHASE 1 strategy S^1 assigns load s_j^1 such that $n_j^1 < s_j^1 < o_j$ to at least one under-loaded machine $M_j \in M^{1^-}$ then M_j will remain frozen to an unfavorably low load $s_j^1 < o_j$.

Then `OpTop` must assign load $s_j^1 = o_j > n_j^1$ to each initially under-loaded machine $M_j \in M^{1^-}$, otherwise under-loaded machines will never attain their overall optimum load. Furthermore, by Theorems 2, 3 and Lemma 1, it is wasteful any assignment of flow $s_i^1 < n_i^1$ to any other machine $M_i \in M^1$. Clearly, no such assignment $o_i < s_i^1 < n_i^1$ can affect favorably any load assignment $s_j^1 = o_j > n_j^1$ to any initially under-loaded machine $M_j \in M^{1^-}$. We conclude that at the end of PHASE 1 algorithm `OpTop` constructs the Stackelberg strategy S^1 such that in each $M_j \in M^{1^-}$ it assigns load $s_j^1 = o_j$, while in each $M_j \in M^{1^+}$ it assigns $s_j^1 = 0$.

Simplification of the initial game: Each initially under-loaded machine $M_j \in M^{1^-}$ will remain frozen to its induced by S^1 optimum load $s_j^1 = o_j > n_j^1$. Using Proposition 2, we can simplify the game by discarding each initially under-loaded machine $M_j \in M^{1^-}$ that becomes frozen by S^1. During PHASE 1 algorithm `OpTop` needs a portion r^{1^-} to frozen the machines in M^{1^-} $r^{1^-} = \sum_{M_j \in M^{1^-}} o_j$. Then the initial Stackelberg game of flow r^1 on M^1 can be simplified to scheduling the remaining flow $r^2 = r^1 - r^{1^-}$ to the remaining machines $M^2 = M^1 \setminus M^{1^-}$.

3.2.2 Phase $i \geq 2$: the recursive nature of `OpTop`.

During PHASE 2, we consider the *Nash* assignment $N^2 = \langle n_j^2 : M_j \in M^2 \rangle$ when scheduling the remaining flow $r^2 = r^1 - r^{1^-}$ on the simplified system $M^2 = M^1 \setminus M^{1^-}$ and, similarly as in PHASE 1 let,

$$M^{2^-} = \{M_j \in M^2 : \ n_j^2 < o_j\} \text{ and } M^{2^+} = \{M_j \in M^2 : \ n_j^2 \geq o_j\}. \tag{10}$$

Then, applying similarly as in Section 3.2.1 Theorem 2 and 3, Lemma 1 and Proposition 1, `OpTop` constructs the subsequent *Stackelberg* strategy S^2 onto subsystem M^2 such that in each $M_j \in M^{2^-}$ it assigns $s_j^2 = o_j$, while in each $M_j \in M^{2^+}$ it assigns $s_j^2 = 0$. Then, once more, `OpTop` simplifies the game, scheduling flow $r^3 = r^2 - r^{2^-} = r^2 - \sum_{M_j \in M^{2^-}} o_j$ onto subsystem $M^3 = M^2 \setminus M^{2^-}$. Finally, `OpTop` terminates as soon as it reaches a PHASE i_0 where the simplified subsystem M^{i_0} has the property

$$M^{i_0{}^-} = \{M_j \in M^{i_0} : \ n_j^{i_0} < o_j\} \equiv \emptyset, \tag{11}$$

and outputs the least possible flow $\beta(M)$ needed to impose the overall optimum on system M that equals

$$\beta(M) = \frac{\sum_{k=1}^{i_0-1} r^{k^-}}{r^1} = \frac{r^1 - r^{i_0}}{r^1}, \ i_0 \geq 1.$$

Discussion

The efficiency of `OpTop` depends solely on the computation of Nash and Optimum assignment on a given system M. In [20] a $O(m^2)$ time algorithm is presented for computing both Nash and Optimum assignment for linear latency functions. In the Appendix we present a similar $O(m^2)$ time algorithm that computes the Nash assignment for M/M/1 latencies while the optimum one can be computed as in [7] using standard optimization techniques from Nonlinear Programming. Therefore, for such latencies `OpTop` runs in $O(m^3)$ time. In addition, for a more general class of latency functions we can consider the approaches as in [1].

