Stackelberg Scheduling Strategies
(DRAFT – DO NOT DISTRIBUTE)

Tim Roughgarden*

November 22, 2000

Abstract

We study the problem of optimizing the performance of a system shared by selfish, noncooperative users. We consider the concrete setting of scheduling jobs on a set of shared machines with load-dependent latency functions specifying the length of time necessary to complete a job; we measure system performance by the *total latency* of the system.

Assigning jobs according to the selfish interests of individual users (who wish to minimize only the latency that their own jobs experience) typically results in suboptimal system performance. However, in many systems of this type there is a mixture of "selfishly controlled" and "centrally controlled" jobs; as the assignment of centrally controlled jobs will influence the subsequent actions by selfish users, we aspire to contain the degradation in system performance due to selfish behavior by scheduling the centrally controlled jobs in the best possible way.

We formulate this goal as an optimization problem via *Stackelberg games*, games in which one player acts a *leader* (here, the centralized authority interested in optimizing system performance) and the rest as *followers* (the selfish users). The problem is then to compute a strategy for the leader (a *Stackelberg strategy*) that induces the followers to react in a way that (at least approximately) minimizes the total latency in the system.

In this paper, we prove that it is NP-hard to compute the optimal Stackelberg strategy and present simple strategies with provable performance guarantees. More precisely, we give a simple algorithm that computes a strategy inducing a job assignment with total latency no more than a constant times that of the optimal assignment of all of the jobs; in the absence of centrally controlled jobs and a Stackelberg strategy, no result of this type is possible. We also prove stronger performance guarantees in the special case where every machine latency function is linear in the machine load.

*Department of Computer Science, Cornell University, Ithaca NY 14853. Supported by an NSF Graduate Fellowship, a Cornell University Fellowship, and ONR grant N00014-98-1-0589. Email: `timr@cs.cornell.edu`.

1 Introduction

Coping with selfishness. One of the most basic problems arising in the management of a set of resources is that of optimizing system performance. A concrete example of such a problem is as follows: given a large set jobs to be assigned to a set of machines with load-dependent latency functions (specifying the length of time necessary to complete a job), find the allocation of jobs to machines minimizing the *total latency* of the system.

In many such systems, including the Internet and other large-scale communication networks, there is no central authority controlling the allocation of shared resources; instead, system users are free to act is a selfish manner [5]. This observation has led many authors (e.g., [3, 8, 16, 19, 22, 26, 28]) to model the behavior of users in such a system by a *noncooperative game* and to study the resulting *Nash equilibria* (see [23] for an introduction to basic game-theoretic concepts).

Motivated by the well-known fact that Nash equilibria may be *inefficient* (i.e., they need not optimize system performance [11, 23]), researchers have proposed several different approaches for *coping with selfishness* – that is, for ensuring that selfish behavior results in a desirable outcome. For example, two recent papers by Korlis et al. [17, 18] give methods for improving system performance by adding additional capacity to system resources, Cocchi et al. [7] investigate the control of selfish users through various pricing policies, and Shenker [28] demonstrates that an appropriate (centralized) protocol at a network switch induces selfish users to exhibit "good" flow control behavior. An area of research that typically does not study Nash equilibria but is nevertheless concerned with controlling selfish behavior is that of *algorithmic mechanism design* [13, 20, 21, 25]. In this setting, an algorithm is designed to collect data from users and compute an outcome using this data (e.g., based on the bids of the users of the system, compute the set of users that will receive some good, such as a movie multicast over the Internet). The difficulty in these problems stems from the assumptions that the algorithm is publicly known and that users are selfish and may report false data (if doing so improves their personal objective function); this problem is typically resolved via a *payment scheme* (i.e., after computing an outcome, the algorithm distributes payments to users according to the outcome and the data collected) that induces all users to report truthful data.

In this paper, we pursue a different approach. In many systems, there will be a mix of "selfishly controlled" and "centrally controlled" jobs – i.e., the shared resource is used by both selfish individuals and some central authority (two examples are given below). We investigate the following question: given such a system, *how can centrally controlled jobs best be assigned to resources so as to induce "good" (albeit selfish) behavior from the noncooperative users?* This approach has several appealing aspects: no communication is required between the system users and an algorithm, no notion of currency is needed, and the assignment of centrally controlled jobs is easily modified to adapt to evolving usage patterns.

One example of such a system arises in networks that allow a large customer to set up a so-called *virtual private network* consisting of guaranteed and preassigned virtual paths for ongoing use [4, 16]. The bandwidth needed for the virtual private network may be viewed as centrally controlled (its routes may be chosen by the network manager) while individual users of the network continue to behave in a selfish and independent fashion. As another example, we note that the owner of a large network will typically require network bandwidth for any number of administrative tasks, such as checking for hardware failures, measuring response time between different regions of the network, updating routing tables, and so on [6, 15]; here again, the shared resource is used both by selfish individuals and by some central authority.

Stackelberg games. We are thus led to consider a type of game in which the roles of different players are asymmetric. One player (responsible for assigning the centrally controlled jobs to resources and interested in optimizing social welfare) acts as a *leader*, in that it may hold its assignment (i.e., its *strategy*) fixed while all other agents (the *followers*) react independently and selfishly to the leader's strategy, reaching a Nash equilibrium relative to the leader's strategy. These types of games, called *Stackelberg games*, and the resulting *Stackelberg equilibria* have been well-studied in the game theory literature (see, e.g., [1, §2.3] or [2, §3.6] for an introduction) and have been previously studied in the contexts of both competitive facility location [24] and networking [9, 10, 12, 16]. With the exception of [16], however, the leader/follower hierarchy has been used to model classes of selfish agents with different priority levels; this setting differs from ours in that no agent is interested in optimizing system performance. The paper of Korlis et al. [16], while more similar in spirit to ours, focuses on deriving necessary and sufficient conditions (on the number of selfish users, the fraction of jobs that are centrally controlled, etc.) for the existence of a leader strategy inducing an optimal assignment of jobs to resources; moreover, only one type of latency function is considered. By contrast, we are interested in simple leader strategies that *always* induce optimal or near-optimal behavior from the system users for *any* set of latency functions.

Problems studied in this paper. We focus on the problem described at the beginning of the paper, of scheduling jobs on a set of machines with load-dependent latencies in order to minimize the total latency. In addition to being one of the most commonly studied models [12, 16, 19, 20, 21, 25] (occasionally in the equivalent formulation of routing on a set of parallel links), the simple setting of scheduling jobs on machines permits a study of the effects of different leader strategies in Stackelberg games without any additional complications, such as the issue of path-selection in a complicated network. We also focus on a scenario in which there is a large number of jobs, each of very small size. This assumption is consistent with a large body of existing literature on Nash equilibria in congested systems (e.g., [3, 8, 26, 27]) and allows us to model assignments of jobs to m machines in a continuous way (e.g., by a vector in $\mathcal{R}^m$); this in turn implies the existence and essential uniqueness of the equilibrium reached by selfish users relative to any Stackelberg strategy.

