

# Online Primal-Dual For Non-linear Optimization with Applications to Speed Scaling

Anupam Gupta\*

Ravishankar Krishnaswamy\*

Kirk Pruhs†

## Abstract

We reinterpret some online greedy algorithms for a class of nonlinear “load-balancing” problems as solving a mathematical program online. For example, we consider the problem of assigning jobs to (unrelated) machines to minimize the sum of the  $\alpha^{th}$ -powers of the loads plus assignment costs (the *online Generalized Assignment Problem*); or choosing paths to connect terminal pairs to minimize the  $\alpha^{th}$ -powers of the edge loads (i.e., *online routing with speed-scalable routers*). We give analyses of these online algorithms using the dual of the primal program as a lower bound for the optimal algorithm, much in the spirit of online primal-dual results for linear problems.

We then observe that a wide class of uni-processor speed scaling problems (with essentially arbitrary scheduling objectives) can be viewed as such load balancing problems with linear assignment costs. This connection gives new algorithms for problems that had resisted solutions using the dominant potential function approaches used in the speed scaling literature, as well as alternate, cleaner proofs for other known results.

---

\*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Supported in part by NSF awards CCF-0964474 and CCF-1016799. R.K. supported in part by an IBM Graduate Fellowship.

†Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA. [kirk@cs.pitt.edu](mailto:kirk@cs.pitt.edu). Supported in part by NSF grant CCF-0830558 and an IBM Faculty Award.

# 1 Introduction

In this paper, we consider two online problems related to load balancing. We call the first problem *Online Generalized Assignment Problem (OnGAP)*:

**Definition of OnGAP:** Jobs arrive one by one in an online manner, and the algorithm must fractionally assign these jobs to one of  $m$  machines. When a job  $j$  arrives, the online algorithm learns  $\ell_{je}$ , the amount by which the load of machine  $e$  would increase for *each unit* of work of job  $j$  that is assigned to machine  $e$ , and  $c_{je}$ , the assignment cost incurred for each unit of work of job  $j$  that is assigned to machine  $e$ . The goal is to minimize the sum of the  $\alpha^{th}$  powers of the machine loads, plus the total assignment cost.

The version of OnGAP without assignment costs was studied by [AAF<sup>+</sup>97, AAG<sup>+</sup>95]. Our original motivation for studying OnGAP is that it models a well-studied class of *speed scaling problems with sum cost scheduling objectives*. In these problems, jobs arrive over time and must be scheduled on a speed scalable processor—i.e., a processor that can run at any non-negative speed, and uses power  $s^\alpha$  when run at speed  $s$ . The objective is the sum of the energy used by the processor plus a fractional scheduling objective that is the sum over jobs of the “scheduling cost” of the individual jobs. These speed scaling problems are a special case of OnGAP where the machines model the times that jobs can be scheduled, the assignment cost  $c_{je}$  models the scheduling cost for scheduling a unit of job  $j$  at time  $e$ . For example, one such scheduling objective is the sum of the fractional flow/response times squared. For this objective,  $c_{je}$  is  $(e - r_j)^2$  for all times  $e$  that are at least the release time  $r_j$  of job  $j$ , and infinite otherwise. Another example is the problem of minimizing energy usage subject to deadline constraints, introduced by [YDS95] and considered in followup papers [BKP07, BCP11, BCPK09]. This problem can be viewed as a special case of OnGAP, where each job  $j$  has an associated deadline  $d_j$ , and  $c_{je}$  is zero if  $e \in [r_j, d_j]$  and infinite otherwise.

The second problem that we consider is a variation/generalization of OnGAP involving *online routing with speed scalable routers* to minimize energy, which was previously considered in [AAF<sup>+</sup>97, AAZ11].

**Definition of Online Routing with Speed Scalable Routers Problem:** A sequence of requests arrive one by one over time. Each request  $j$  has an associated source-sink pair  $(s_j, t_j)$  in a network of speed scalable routers, and the online algorithm must route flow between the source-sink pair, with an objective of minimizing the total energy used by the network, where the energy incurred by an edge  $e$  is the  $\alpha^{th}$  power of the load flowing through it.