References

[1] V. S. Anil Kumar, Madhav V. Marathe. Improved Results for Stackelberg Scheduling Strategies. *In Proceedings of the 29th International Colloquium on Automata,Languages, and Programming*, 2002, pp. 776-787

[2] T. Basar, G. J. Olsder.*Dynamic Noncooperative Game Theory*. SIAM, 1999

[3] A. Czumaj and B. Voecking. Tight bounds for worst-case equilibria. *In Proceedings of the 13th Annual Symposium on Discrete Algorithms*, 2002.

[4] A. Czumaj. *Selfish Routing on the Internet*, Handbook of Scheduling: Algorithms, Models, and Performance Analysis,CRC Press, 2004

[5] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, May 2000.

[6] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *In Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pp 123–134. Springer-Verlag, 2002

[7] Y.A. Korilis, A.A. Lazar, A. Orda: Achieving network optima using stackelberg routing strategies, *In Proceedings of the IEEE/ACM Transactions of Networking*. Extended version http://citeseer.ist.psu.edu/221983.html, 1997

[8] Y. A. Korilis, A. A. Lazar and A. Orda. The designer's perspective to noncooperative networks. *In Proceedings of the IEEE INFOCOM 95*, Boston, MA, April 1995.

[9] Y.A. Korilis, A.A. Lazar, A. Orda: Capacity allocation under noncooperative routing, *In Proceedings of the IEEE/Transactions on Automatic Control*, 1997

[10] E. Koutsoupias and C. Papadimitriou. Worst-case Equilibria. *In Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pp. 387–396, Vol. 1563, Lecture Notes in Computer Science, Springer-Verlag, Trier, Germany, March 1999

[11] M. Mavronikolas and P. Spirakis: The price of Selfish Routing,*In Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, 2001

[12] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991

[13] N. Nisan and A. Ronen. Algorithmic mechanism design.*In Proceedings of the 31st ACM Symposium on Theory of Computing*, 1999, pp 129-140

[14] N. Nisan. Algorithms for selfish agents: Mechanism design for distributed computation. *In Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 1-15, 1999.

[15] M. J. Osborne, A. Rubinstein. *A course in Game Theory*, MIT Press

[16] G. Owen. *Game Theory*. Academic Press. Orlando, FL, 3rd Edition, 1995.

[17] C. Papadimitriou. Game Theory and Mathematical Economics: A Theoratical Computer Scientist's Introduction. *In Proceedings of the 42nd IEEE symposium on Foundations of Computer Science (FOCS '01)*,2001

[18] C. Papadimitriou. Algorithms, Games, and the Internet. *In Proceedings of the 33rd Symposium on Theory of Computing*, ACM Press, New York, pp 749-753, 2001

[19] T. Roughgarden. The price of anarchy is independent of the network topology. *In Proceedings of the 34th ACM Symposium on the Theory of Computing*, 2002. pp 428-437. 105

[20] T. Roughgarden. Stackelberg scheduling strategies. *In Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pp 104-113, 2001.

[21] T. Roughgarden. Stackelberg scheduling strategies. *In Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pp 104-113, 2001 Slides presentation http://theory.stanford.edu/∼tim/slides/`stack_cornell.pdf`

[22] T. Roughgarden, E. Tardos. How bad is Selfish Routing?. *In Proceedings of the 41st Annual Symponsium on Foundations of Computer Science*,pp 93-102, 2000

[23] T. Roughgarden. Designing networks for selfish users is hard. *In Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pp 472–481, 2001.

[24] H. von Stackelberg. *Marktform aund Gleichgewicht*. Springer-Verlag, 1934

[25] http://students.ceid.upatras.gr/∼ politop/opTop, opTop Experimental results.

Appendix

Experimental validation of optimality

In this section we present our results which show that the `OpTop` algorithm computes the least possible fraction β of the total flow r the leader should control to induce optimal assignment to the set M of machines. To do so, we computed using the algorithm `Exhaustive` below the corresponding minimum portion β of flow, for random 3-tuples machines for both linear and M/M/1 functions.

Algorithm: Exhaustive (M, r)
Input: `Machines` $M = \{M_1, M_2, M_3\}$, `flow rate` r
Output: `A portion` β `of flow rate` r
begin:
  `Compute the Optimum assignment` $O := \langle o_1, o_2, o_3 \rangle$ `of flow` r `on machines` M;
  `Compute the Cost` C_{opt} `of the Optimum assignment` O;
  for $r_0 = 0$ **to** r **do:**
    for $x = 0$ **to** r_0 **do:**
      for $y = 0$ **to** $r_0 - x$ **do:**
        $S := \langle x, y, r_0 - x - y \rangle$

```
            Compute the induced Nash assignment N := ⟨n1, n2, n3⟩
                of flow r − r0 on machines M and leader assignment S;
            Compute the Cost C_ind of the induced Nash assignment N;
            If (C_opt == C_ind) return β = r0/r; break; end if
        end for;
    end for;
  end for;
end;
```

We made extensive experiments both 3-tuples of machines with both linear and M/M/1 latency functions and the results stated that the `OpTop` algorithm computes the least possible portion of flow that should be controlled by the leader. An interested reader can find our experimental results in [25].