We may now restate our central questions quantitatively:

(1) among all leader strategies for a given set of machines and jobs, can we characterize and/or compute the strategy inducing the *Stackelberg equilibrium* – i.e., the equilibrium of minimum total latency?

(2) what is the worst-case ratio between the total latency of the Stackelberg equilibrium and that of the optimal assignment of jobs to machines?

Our results. We give a simple polynomial-time algorithm for computing a leader strategy that induces an equilibrium with total latency no more than $\frac{1}{\alpha}$ times that of the optimal assignment of jobs to machines, where α denotes the fraction of the jobs that are centrally controlled. We also exhibit, for each α, an instance in which *no* strategy can achieve a better performance guarantee. This result stands in sharp contrast to known results about *Nash* equilibria in this model; in particular, the total latency of the Nash equilibrium may be arbitrarily worse than that of the optimal assignment of jobs to machines [26].

In the well-studied special case where every machine possesses a latency function linear in the congestion, we give a simple $O(m^2)$ algorithm for computing a strategy inducing an equilibrium with total latency no worse than $\frac{4}{3+\alpha}$ times the optimal (where α is the fraction of centrally controlled jobs, and m is the number of machines). We again give instances in which no strategy can provide a stronger guarantee.

Finally, we consider the optimization problem of computing the strategy inducing the Stackelberg equilibrium and show that it is NP-hard, even in the special case where every latency function is linear.

We note that our results give a (sharp) trade-off between the optimal assignment and the Nash equilibrium (as a function of the fraction of centrally controlled jobs) in the following sense. In previous work by the author and Éva Tardos [26] (motivated by the paper of Koutsoupias and Papadimitriou [19]) it is shown that in general the Nash equilibrium can be arbitrarily more costly than the optimal assignment, but if every machine latency function is linear then the total latency of the Nash equilibrium is no more than $\frac{4}{3}$ times that of the optimal assignment. Thus, our results reduce to those of [26] when $\alpha = 0$, give the trivial result that the Stackelberg equilibrium for $\alpha = 1$ is the optimal assignment, and quantifies the worst possible ratio between the cost of the Stackelberg equilibrium (in some sense, a "mixture" of the Nash equilibrium and optimal assignment) and the cost of the optimal assignment for all intermediate values of α.

Our approach also adds an algorithmic dimension to the existing studies comparing Nash equilibria and optimal solutions [19, 26], in that one aspect of our analysis of Stackelberg equilibria is the design of algorithms for efficiently computing good Stackelberg strategies. Further, while Nash and optimal assignments can be characterized and computed efficiently via convex programming [8, 26], our hardness result for computing Stackelberg equilibria implies that no such characterization is possible. With the central approach of [26] ruled out, new techniques are required for our results.

2 Preliminaries

2.1 The Model

We consider a set M of m machines $1, 2, \ldots, m$, where machine i is endowed with a *latency function* $\ell_i(\cdot)$ that measures the (load-dependent) time required to complete a job. We require that each latency function be nonnegative, continuous, and nondecreasing in its argument. We also impose the very weak condition that $x \cdot \ell_i(x)$ is a weakly convex function for each i.[1] We assume a *rate* r of job arrivals; an *assignment* of the jobs to the machines is an m-vector $x \in \mathcal{R}_+^m$ such that $\sum_{i=1}^m x_i = r$. When we are interested in the total load on a subset $M' \subseteq M$ of the machines, we write $x(M') = \sum_{i \in M'} x_i$. We measure system performance via the *cost* of *total latency* $C(x)$ of an assignment x, defined by $C(x) = \sum_{i=1}^m x_i \ell_i(x_i)$. We note that all jobs assigned to the same machine experience the same latency; this differs from much of the traditional scheduling literature but agrees with common models of equilibria in congested systems (e.g., [8, 16, 22, 26]), where a particular allocation of resources represents a "steady-state solution" with jobs arriving continuously over time.

We will consider instances with and without centrally controlled jobs. We denote an instance with machines M, rate r, and no centrally controlled jobs by (M, r). An instance with centrally controlled jobs (a *Stackelberg instance*) will be denoted by (M, r, α), where the third parameter $\alpha \in (0, 1)$ indicates the fraction of the overall traffic that is centrally controlled.

2.2 Nash Equilibria and Optimal Assignments

If jobs are generated and assigned to machines by selfish, noncooperative agents (who wish to minimize the amount of time it takes for their work to complete), we expect the assignment to be "stable" or "at equilibrium" in the following sense: no job can strictly decrease the latency it experiences by changing machines. The following definition is motivated by this notion of a stable assignment by noncooperative agents.

Definition 2.1 *An assignment x to M is* at Nash equilibrium *(or is a* Nash assignment*) if whenever $i, j \in M$ with $x_i > 0$, $\ell_i(x_i) \leq \ell_j(x_j)$.*

In particular, all machines in use by an assignment at Nash equilibrium have the same latency. We may thus express the cost of an assignment at Nash equilibrium in the following simple form.

Lemma 2.2 *If x is an assignment at Nash equilibrium for (M, r) such that all machines in use have common latency L, then*

$$C(x) = rL.$$

[1]Thus, $\ell_i(x)$ may be any weakly convex function, or $\log(1 + x)$, etc.

It is worth noting that an assignment at Nash equilibrium does not in general optimize the system performance. To see this, consider a two-machine example, in which the first machine has constant latency function $\ell_1(x) = 1$ and the second has latency function $\ell_2(x) = x$. If we put $r = 1$, we see that the (optimal) assignment $(\frac{1}{2}, \frac{1}{2})$ has total latency $\frac{3}{4}$ whereas the (unique) assignment at Nash equilibrium assigns all work to the second machine, thereby incurring a total cost of 1.

We end our preliminary discussion of Nash assignments by noting that they exist and are essentially unique.

Lemma 2.3 ([3, 8, 26]) *Suppose M is a set of machines with continuous, nondecreasing latency functions. Then:*

(a) For any rate $r \geq 0$ of job traffic, there exists an assignment of jobs to M at Nash equilibrium.

(b) Any two Nash assignments for an instance (M, r) have equal cost.

An analogous characterization of optimal assignments can also be given; to do this, we introduce a bit of new notation. Define $\ell_i^*(x)$ by $(x \cdot \ell_i(x))' = \ell_i + x \cdot \ell_i'(x)$ – i.e., as the marginal cost of increasing the load of machine i. We will say that $\ell_i^*(x_i)$ is the *gradient of machine* i (with respect to the assignment x). Then, the following lemma holds.