For load balancing and online routing, it was known that natural online greedy algorithms, which assign jobs to the machine(s) that minimize the increase in cost, can be shown to be  $O_\alpha(1)$ -competitive via an exchange argument, and directly bounding the cost compared to the optimal cost [AAF<sup>+</sup>97, AAG<sup>+</sup>95]. (In fact, basically the same argument shows that the online greedy algorithm is  $O_\alpha(1)$ -competitive for *integer* assignments.) Once we observe that speed scaling problems with sum scheduling objectives can be reduced to OnGAP, it is not too difficult to see that the analysis technique in [AAG<sup>+</sup>95] can be used to show that natural greedy speed scaling algorithms are  $O_p(1)$ -competitive.

**Our Contribution.** In this paper, we first interpret these online problems as solving a mathematical program online, where the constraints arrive one-by-one, and in response to the arrival of a new constraint, the online algorithm has to raise some of the primal variables so that the new constraint will be satisfied. The online algorithms that we consider raise the primal variables greedily. Our competitive analysis will use the dual function of the primal program as a lower bound for optimal. For analysis purposes, we assign a value to the dual variable corresponding to a constraint after the online algorithm has satisfied that constraint. Our goal is to set the dual variables so that the resulting dual solution is closely related to the online solution. (In our analyses, the settings of the dual variables naturally correspond to (some approximation for) the increase in the objective function.) We first show how to obtain fractional solutions to these problems, and subsequently show how similar ideas can be used for integer assignments.

Our analyses are very much in the spirit of the online primal dual technique for linear programs [BN07]. The main difference is that in the nonlinear setting, the dual is more complicated than in the linear setting (where the dual is just another linear program). Indeed, in the nonlinear setting, one can not disentangle the objective and the constraints, since the dual itself contains a version of the primal objective, and hence copies of the primal variables, within it. Consequently, the arguments for the dual function in the nonlinear setting have a somewhat different feel to them than in the linear setting. In particular, we need to set the dual variables  $\lambda$ , and then find minimizing settings for the copies of the primal variables to get a good lower bound. For the load-balancing and speed scaling problems, this proceeds relatively naturally. But for the routing problem the dual minimization problem is itself non-trivial: in this case we first show how to write a “relaxed/decoupled” dual, which is potentially weaker than the original dual, but easier to argue about, and then set the variables of this relaxed dual to achieve a good lower bound. We hope this analytical technique will be useful for other problems.

We then show how a wide class of uni-processor speed-scaling problems (with essentially arbitrary scheduling objectives) can be viewed as load balancing problems with linear assignment costs. This connection gives new algorithms for speed-scaling problems that had resisted solutions using the dominant potential function approaches used in the speed scaling literature, as well as alternate, cleaner analyses for some known results. For speed scaling problems, our analysis using duality is often cleaner (compare for example, the analysis of OA in [BKP07] to the analysis given here) and more widely applicable (for example, to nonlinear scheduling objectives) than the potential function-based analyses. Furthermore, we believe that much like the online primal-dual approach for linear problems, the techniques presented here have potential for wide applicability in the design and analysis of online algorithms for other non-linear optimization problems.

**Roadmap:** In Section 1.1 we discuss related work. In Section 2 we consider OnGAP. In Section 3, we make some comments about the application of these results to speed scaling problems. In Section 4 we consider the online routing problem. In Section 5 we show how to alter the water-filling algorithm to obtain *integer assignments* with a similar competitive ratio, as well a simple randomized rounding with a slightly worse performance.