Computing *Nash Equilibria* on systems with M/M/1 latency functions

We compute the Nash equilibrium of total flow r on a system M of machines with M/M/1 latency functions by using a similar algorithm with the one presented in [20]. Let M be the set of machines with latency functions $\ell_i(x_i) = \frac{1}{c_i - x_i}$, with $c_i > 0$, $i \in M$. The selfish users prefer the machines of highest capacity, so we order the machines from the highest c_1 to the smallest one c_m as: $c_1 \geq c_2 \geq \cdots \geq c_m$.

Our goal is to understand the structure of the Nash assignment N in relation to the flow rate r. We construct the Nash assignment for 4 machines in M, using the Figure 2 in Appendix and afterwards we generalize for m machines in M. While the portion x of the total flow r which is currently scheduled on M increases, and remains $x \in [0,\ c_1 - c_2)$ then $\forall\ i \geq 2,\ \ell_1(x) < \ell_i(0)$, that is, M_1 remains most appealing. Since M_1 receives flow, it becomes loaded reducing its capacity and increasing its latency. After a while, the latency in M_1 becomes equal to the latency experienced in M_2. This happens as soon as M_1 receives flow $x_1 = c_1 - c_2$, which gives $\ell_1(x_1) = \ell_2(0)$. At this point Phase 1 in Figure 2 in Appendix ends. Now, it starts Phase 2 depicted in Figure 2(phase:2) and x accumulates further. During this phase, for each $x \in [c_1 - c_2,\ c_2 - c_3)$ it holds: $\forall\ i \geq 3,\ \ell_1(x) = \ell_2(x) < \ell_i(0)$. That is, M_1 and M_2 are *equally appealing* while M_3 and M_4 remain unfavorable. As soon as x reaches $(c_2 - c_3)$ then M_3 becomes equally attractive for the selfish users as M_1 and M_2. Here Phase 2 ends and Phase 3 starts. That is, for each $x \in [c_2 - c_3,\ c_3 - c_4)$ it holds $\ell_1(x) = \ell_2(x) = \ell_3(x) < \ell_3(0)$, which means that any additional portion x of flow is assigned equally to M_1,M_2 and M_3 till M_4 becomes also appealing to the users.

In a more general setting of m machines in M the Nash assignment is constructed in phases: at the end of Phase i the jobs are assigned to the i first machines and the M_{i+1} machine is ready to receive jobs if there are any left. Formally the Nash assignment N can be constructed using the following: For $i = 1, \ldots, m-1$, let u_i the m-vector $((c_1 - c_2), (c_2 - c_3), \ldots, (c_{i-1} - c_i), 0, \ldots, 0) \in R^m$ and $u_m = \left(\frac{(r - \sum_{i=1}^{m-1} u_i)}{m}, \ldots, \frac{(r - \sum_{i=1}^{m-1} u_i)}{m}\right)$. The vector u_i should be interpreted as specification of the way jobs are assigned to the first i machines during Phase i. Next, we define the vector δ_i for $i = 1, \ldots, m$ as follows: We find the machine M_k for which it holds that: $\sum_{n=1}^{k} u_n - r < k\ (c_{k-1} - c_k)$. Then the vector δ is $\delta_i = 1$ for $i = 1, \ldots, k-1$, $\delta_k = \frac{\sum_{n=1}^{k} u_n - r}{k^2\ (c_{k-1} - c_k)}$ and $\delta_i = 0$ for $i = k+1, \ldots, m$

The scalar δ_i should be interpreted as the portion of jobs that could be assigned to the machines during PHASE i. Then we can describe the N Nash assignment as follows:

Lemma 2 *Let an instance I of set of machines M with M/M/1 latency functions and flow rate r. The Nash assignment N for I is given by:*

$$N = \sum_{i=1}^{m} \delta_i \cdot u_i \tag{12}$$

Figures

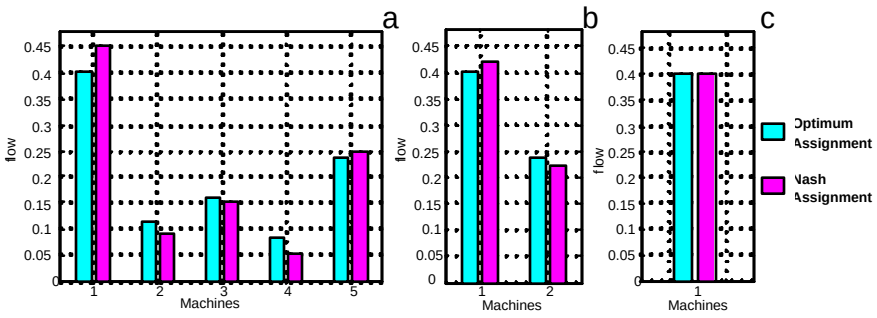

Figure 1: (a) Optimum and Nash assignments of the set $M = \{M_1, M_2, M_3, M_4, M_5\}$. (b) Optimum and Nash assignments on the set $M' = \{M_1, M_5\}$. (c) Optimum and Nash assignments on the set $M'' = \{M_1\}$

.

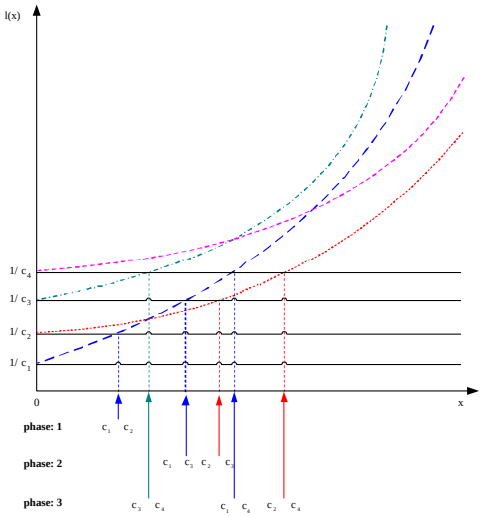

Figure 2: Calculation of Nash assignment for M/M/1 latency functions

Lemma 1, Case 2

Given strategy S, let the subsystem $M_0^{S^-} \subseteq M^{S^-}$ containing all machines $M_i \in M^{S^-}$ such that $s_i > n_i^{S^-}$. Consider strategy S' such that on each $M_i \in M^{S^-} \setminus M_0^{S^-}$ it assigns load $s'_i = s_i$ and for each $M_0^{S^-}$ it assigns load $s'_i = s_i - (s_i - n_i^{S^-}) = n_i^{S^-}$ (that is, it subtracts load $(s_i - n_i^{S^-})$). In this way we get

$$\sum_{M_i \in M^{S^-}} s'_i < \sum_{M_i \in M^{S^-}} s_i. \tag{13}$$

Given strategy S', Theorem 2 applies on assigning selfishly the flow that appears on the RHS of (14) onto subsystem M^{S^-}, and let $T^{S'^-}$ the corresponding induced Nash assignment. Then, for each machine $M_i \in M^{S^-}$ it holds $t_i^{S'^-} = n_i^{S^-} - s'_i$, and *most* importantly, each $M_i \in M_0^{S^-}$ gets induced load $t_i^{S'^-} = 0$. The crucial observation is that if we *add back* the subtracted load $(s_i - n_i^{S^-})$ on each $M_i \in M_0^{S^-}$ then (i) strategy S' *becomes* S and (ii) each $M_i \in M_0^{S^-}$ becomes *even* less appealing (recall $t_i^{S'^-} = 0$ under S'). Furthermore, given strategy S, let T^{S^-} the induced Nash assignment of the flow that appears in the LHS of (14)

$$\sum_{M_i \in M^{S^-}} t_i^{S^-} = r^{S^-} - \sum_{M_i \in M^{S^-}} s_i < \sum_{M_i \in M^{S^-}} t_i^{S'^-} = r^{S^-} - \sum_{M_i \in M^{S^-}} s'_i, \tag{14}$$

on subsystem M^{S^-}. From Proposition (1), on selfisly assigning the flow in LHS of (14) onto subsystem $M^{S^-} \setminus M_0^{S^-}$, we conclude that each $M_i \in M^{S^-} \setminus M_0^{S^-}$ now receives flow

$$t_i^{S^-} \leq t_i^{S'^-} = n_i^{S^-} - s'_i = n_i^{S^-} - s_i,$$

while each $M_i \in M_0^{S^-}$ now receives $t_i^{S^-} = 0$, as it was claimed in the Proof of Lemma 1, Case 2.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'