Lemma 2.4 ([8, 26]) *Suppose $x_i \cdot \ell_i(x_i)$ is a weakly convex function for $i = 1, \ldots, m$. Then an assignment x to M is optimal if and only if whenever $i, j \in M$ with $x_i > 0$, $\ell_i^*(x_i) \leq \ell_j^*(x_j)$. Moreover, the optimal assignment can be computed in polynomial time.*

We conclude with a final observation (needed in Section 4) relating the latencies of machines with respect to the Nash and optimal assignments. The following lemma follows from the definition of a Nash assignment and the obvious fact that if $\bar{x}, x^*$ are distinct assignments at the same rate then there is a machine i with $x_i^* > \bar{x}_i$.

Lemma 2.5 *Let (M, r) be an instance with latency functions ℓ, admitting a Nash assignment $\bar{x}$ and an optimal assignment x^*. Let L be the common latency (w.r.t. $\bar{x}$) of every machine used by $\bar{x}$. Then there exists a machine i with $\ell_i(x_i^*) \geq L$.*

2.3 Stackelberg Strategies and Induced Equilibria

In this subsection we define our notion of a Stackelberg game and consider two examples. Recall we desire a *hierarchical* game, where a *leader* assigns centrally controlled jobs to machines and, holding this strategy fixed, the selfish users of the system react in a noncooperative and selfish manner. This idea is formalized in the next two definitions.

Definition 2.6 *A* (Stackelberg) strategy *for the Stackelberg instance (M, r, α) is an assignment feasible for $(M, \alpha r)$.*

Definition 2.7 *Let s be a strategy for Stackelberg instance (M, r, α) where machine i has latency function ℓ_i, and let $\tilde{\ell}_i(x) = \ell_i(s_i + x)$ for each $i \in M$. An* equilibrium induced by strategy *s is an assignment t at Nash equilibrium for the instance $(M, (1 - \alpha)r)$ w.r.t. latency functions $\tilde{\ell}$. We then say that $s + t$ is an* assignment induced by *s for (M, r, α).*

The next fact is immediate from Lemma 2.3.

Lemma 2.8 *Let s be a strategy for a Stackelberg instance with continuous, nondecreasing latency functions. Then there exists an assignment induced by s, and any two such induced assignments have equal cost.*

The following simple observation will be useful in sections 4 and 5.

Lemma 2.9 *Let s be a strategy for Stackelberg instance (M, r, α) inducing equilibrium t. Let M' denote the machines on which $t_i > 0$. Then $s + t$, restricted to M', is an assignment at Nash equilibrium for the instance $(M', s(M') + t(M'))$. In particular, all machines on which $t_i > 0$ have a common latency w.r.t. $s + t$.*

We next consider two examples that demonstrate both the usefulness and the limitations of Stackelberg strategies. First consider the two machine example of the previous subsection (one machine with latency function $\ell_1(x) = 1$, the other with latency function $\ell_2(x) = x$). Recall that in the absence of any jobs under centralized control, the assignment at Nash equilibrium is $\frac{4}{3}$ as costly as the optimal assignment. Suppose that half of the jobs are controlled by the system manager (i.e., $\alpha = \frac{1}{2}$) and consider the strategy $s = (\frac{1}{2}, 0)$. Then, as all remaining jobs will be assigned to the second machine in the equilibrium induced by s, the assignment induced by s is precisely the optimal assignment. Thus, in this particular instance, system performance can be optimized via a Stackelberg strategy.

Now consider a small modification to the previous example, in which we replace the latency function of the second machine with the latency function $\ell_2(x) = 2x$. It is easily checked that the assignment at Nash equilibrium puts half of the jobs on each machine (for a cost of 1) while the optimal assignment is $(\frac{3}{4}, \frac{1}{4})$ (with a cost of $\frac{7}{8}$). On the other hand, if we again allow the system manager to assign half of the jobs, we see that for *any* strategy s, the assignment induced by s is $(\frac{1}{2}, \frac{1}{2})$ and hence is not optimal. In this example, there is no available strategy by which the system manager can improve system performance.

3 Three Stackelberg Strategies

3.1 Two Natural Strategies

We begin our investigation of Stackelberg strategies by considering two natural approaches that provide suboptimal performance guarantees. To motivate our results in the simplest possible way, throughout this subsection we will consider examples in which all latency functions are linear and half of the jobs are centrally controlled ($\alpha = \frac{1}{2}$). We are thus hoping for strategies that always induce an equilibrium of cost at most $\frac{8}{7}$ times that of the optimal assignment (this is best possible by the second example of subsection 2.3).

First consider the following strategy for an instance $(M, r, \frac{1}{2})$: if x^* is the optimal assignment for instance $(M, \frac{1}{2}r)$, put $s = x^*$. In words, we choose the strategy of minimum-cost (ignoring the existence of jobs that are not centrally controlled). We call this the *Aloof* strategy since it fails to acknowledge the rest of the traffic in the system. The example of subsection 2.2 shows that this strategy performs quite poorly: the strategy is $(0, \frac{1}{2})$ and the induced assignment is $(0, 1)$, an assignment that we have seen to be $\frac{4}{3}$ more costly than the optimal assignment.

A second attempt for a good strategy might be as follows: if x^* is the optimal assignment for (M, r), put $s = \frac{1}{2}x^*$. We call this the *Scale* strategy, since it is simply the optimal assignment of all the jobs, suitably scaled. Unfortunately, a simple example shows that the Scale strategy also fails to provide the performance guarantee of $\frac{8}{7}$ that we are looking for: in a two-machine example with latency functions 1 and $\frac{3}{2}x$ and rate 1, the optimal assignment is $(\frac{2}{3}, \frac{1}{3})$ (with total cost $\frac{5}{6}$) and thus the Scale strategy will be $(\frac{1}{3}, \frac{1}{6})$ which induces the equilibrium $(\frac{1}{3}, \frac{2}{3})$ having cost 1. Hence, the Scale strategy may result in an induced equilibrium $\frac{6}{5}$ as costly as the optimal assignment.

3.2 The Largest Latency First (LLF) Strategy

Intuitively, both the Aloof and Scale strategies suffer from a common flaw: both allocate jobs to machines that will be subsequently inundated in any induced equilibrium while assigning too little work to machines that selfish users are prone to ignore. This observation suggests that a good strategy should give priority to the machines that are least appealing to selfish users – machines with relatively high latency. With this intuition in mind, the following strategy for a Stackelberg instance (M, r, α) (which we call the *Largest Latency First* or *LLF* strategy) should seem natural:

(1) Compute the optimal assignment x^* for (M, r)

(2) Index the machines of M so that $\ell_1(x_1^*) \leq \cdots \leq \ell_m(x_m^*)$

(3) Let $k \leq m$ be minimal with $\sum_{i=k+1}^{m} x_i^* \leq \alpha r$

(4) Put $s_i = x_i^*$ for $i > k$, $s_k = \alpha r - \sum_{i=k+1}^{m} x_i^*$, and $s_i = 0$ for $i < k$

We will say that a machine i is *saturated* by a strategy s if $s_i = x_i^*$. Thus, the LLF strategy saturates machines one-by-one (in order from the largest latency w.r.t. x^* to the smallest) until there are no centrally controlled jobs remaining. Note that Lemma 2.4 implies that the LLF strategy can be computed in polynomial-time (the bottleneck is step (1)); in Section 5 we will see that it can computed in $O(m^2)$ when every latency function is linear.