## 1.1 Related Work

An  $O(\alpha)$ -competitive online greedy algorithm for the unrelated machines load-balancing problem in the  $L_\alpha$ -norm was given by [AAF<sup>+</sup>97, AAG<sup>+</sup>95]; Caragiannis [Car08] gave better analyses and improvements using randomization. An offline  $O(1)$ -approximation for this problem was given by [AE05] and [AKMPS09], via solving the convex program and then rounding the solution in a correlated fashion. For the routing problem, the  $O(\alpha)^\alpha$ -algorithm can be inferred from the ideas of [AAF<sup>+</sup>97, AAG<sup>+</sup>95]. Followup work in a setting of a network consisting of routers with static and dynamic power components can be found in [AAZ10, AAZ11].

There are two main lines of speed scaling research that fit within the framework that we consider here. This first is the problem of minimizing energy used with deadline feasibility constraints. [YDS95] proposed two online algorithms *OA* and *AVR*, and showed that *AVR* is  $O_\alpha(1)$ -competitive by reasoning directly about the optimal schedule. [BKP07] introduced the use of potential functions for analyzing online scheduling problems, and showed that *OA* and another algorithm *BKP* are  $O_\alpha(1)$ -competitive. [BBCP11] gave a potential function analysis to show that *AVR* is  $O_\alpha(1)$ -competitive. [BCPK09] introduced the algorithm *qOA*, and gave a potential function analysis to show that it has a better competitive ratio than *OA* or *AVR* for smallish  $\alpha$ .

The second main line of speed scaling research is when the scheduling objective is total flow, or more generally total weighted flow. [PUW08, AF07] gave offline algorithms for unit-weight unit-work jobs. All of the work on online algorithms consider some variation of the “natural” algorithm, which uses the “right” job selection algorithm from the literature on scheduling fixed speed processors, and sets

the power of the processor equal to the weight of the outstanding jobs. This speed scaling policy is “natural” in that it balances the energy and scheduling costs. By reasoning directly about the energy optimal schedule, [AF07] showed that a batched version of the natural algorithm is  $O_\alpha(1)$ -competitive for unit-work unit-weight jobs. Using a potential function analysis, [BPS09] showed that a variation of the natural algorithm is  $O_\alpha(1)$ -competitive for arbitrary-weight arbitrary-work jobs. For the objective of total flow plus energy, the bound on the competitive ratio was improved in [LLTW08] by use of potential function tailored to integer flow instead of fractional flow. Using a potential function analysis, [BCP09] showed a variation on the natural algorithm is  $O(1)$ -competitive for total flow plus energy for an arbitrary power function, and a variation on the natural algorithm is scalable, for fractional weighted flow plus energy for an arbitrary power function. [ALW10] improved the bound on the competitive ratio for total flow plus energy. Nonclairvoyant algorithms are analyzed in [CEL<sup>+</sup>09, CLL10]. A relatively recent survey of the algorithmic power management literature in general, and the speed scaling literature in particular, can be found in [Alb10].

An extensive survey/tutorial on the online primal dual technique for linear problems, as well the history of the development of this technique, can be found in [BN07].

## 2 The Online Generalized Assignment Problem

In this section we consider the problem of Online Generalized Assignment Problem (OnGAP). If  $x_{je}$  denotes the extent to which job  $j$  is assigned on machine  $e$ , then this problem can be expressed by the following mathematical program:

$$\begin{aligned} \min \quad & \sum_e \left( \sum_j \ell_{je} x_{je} \right)^\alpha + \sum_e \sum_j c_{je} x_{je} \\ \text{subject to} \quad & \sum_e x_{je} \geq 1 \quad j = 1, \dots, n \end{aligned}$$

The dual function of the primal relaxation is then

$$g(\lambda) = \min_{x \geq 0} \left( \sum_j \lambda_j + \sum_e \left( \sum_j \ell_{je} x_{je} \right)^\alpha + \sum_{j,e} c_{je} x_{je} - \sum_{j,e} \lambda_j x_{je} \right) \quad (2.1)$$

One can think of the dual problem as having the same instance as the primal, but where jobs are allowed to be assigned to extent zero. In the objective, in addition to the same load cost  $\sum_e \left( \sum_j \ell_{je} x_{je} \right)^\alpha$  as in the primal, a fixed cost of  $\lambda_j$  is paid for each job  $j$ , and a payment of  $\lambda_j - c_{je}$  is obtained for each unit of job  $j$  assigned. It is well known that each feasible value of the dual function is a lower bound to the optimal primal solution; this is *weak duality* [BV04].