The next two sections are devoted to proving that the LLF strategy always induces an assignment with near-optimal total latency.

4 A $\frac{1}{\alpha}$ Performance Guarantee for Arbitrary Latency Functions

In this section we prove performance guarantees for the LLF strategy for any set of latency functions and any number of machines. We note that *no* performance guarantee is possible in the absence of centrally controlled jobs – i.e., in this setting, the Nash assignment may incur arbitrarily more latency than the optimal assignment [26]. Thus, the benefit of a leader (and of a carefully chosen leader strategy) is particularly striking in this general setting.

A simple variation on previous examples demonstrates the limits of Stackelberg strategies. In a two-machine instance with $\alpha = \frac{1}{2}$ and latency functions $\ell_1(x) = 1$ and $\ell_2(x) = 2^k x^k$ for $k \in \mathcal{Z}^+$, any Stackelberg strategy induces the assignment $(\frac{1}{2}, \frac{1}{2})$ (having total latency 1) while the optimal assignment is $(\frac{1}{2} + \delta_k, \frac{1}{2} - \delta_k)$ having cost $\frac{1}{2} + \epsilon_k$, where $\delta_k, \epsilon_k \to 0$ as $k \to \infty$. Thus the best induced assignment may be (arbitrarily close to) twice as costly as the optimal assignment. Similar examples show that for any $\alpha \in (0, 1)$, the best induced assignment may be $\frac{1}{\alpha}$ times as costly as the optimal assignment.

The main result of this section is that the LLF strategy always induces an assignment of cost no more than $\frac{1}{\alpha}$ times that of the optimal assignment. A rough outline of the proof is as follows. Our goal is to proceed by induction on the number of machines. If the LLF strategy first saturates the mth machine, a natural idea is to apply the inductive hypothesis to the remainder of the LLF strategy on the first $m-1$ machines (note that the job rate and fraction of centrally controlled jobs in the inductive instance will be smaller than in the original instance) to derive a performance guarantee. This idea nearly succeeds, but there are two difficulties. First, it is possible that the LLF strategy fails to saturate any machines; we will see below that this case is easy to analyze and causes no trouble. Second, in order to obtain a clean application of the inductive hypothesis to the first $m-1$ machines, we require that the the optimal and LLF-induced assignments place the same amount of jobs on these machines – i.e., that the LLF-induced assignment eschews the mth machine. This difficulty is resolved by the following lemma (the proof may be found in the Appendix).

Lemma 4.1 *Let (M, r, α) denote a Stackelberg instance with optimal assignment x^* and index the machines of M so that $\ell_m(x_m^*) \geq \ell_i(x_i^*)$ for all i. If s is a strategy with $s_m = x_m^*$ and $s_i \leq x_i^*$ for all i, then there exists an induced equilibrium t with $t_m = 0$.*

We now make our previous arguments precise.

Theorem 4.2 *Let $\mathcal{I} = (M, r, \alpha)$ denote a Stackelberg instance. If s is an LLF strategy for $\mathcal{I}$ inducing equilibrium t and x^* is the optimal assignment for the instance (M, r), then $C(s + t) \leq \frac{1}{\alpha} C(x^*)$.*

Proof. We proceed by induction on the number of machines m (for each fixed m, we will prove the theorem for arbitrary ℓ, r, and α). The case of one machine is trivial.

Fix a Stackelberg instance $\mathcal{I} = (M, r, \alpha)$ with at least two machines and let x^* denote an optimal assignment to the instance (M, r). By scaling, we may assume that $r = 1$ (use latency functions $\tilde{\ell}$ with $\tilde{\ell}_i(x) = \ell_i(rx)$). Let L denote the common latency w.r.t. $s+t$ of every machine with $t_i > 0$ (see Lemma 2.9).

Case 1: Suppose that $\alpha > x_m^*$. Then $s_m = x_m^*$ (i.e., the LLF strategy saturates the last machine), so by Lemmas 2.8 and 4.1, we may assume that $t_m = 0$. Let M_1 denote the machines i for which $t_i = 0$ and M_2 the machines for which $t_i > 0$; both of these sets are non-empty. For $i = 1, 2$ let α_i denote the amount of centrally controlled jobs assigned to machines in M_i (i.e., $\alpha_i = s(M_i)$) and λ_i the cost incurred by $s + t$ on machines in M_i. By Lemma 2.9, $\lambda_2 = (1 - \alpha_1)L$ and $\lambda_1 \geq \alpha_1 L$. Now, x^* restricted to M_2 is an optimal assignment for $(M_2, 1 - \alpha_1)$ and hence s restricted to M_2 is an LLF strategy for the instance $\mathcal{I}_2 = (M_2, 1 - \alpha_1, \frac{\alpha_2}{1-\alpha_1})$. The inductive hypothesis and the fact that $x_i^* \geq s_i = s_i + t_i$ for all $i \in M_1$ implies that

$$C(x^*) \geq \lambda_1 + \alpha' \lambda_2$$

where $\alpha' = \frac{\alpha_2}{1-\alpha_1}$. Proving that $C(s + t) \leq \frac{1}{\alpha} C(x^*)$ thus reduces to showing

$$\alpha(\lambda_1 + \lambda_2) \leq \lambda_1 + \alpha' \lambda_2.$$

Since $\alpha \leq 1$ and $\lambda_1 \geq \alpha_1 L$, it suffices to prove this inequality with λ_1 replaced by $\alpha_1 L$. Substituting for λ_2 and α' and dividing through by L, we need only check that

$$\alpha(\alpha_1 + (1 - \alpha_1)) \leq \alpha_1 + \frac{\alpha_2}{1 - \alpha_1}(1 - \alpha_1)$$

which clearly holds (both sides are equal to α).

Case 2: Suppose $\alpha \leq x_m^*$. Then $s_m = \alpha$ and $s_i = 0$ for $i < m$. It follows that if $t_m > 0$ then $s + t$ is simply a Nash assignment for $(M, 1)$ with total cost L. By Lemma 2.5, we have $\ell_m(x_m^*) \geq L$ and thus $C(x^*) \geq x_m^* \ell_m(x_m^*) \geq \alpha L = \alpha C(s + t)$, implying the desired result.

On the other hand, if $t_m = 0$ then $s+t$ restricted to $M \setminus \{m\}$ is at Nash equilibrium for $(M \setminus \{m\}, 1-\alpha)$, so we may write $C(s + t) = \alpha \ell_m(\alpha) + (1 - \alpha)L$. Since $C(x^*) \geq \alpha \ell_m(\alpha)$ and $\ell_m(\alpha) \geq L$ (the latter since $t_m = 0$) we have $C(s + t) \leq \ell_m(\alpha) \leq \frac{1}{\alpha} C(x^*)$. ■

5 A $\frac{4}{3+\alpha}$ Performance Guarantee for Linear Latency Functions

5.1 Properties of the Nash and Optimal Assignments

In this subsection we undertake a deeper study of the Nash and optimal assignments for instances with linear latency functions. The results of this subsection will be instrumental in proving strengthened performance guarantees for the LLF strategy in these instances.

Fix a set of machines M with latency functions $\ell_i(x) = a_i x + b_i$ for each i ($a_i, b_i \geq 0$) and index them so that $b_1 \leq b_2 \cdots \leq b_m$. We may assume that at most one machine has a constant latency function ($a_i = 0$) since all but the fastest may be safely discarded; under this assumption, the Nash and optimal assignments are always unique. We may similarly assume that a machine with a constant latency function is the last machine.

Our first goal is to understand the structure of the Nash assignment $\bar{x}$ as a function of the rate r. It is useful to imagine r increasing from 0 to a large value, with the corresponding Nash assignment changing in a continuous fashion; an intuitive description of this process is as follows. Initially, when r is nearly zero, all jobs will be assigned to the machine having the smallest constant term. Once the first machine is sufficiently loaded, the second machine looks equally attractive (this occurs when $a_1\bar{x}_1 + b_1 = b_2$ – i.e., when the load on machine 1 is $\frac{b_2-b_1}{a_1}$). At this point, new jobs will be assigned to both of the first two machines, at rates proportional to $\frac{1}{a_1}$ and $\frac{1}{a_2}$ (jobs will be assigned so that these two machines will continue to have equal latency). Once $(b_3 - b_2)(\frac{1}{a_1} + \frac{1}{a_2})$ further units of work have arrived and been assigned to the first two machines, machine 3 will be equally attractive and new jobs will be spread out amount the first three machines, and so on. We may thus envision the Nash assignment as being constructed in phases: within phase i jobs are assigned to the first i machines according to fixed relative proportions and at the end of the phase (after enough new jobs have been assigned) an additional machine is put into use.

We now formalize this intuitive description of the Nash assignment $\bar{x}$. For $i = 1, \ldots, m$, let v_i denote the m-vector $(\frac{1}{a_1}, \frac{1}{a_2}, \ldots, \frac{1}{a_i}, 0, 0, \ldots, 0) \in \mathcal{R}^m_+$; if $a_m = 0$ put $v_m = (0, 0, \ldots, 1)$. Let $\|\cdot\|_1$ denote the L_1-norm of a vector. Define δ_i for $i = 0, 1, \ldots, m-1$ inductively by $\delta_0 = 0$ and $\delta_i = \min\{(b_{i+1} - b_i)\|v_i\|_1, r - \sum_{j=1}^{i-1} \delta_i\} \geq 0$. We also put $\delta_m = r - \sum_{j=1}^{m-1} \delta_i$. We can then describe $\bar{x}$ as follows.

Lemma 5.1 *Let $\mathcal{I}$ be an instance with linear latency functions, as above. Then the Nash assignment for $\mathcal{I}$ is given by*

$$\bar{x} = \sum_{i=1}^{m} \delta_i v_i.$$

Our characterization of optimal assignments (Lemma 2.4) yields an analogous result for computing them by an explicit formula. Note that when a latency function has the form $\ell_i(x) = a_i x + b_i$, the corresponding marginal cost function is $\ell^*_i(x) = 2a_i x + b_i$. Recalling that in an optimal assignment all gradients of used machines are equal (Lemma 2.4), we see that the optimal assignment is created by the same process as the Nash assignment, except that new machines are incorporated at a more rapid pace so as to spread jobs over a larger range of machines (and thus achieve a smaller overall total latency).

Formally, let v_i be as above and define δ^*_i inductively by $\delta^*_0 = 0$, $\delta^*_i = \min\{\frac{1}{2}(b_{i+1} - b_i)\|v_i\|_1, r - \sum_{j=1}^{i-1} \delta^*_i\}$, and $\delta^*_m = r - \sum_{j=1}^{m-1} \delta^*_i$. Letting x^* denote the optimal assignment to (M, r), the analog of Lemma 5.1 is as follows.

Lemma 5.2 *Let $\mathcal{I}$ be an instance with linear latency functions, as above. Then the optimal assignment for $\mathcal{I}$ is given by*

$$x^* = \sum_{i=1}^{m} \delta^*_i v_i.$$

Lemmas 5.1 and 5.2 have several useful corollaries. We summarize them below.

Corollary 5.3 *Let M be a set of machines with latency functions $\{\ell_i(x) = a_i x + b_i\}_{i \in M}$ and at most one machine with a constant latency function. Let b_i be nondecreasing in i. Then:*

(a) If x^* is the optimal assignment for (M, r_1) and y^* is the optimal assignment for (M, r_2) with $r_1 \geq r_2$, then $x^*_i \geq y^*_i$ for each i.

(b) If x^* and $\bar{x}$ denote the optimal and Nash assignments to (M, r) and $\bar{x}_i > 0$, then $x_i^* > 0$.

(c) If x^* and $\bar{x}$ denote the optimal and Nash assignments to (M, r), then $x_1^* \leq \bar{x}_1 \leq 2x_1^*$.

(d) If x^* and $\bar{x}$ denote the optimal and Nash assignments to (M, r), then $x_m^* \geq \bar{x}_m$.

(e) For any rate r, the optimal and Nash assignments of (M, r) can be computed in $O(m^2)$ time.

Note Corollary 5.3(e) implies that the LLF strategy can be computed in $O(m^2)$ time for instances with linear latency functions.

5.2 Analysis

In subsection 2.3 we saw an example with linear latency functions and $\alpha = \frac{1}{2}$ in which *no* strategy can induce an assignment with cost less than $\frac{8}{7}$ times that of the optimal assignment. This example is easily modified (by giving the second machine a latency function of $\frac{1}{1-\alpha}x$) to show that, for any $\alpha \in (0, 1)$, the minimum-cost induced assignment for a Stackelberg instance (M, r, α) may be $\frac{4}{3+\alpha}$ as costly as the optimal assignment for (M, r). The main result of this section is a matching upper bound for the LLF strategy.

Before proving this result, we give an alternative description of LLF that is more convenient for our analysis based on the following lemma (the proof may be found in the Appendix).