**Online Greedy Algorithm Description:** Let  $\delta$  be a constant that we will later set to  $\frac{1}{\alpha^{\alpha-1}}$ . To schedule job  $j$ , the load is increased on the machines for which the increase the cost will be the least, assuming that energy costs are discounted by a factor of  $\delta$ , until a unit of job  $j$  is scheduled. More formally, the value of all the primal variables  $x_{je}$  for all the machines  $e$  that minimize

$$\delta \cdot \alpha \cdot \ell_{je} \left( \sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{je} \quad (2.2)$$

are increased until all the work from job  $j$  is scheduled, i.e.,  $\sum_e x_{je} = 1$ . Notice that  $\alpha \cdot \ell_{je} \left( \sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1}$  is the rate at which the load cost is increasing for machine  $e$ , and  $c_{je}$  is the rate that assignment costs are increasing for machine  $e$ . In other words, our algorithm fractionally assigns the job to the machines on which the overall objective function increases at the least rate. Furthermore, observe that if the

algorithm begins assigning the job to some machine  $e$ , it does not stop raising the primal variable  $x_{je}$  until the job is fully assigned<sup>1</sup>. By this monotonicity property, it is clear that all machines  $e$  for which  $x_{je} > 0$  have the same value of the above derivative when  $j$  is fully assigned. Now, for the purpose of analysis, we set the value  $\hat{\lambda}_j$  to be the rate of increase of the objective value when we assigned the last infinitesimal portion of job  $j$ . More formally, if  $e$  is any machine on which job  $j$  is run, i.e., if  $x_{je} > 0$ , then

$$\hat{\lambda}_j := \delta \cdot \alpha \cdot \ell_{je} \left( \sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{je} \quad (2.3)$$

Intuitively,  $\hat{\lambda}_j$  is a surrogate for the total increase in objective function value due to our fractional assignment of job  $j$  (we assign a total of 1 unit of job  $j$ , and  $\lambda_j$  is set to be the rate at which objective value increases).

We now move on to the analysis of our algorithm. To this end, let  $\tilde{x}$  denote the final value of the  $x_{je}$  variables for the online algorithm.

**Algorithm Analysis.** To establish that the online algorithm is  $\alpha^\alpha$ -competitive, note that it is sufficient (by weak duality) to show that  $g(\hat{\lambda})$  is at least  $\frac{1}{\alpha^\alpha}$  times the cost of the online solution. Towards this end, let  $\hat{x}$  be the value of the minimizing  $x$  variables in  $g(\hat{\lambda})$ , namely

$$\hat{x} = \arg \min_{x \geq 0} \left( \sum_j \hat{\lambda}_j + \sum_e \left( \sum_j \ell_{je} x_{je} \right)^\alpha - \sum_{j,e} \left( \hat{\lambda}_j - c_{je} \right) x_{je} \right)$$

Observe that the values  $\hat{x}$  could be very different from the values  $\tilde{x}$ , and indeed the next few Lemmas try to characterize these values. Lemma 2.1 notes that  $\hat{x}$  only has one job  $\varphi(e)$  on each machine  $e$ , and Lemma 2.2 shows how to determine  $\varphi(e)$  and  $\hat{x}_{\varphi(e)e}$ . Then, in Lemma 2.3, we show that a feasible choice for the job  $\varphi(e)$  is the latest arriving job for which the online algorithm scheduled some bit of work on machine  $e$ ; Let us denote this latest job by  $\psi(e)$ .