Lemma 5.4 *Let x^* be an optimal assignment for (M, r) where machine i has latency function $\ell_i(x) = a_i x + b_i$. Then $\ell_i(x_i^*) \geq \ell_j(x_j^*)$ if and only if $b_i \geq b_j$.*

Lemma 5.4 implies the following equivalent description of the LLF strategy: saturate machines one-by-one, in decreasing order of constant terms, until no centrally controlled jobs remain.

We are finally prepared to prove a $\frac{4}{3+\alpha}$ performance guarantee for the LLF strategy for instances with linear latency functions.

Theorem 5.5 *Let $\mathcal{I} = (M, r, \alpha)$ denote an instance with linear latency functions. If s is an LLF strategy for $\mathcal{I}$ inducing equilibrium t and x^* is the optimal assignment for (M, r), then $C(s + t) \leq \frac{4}{3+\alpha} C(x^*)$.*

Proof. We sketch only the main ideas and difficulties; the complete proof is given in the Appendix. The general approach is similar to that of Theorem 4.2 and is again by induction on the number of machines. However, new difficulties arise in proving a stronger performance guarantee. The case in which there is some machine i on which the induced equilibrium assigns no jobs ($t_i = 0$) is very similar to the first case of Theorem 4.2 (the desired performance guarantee can be easily extracted from the inductive guarantee for the smaller instance of machines on which $t_i > 0$), but the second case (in which the induced equilibrium assigns jobs to all machines) is substantially more complicated. In particular, the simple approach in the proof of Theorem 4.2 does *not* use any inductive guarantee in this case and is not strong enough to prove a guarantee better than $\frac{1}{\alpha}$. For this reason, much of the proof is devoted to defining an appropriate smaller instance that allows for clean application of the inductive hypothesis and extending the inductive guarantee into one for the original instance. ■

We remark that Theorem 5.5 was proved for $\alpha = 0$ (in a more general setting) in [26].

6 The Complexity of Computing Optimal Strategies

We have thus far measured the performance of a Stackelberg strategy by comparing the cost of the corresponding induced assignment to the cost of the optimal assignment of all of the jobs. Another natural approach for evaluating a strategy is to compare the cost of the induced assignment to that of the *least costly assignment induced by some Stackelberg strategy*, i.e., to the cost of the assignment induced by the *optimal* strategy. Motivated by the latter measure, in this section we study the optimization problem of computing the optimal Stackelberg strategy.

We have seen that the LLF strategy provides the best possible (worst-case) performance guarantee relative to the cost of the optimal assignment, and in particular that the algorithm of subsection 3.2 may be viewed as a $\frac{1}{\alpha}$-approximation algorithm for computing the optimal strategy (or a $\frac{4}{3+\alpha}$-approximation algorithm when every latency function is linear). However, simple examples (one is given in the Appendix) show that the LLF strategy is *not* always the optimal strategy, and thus our algorithm fails to solve this optimization problem exactly. Our main result of this section is evidence that no such polynomial-time algorithm exists.

Theorem 6.1 *The problem of computing the optimal Stackelberg strategy is NP-hard, even for instances with linear latency functions.*

Proof. We provide only a sketch of the main ideas and defer the details to the full version. We will reduce the NP-complete problem of partitioning a set of positive integers into two sets with equal sums (PARTITION, [SP12] in [14]) to that of computing the optimal strategy for a Stackelberg instance with linear latency functions.

The key observation about the relative performance of different Stackelberg strategies is the following. Suppose an instance admits a Nash assignment $\bar{x}$ and an optimal assignment x^*. Intuitively, the role of a Stackelberg strategy is to induce an assignment "close to" x^* in spite of the fact that, when there is no centralized intervention, jobs are assigned according to $\bar{x}$. Roughly, a machine i with $\bar{x}_i \geq x_i^*$ is of no use since it is overloaded by selfish users even when no centrally controlled jobs are assigned to i; on the other hand, assigning s_i units of centrally controlled jobs to a machine i with $x_i^* > \bar{x}_i$ should improve the cost of the induced assignment, provided $s_i > \bar{x}_i$ (as we have then succeed in producing an induced equilibrium that is "closer to x^*" and "further from $\bar{x}$"). Moreover, we expect the strategy to improve as we increase s_i up to x_i^*. With this intuition in mind, it is tempting to conjecture the following characterization of optimal strategies (for a fixed value of α):

(*) a strategy s is optimal if and only if it maximizes $\sum_i \max\{0, s_i - \bar{x}_i\}$ subject to the conditions that $\sum_i s_i = \alpha$ and $s_i \leq x_i^*$ for each i.

Suppose for a moment that this condition characterizes the optimal strategies of any Stackelberg instance, and consider an instance of PARTITION specified by positive integers $c_1, \ldots, c_n$. Define an instance $\mathcal{I} = (M, 2C, \frac{1}{4})$ (where C denotes $\sum_{i=1}^n c_i$) with $n+1$ machines having linear latency functions such that $x_i^* = 2\bar{x}_i = c_i$ for $i = 1, \ldots, n$ (and thus $x_{n+1}^* = C, \bar{x}_{n+1} = \frac{3}{2}C$); for example we could put $\ell_i(x) = \frac{x}{c_i} + 4$ for $i = 1, \ldots, n$ and $\ell_{n+1}(x) = \frac{3x}{C}$. If $\{c_1, \ldots, c_n\}$ can be partitioned into two sets of equal sum (say $S \subseteq \{1, \ldots, n\}$ satisfies $\sum_{i \in S} c_i = \frac{1}{2}C$) then putting $s_i = x_i^* = c_i$ for $i \in S$ and $s_i = 0$ yields a strategy with $\sum_i \max\{0, s_i - \bar{x}_i\} = \sum_{i \in S}(c_i - \frac{1}{2}c_i) = \frac{1}{4}C$. On the other hand, if $\{c_1, \ldots, c_n\}$ cannot be partitioned, for any strategy s with $s_i \leq x_i^*$ there will be a machine i with $0 < s_i < 2\bar{x}_i$ (also, $s_i \leq 2\bar{x}_i$ for every i), implying that $\sum_i \max\{0, s_i - \bar{x}_i\} < \frac{1}{4}C$. Thus, if (*) did characterize optimal strategies, we could solve the original PARTITION instance by computing the optimal Stackelberg strategy and checking if it fails to saturate any of the machines to which it assigns jobs.

Unfortunately, (*) fails to characterize optimal strategies in all instances (even restricting to instances with linear latency functions). However, it can be shown that instance $\mathcal{I}$ *does* have this property. Thus, we have constructed a class of instances simultaneously rich enough to encode all PARTITION instances and simple enough to possess the structural properties needed in our reduction; it follows that PARTITION reduces to the problem of computing the optimal Stackelberg strategy for an instance with linear latency functions, proving the latter NP-hard. ∎

Acknowledgements

We thank Éva Tardos for several helpful discussions and for comments on an earlier draft of this paper. We also thank Anupam Gupta for useful discussions and Ben Atkin and Jon Kleinberg for pointing out relevant references.