**Lemma 2.1** *There is a minimizing solution  $\hat{x}$  such that if  $\hat{x}_{je} > 0$ , then  $\hat{x}_{ie} = 0$  for all  $i \neq j$ .*

**Proof.** Suppose for some machine  $e$ , there exist distinct jobs  $i$  and  $k$  such that  $\hat{x}_{ie} > 0$  and  $\hat{x}_{ke} > 0$ . Then by the usual argument of either increasing or decreasing these variables along the line that keeps their sum constant, we can keep the convex term  $(\sum_j \ell_{je} \hat{x}_{je})^\alpha$  term fixed and not increase the linear term  $\sum_j (\hat{\lambda}_j - c_{je}) \hat{x}_{je}$ . This allows us to either set  $\hat{x}_{ie}$  or  $\hat{x}_{ke}$  to zero without increasing the objective. ■

**Lemma 2.2** *Define  $\varphi(e) = \arg \max_j \frac{(\hat{\lambda}_j - c_{je})}{\ell_{je}}$ . Then  $\hat{x}_{\varphi(e)e} = \frac{1}{\ell_{\varphi(e)e}} \left( \frac{\hat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha \ell_{\varphi(e)e}} \right)^{1/(\alpha-1)}$  and  $\hat{x}_{je} = 0$  for  $j \neq \varphi(e)$ . Moreover, the contribution of machine  $e$  towards  $g(\hat{\lambda})$  is exactly  $(1-\alpha) \left( \frac{\hat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha \ell_{\varphi(e)e}} \right)^{\alpha/(\alpha-1)}$ .*

**Proof.** By Lemma 2.1 we know that in  $\hat{x}$  there is at most one job (say  $j$ , if any) run on machine  $e$ . Then the contribution of this machine to the value of  $g(\hat{\lambda})$  is

$$(\ell_{je} \hat{x}_{je})^\alpha - (\hat{\lambda}_j - c_{je}) \hat{x}_{je} \quad (2.4)$$

Since  $\hat{x}$  is a minimizer for  $g(\hat{\lambda})$ , we know that the partial derivative of the above term evaluates to zero. This gives  $\alpha \ell_{je} \cdot (\ell_{je} \hat{x}_{je})^{\alpha-1} - (\hat{\lambda}_j - c_{je}) = 0$ , or equivalently,  $\hat{x}_{je} = \frac{1}{\ell_{je}} \left( \frac{\hat{\lambda}_j - c_{je}}{\alpha \ell_{je}} \right)^{1/(\alpha-1)}$ . Substituting

---

<sup>1</sup>It may however increase  $x_{je}$  and  $x_{je'}$  at different rates so as to balance the derivatives where  $e$  and  $e'$  are both machines which minimize equation 2.2

into this value of  $\hat{x}_{je}$  into equation (2.4), the contribution of machine  $e$  towards the dual  $g(\hat{\lambda})$  is

$$\left(\frac{\hat{\lambda}_j - c_{je}}{\alpha \ell_{je}}\right)^{\alpha/(\alpha-1)} - \frac{(\hat{\lambda}_j - c_{je})}{\ell_{je}} \left(\frac{\hat{\lambda}_j - c_{je}}{\alpha \ell_{je}}\right)^{1/(\alpha-1)} = (1 - \alpha) \left(\frac{\hat{\lambda}_j - c_{je}}{\alpha \ell_{je}}\right)^{\alpha/(\alpha-1)}$$

Hence, for each machine  $e$ , we want to choose that the job  $j$  that minimizes this expression, which is also the job  $j$  that maximizes the expression  $(\hat{\lambda}_j - c_{je})/\ell_{je}$  since  $\alpha > 1$ . This is precisely the job  $\varphi(e)$  and the proof is hence complete.  $\blacksquare$

**Lemma 2.3** *For all machines  $e$ , job  $\psi(e)$  is feasible choice for  $\varphi(e)$ .*

**Proof.** The line of reasoning is the following:

$$\varphi(e) = \arg \max_j \frac{(\hat{\lambda}_j - c_{je})}{\ell_{je}} = \arg \max_j \left( \delta \cdot \alpha \cdot \left( \sum_{i \leq j} \ell_{je} x_{ie} \right)^{\alpha-1} \right) = \arg \max_j \left( \left( \sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} \right) = \psi(e).$$