References

[1] A. Bagchi. *Stackelberg Differential Games in Economic Models*. Springer-Verlag, 1984.

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

[3] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economices of Transportation*. Yale University Press, 1956.

[4] K. P. Birman. *Building Secure and Reliable Network Applications*. Manning, 1996.

[5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobsob, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. Network Working Group Request for Comments 2309, April 1998.

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol. Network Working Group Request for Comments 1067, August 1988.

[7] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transactions on Networking*, 1(6):614–627, 1993.

[8] S. C. Dafermos and F. T. Sparrow. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards, Series B*, 73B(2):91–118, 1969.

[9] C. Douligeris and R. Mazumdar. Multilevel flow control of queues. In *Proceedings of the Johns Hopkins Conference on Information Sciences and Systems*, page 21, 1989.

[10] C. Douligeris and R. Mazumdar. A game theoretic perspective to flow control in telecommunication networks. *Journal of the Franklin Institute*, 329:383–402, 1992.

[11] P. Dubey. Inefficiency of Nash equilibria. *Mathematics of Operations Research*, 11(1):1–8, 1986.

[12] A. A. Economides and J. A. Silvester. Priority load sharing: An approach using Stackelberg games. In *Proceedings of the 28th Annual Allerton Conference on Communications, Control, and Computing*, pages 674–683, 1990.

[13] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 218–227, 2000.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Thoery of NP-Completeness*. Freeman, 1979.

[15] S. Keshav. *An Engineering Approach to Computer Networking*. Addison-Wesley, 1997.

[16] Y. A. Korlis, A. A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161–173, 1997.

[17] Y. A. Korlis, A. A. Lazar, and A. Orda. Capacity allocation under noncooperative routing. *IEEE Transactions on Automatic Control*, 42(3):309–325, 1997.

[18] Y. A. Korlis, A. A. Lazar, and A. Orda. Avoiding the Braess paradox in noncooperative networks. *Journal of Applied Probability*, 42(3):309–325, 1999.

[19] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 404–413, 1999.

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

[21] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 129–140, 1999.

[22] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multi-user communication networks. *IEEE/ACM Transactions on Networking*, 1:510–521, 1993.

[23] G. Owen. *Game Theory*. Academic Press, 1995. Third Edition.

[24] C. ReVelle and D. Serra. The maximum capture problem including relocation. *INFOR*, 29:130–138, 1991.

[25] A. Ronen. *Solving Optimization Problems among Selfish Agents*. PhD thesis, Hebrew University of Jerusalem, 2000.

[26] T. Roughgarden and É. Tardos. How bad is selfish routing? In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000. To appear.

[27] Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice-Hall, 1985.

[28] S. J. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3(6):819–831, 1995.

A Missing Proofs

A.1 Proofs from Section 4

Proof of Lemma 4.1. Consider an arbitrary induced equilibrium t and suppose $t_m > 0$. Let $L = \ell_m(x^*_m + t_m)$ denote the common latency w.r.t. $s + t$ of every machine with $t_i > 0$ (see Lemma 2.9). An argument similar to that for Lemma 2.5 shows that $\ell_m(x^*_m) \geq L$, which in turn implies $\ell_m(x^*_m) = \ell_m(x^*_m + t_m) = L$ (as ℓ_m is nondecreasing). Therefore, ℓ_m is constant on $[x^*_m, x^*_m + t_m]$ and in particular the gradient of machine m at x^*_m is given by $\ell^*_m(x^*_m) = \ell_m(x^*_m) = L$. Applying Lemma 2.4, we find that $\ell^*_i(x^*_i) = L$ whenever $x^*_i > 0$ and hence $\ell_i(x^*_i) \leq L$ whenever $x^*_i > 0$.

Let M' denote the machines on which $s_i + t_i < x_i^*$. Note that $x_i^* > 0$ for each $i \in M'$ and hence for each such i, ℓ_i is equal to L on $[s_i + t_i, x_i^*]$ (since we know that $\ell_i(s_i + t_i) \geq L$, $\ell_i(x_i^*) \leq L$, and ℓ_i is nondecreasing). Since x^* and $s+t$ are assignments for the same rate, we must have $\sum_{i \in M'} x_i^* - (s_i + t_i) \geq t_m$ (since $t_m > 0$, M' is non-empty). Finally, consider modifying t as follows: move all jobs previously assigned to machine m to machines in M', subject to the constraint $s_i + t_i \leq x_i^*$. We have already observed that there is sufficient "room" on machines in M' for this modification, and that all latency functions are constant in the domain of our modifications. We have thus exhibited a new induced equilibrium with no jobs assigned to machine m, completing the proof. ■

A.2 Proofs from Section 5

Proof of Lemma 5.4. The lemma is clear when $x_i^* = x_j^* = 0$. If one of x_i^*, x_j^* is 0 (say x_i^*) and the other is positive, then by Lemma 2.4 we know that $\ell_i^*(x_i^*) = \ell(0) = b_i$ is at least $\ell_j^*(x_j^*) = 2a_j x_j^* + b_j$. Thus we necessarily have both $b_i \geq b_j$ and $\ell_i(x_i^*) \geq \ell_j(x_j^*)$. Finally, if $x_i^*, x_j^* > 0$ then by Lemma 2.4 we have $2a_i x_i^* + b_i = 2a_j x_j^* + b_j = L^*$ for some L^*; thus $b_i \geq b_j$ if and only if $a_i x_i^* \leq a_j x_j^*$. The lemma follows by writing $\ell(x_i^*) = L^* - a_i x_i^*$ and $\ell(x_j^*) = L^* - a_j x_j^*$. ■

Proof of Theorem 5.5. We again proceed by induction on the number of machines m (for each fixed m, we will prove the theorem for arbitrary (linear) ℓ, r, and α). The case of one machine is trivial.

Fix a Stackelberg instance $\mathcal{I} = (M, r, \alpha)$ with at least two machines and let $\ell_i(x) = a_i x + b_i$ (with $a_i, b_i \geq 0$). Let x^* denote an optimal assignment to (M, r). We begin with several simplifying assumptions, each made with no loss of generality. We assume (as usual) that there is at most one machine with a constant latency function. We also assume that no machine has latency function $\ell(x) = 0$ (otherwise the instance is trivial) and that every machine is used by x^* (using Corollary 5.3(b), other machines may be discarded without affecting our argument). As in Theorem 4.2, we may assume that $r = 1$. It will also be convenient to assume that some machine i has constant term 0 (i.e., $b_i = 0$). To enforce this assumption we may subtract $\min_i b_i$ from every latency function before applying our argument: assuming $r = 1$, this modification decreases the cost of every assignment by precisely $\min_i b_i$ and will only increase the ratio in costs between any two strategies.

Let s denote the LLF strategy for $\mathcal{I}$ and t the induced equilibrium. Let L denote the common latency of every machine used by t. Index the machines of M as in the second description of the LLF strategy, so that $0 = b_1 \leq b_2 \leq \cdots \leq b_m$ and $a_1 > 0$. We will need to apply the inductive hypothesis in two different ways, and our analysis breaks into two cases.

Case 1: Suppose $t_i = 0$ for some i. As in the proof of Theorem 4.2, let M_1 denote the machines on which $t_i = 0$ and M_2 the machines on which $t_i > 0$. For $i = 1, 2$, let α_i denote the amount of centrally controlled traffic on machines in M_i, so $\alpha_i = s(M_i)$. For $i = 1, 2$ let λ_i denote the cost incurred by $s + t$ on machines in M_i. Observe that $\lambda_1 \geq \alpha_1 L$ and $\lambda_2 = (1 - \alpha_1)L$ (see Lemma 2.9). Since x^* restricted to M_2 is an optimal assignment for $(M_2, 1 - \alpha_1)$, s restricted to M_2 is an LLF strategy for $\mathcal{I}_2 = (M_2, 1 - \alpha_1, \frac{\alpha_2}{1-\alpha_1})$. The inductive hypothesis (applied to $\mathcal{I}_2$) and the fact that $x_i^* \geq s_i = s_i + t_i$ for all $i \in M_1$ implies that

$$C(x^*) \geq \lambda_1 + \frac{3 + \alpha'}{4} \lambda_2$$

where $\alpha' = \frac{\alpha_2}{1-\alpha_1}$. Proving that $C(s + t) \leq \frac{4}{3+\alpha} C(x^*)$ thus reduces to showing

$$(3 + \alpha)(\lambda_1 + \lambda_2) \leq 4\lambda_1 + (3 + \alpha')\lambda_2.$$

Since $\alpha \leq 1$ and $\lambda_1 \geq \alpha_1 L$, it suffices to prove this inequality with λ_1 replaced by $\alpha_1 L$. Substituting for λ_2, α' and dividing through by L verifies the result.

Case 2: Suppose $t_i > 0$ for all machines $i \in M$. This implies that $s+t$ is a Nash assignment for $(M, 1)$; by Corollary 5.3(d) we have $s_m < s_m + t_m \leq x^*_m$ and hence (by definition of LLF) $s_i = 0$ for $i < m$. Also, $\alpha = s_m$.

Our first goal is to show that s is an LLF strategy not only for $\mathcal{I}$ but also for $\mathcal{I}' = (M', 1 - t_1, \frac{\alpha}{1-t_1})$, where $M' = M \setminus \{1\}$ (we may then apply the inductive hypothesis to s in a smaller instance). Toward this end, let y^* denote the optimal assignment to the instance $(M', 1 - t_1)$. Since $s + t$ restricted to M' is a Nash assignment for $(M', 1 - t_1)$, we must have $y^*_m \geq s_m + t_m$ (see Corollary 5.3(d)); since $\alpha = s_m < s_m + t_m \leq y^*_m$, the LLF strategy for $\mathcal{I}'$ is precisely s.

It will be convenient to split the cost of x^* into two terms; let μ_1, μ_2 denote the total latency incurred by x^* on machine 1 and in M' respectively. Recalling that L denotes the common latency of every machine (with $t_i > 0$) w.r.t. $s + t$, we have $C(s + t) = L$, with total latency $t_1 L$ on machine 1 and $(1 - t_1)L$ in M'. The following claim gives us a lower bound on μ_2.

Claim: *If $r \geq 1 - t_1$, then the cost of the optimal assignment for (M', r) is at least*

$$\frac{3+\alpha'}{4}(1-t_1)L + (r - 1 + t_1)L$$

where $\alpha' = \frac{\alpha}{1-t_1}$.

Proof. The claim is proved for $r = 1 - t_1$ by applying the inductive hypothesis to the instance $\mathcal{I}'$ and using the fact that the LLF strategy s induces an equilibrium in M' of cost $(1 - t_1)L$. Suppose now that $r > 1 - t_1$. We again denote the optimal assignment for $(M', 1 - t_1)$ by y^*. An argument similar to that for Lemma 2.5 shows that the common gradient of every machine in M' is at least L w.r.t. y^*. By Corollary 5.3(a) and the observation that gradients are nondecreasing in congestion, all future jobs will be assigned at a marginal cost of at least L. Since there is $r - (1 - t_1)$ further work to schedule (beyond the work already assigned according to y^*), the overall cost of an optimal assignment to (M', r) must be at least $C(y^*) + (r - 1 + t_1)L \geq \frac{3+\alpha'}{4}(1 - t_1)L + (r - 1 + t_1)L$. ■

We have thus reduced the proof of the theorem to proving the inequality

$$(3 + \alpha)L \leq (3 + \alpha')(1 - t_1)L + 4(t_1 - x^*_1)L + 4a_1(x^*_1)^2$$

where $\alpha' = \frac{\alpha}{1-t_1}$ (since by the Claim, the RHS is at most $4(\mu_1+\mu_2)$). For any fixed value of t_1, $x^*_1 \in [\frac{1}{2}t_1, t_1]$ (see Corollary 5.3(c)); since the LHS is independent of x^*_1 and the RHS is minimized by $x^*_1 = \frac{1}{2}t_1$ it suffices to prove that

$$(3 + \alpha)L \leq (3 + \alpha')(1 - t_1)L + 2t_1 L + a_1 t_1^2.$$

Substituting for α', using the identity $a_1 t_1 = L$, and dividing by L gives

$$3 + \alpha \leq (3 + \frac{\alpha}{1 - t_1})(1 - t_1) + 3t_1$$

which clearly holds, proving the theorem. ■

A.3 Proofs from Section 6

Claim: *The LLF strategy need not be optimal.*

Proof. Consider a three machine example with latency functions $\ell_1(x) = x, \ell_2(x) = \ell_3(x) = 1 + x$, with $r = 1$ and $\alpha = \frac{1}{6}$. The optimal assignment is $(\frac{2}{3}, \frac{1}{6}, \frac{1}{6})$ and thus the LLF strategy is $(0, 0, \frac{1}{6})$ inducing the assignment $(\frac{5}{6}, 0, \frac{1}{6})$ with cost $\frac{8}{9}$. On the other hand, the strategy $(0, \frac{1}{12}, \frac{1}{12})$ induces the assignment $(\frac{5}{6}, \frac{1}{12}, \frac{1}{12})$ having cost $\frac{63}{72}$. ■