The first equality is the definition of  $\varphi(e)$ . For the second equality, observe that for any job  $k$ ,

$$\hat{\lambda}_k \leq \delta \cdot \alpha \cdot \ell_{ek} \left( \sum_{i \leq k} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{ke} \implies \frac{\hat{\lambda}_k - c_{ke}}{\ell_{ke}} \leq \delta \alpha \left( \sum_{i \leq k} \ell_{ie} x_{ie} \right)^{\alpha-1}.$$

The expression on the right is monotone increasing in  $\sum_{i \leq k} \ell_{ie} x_{ie}$ , the load due to jobs up to (and including  $k$ ). Moreover, it is maximized by the last job to assign fractionally to  $e$  (since the inequality is strict for all other jobs). Since this last job is  $\psi(e)$ , the last equality follows.  $\blacksquare$

**Theorem 2.4** *The online greedy algorithm is  $\alpha^\alpha$ -competitive.*

**Proof.** By weak duality it is sufficient to show that  $g(\hat{\lambda}) \geq \text{ON}/\alpha^\alpha$ . Applying Lemma 2.2 to the expression for  $g(\hat{\lambda})$  (equation (2.1)) and substituting the contribution of each machine towards the dual, we get that

$$g(\hat{\lambda}) = \left( \sum_j \hat{\lambda}_j + \sum_e (1 - \alpha) \left( \frac{\hat{\lambda}_{\psi(e)} - c_{\psi(e)e}}{\alpha \ell_{\psi(e)e}} \right)^{\alpha/(\alpha-1)} \right) \quad (2.5)$$

Now we consider only the first term  $\sum_j \hat{\lambda}_j$  and evaluate it.

$$\sum_j \lambda_j = \sum_{j,e} \hat{\lambda}_j \tilde{x}_{je} \quad (2.6)$$

$$= \sum_e \tilde{x}_{je} \left( \sum_j \delta \cdot \alpha \cdot \ell_{je} \left( \sum_{i \leq j} \ell_{ie} \tilde{x}_{ie} \right)^{\alpha-1} + c_{je} \right) \quad (2.7)$$

$$= (\delta \cdot \alpha) \sum_e \sum_j \ell_{je} \tilde{x}_{je} \left( \sum_{i \leq j} \ell_{ie} \tilde{x}_{ie} \right)^{\alpha-1} + \sum_{j,e} \tilde{x}_{je} c_{je} \quad (2.8)$$

$$\geq \delta \sum_e \left( \sum_j \ell_{je} \tilde{x}_{je} \right)^\alpha + \sum_{j,e} \tilde{x}_{je} c_{je} \quad (2.9)$$

Now consider the second term of equation (2.5). Note that if we substitute the value of  $\hat{\lambda}_{\psi(e)}$ , it evaluates to  $(1 - \alpha) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_j \ell_{je} \tilde{x}_{je} \right)^\alpha$ . Putting the above two estimates together, we get

$$g(\hat{\lambda}) \geq \delta \sum_e \left( \sum_j \ell_{je} \tilde{x}_{je} \right)^\alpha + \sum_{j,e} \tilde{x}_{je} c_{je} + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_j \ell_{je} \tilde{x}_{je} \right)^\alpha \quad (2.10)$$



$$= \left( \delta + (1 - \alpha)\delta^{\alpha/(\alpha-1)} \right) \sum_e \left( \sum_j \tilde{x}_{je} \ell_{je} \right)^\alpha + \sum_{j,e} \tilde{x}_{je} c_{je} \quad (2.11)$$

$$\geq \text{ON}/\alpha^\alpha \quad (2.12)$$

The final inequality is due to the choice of  $\delta = 1/\alpha^{\alpha-1}$  which maximizes  $(\delta + (1 - \alpha)\delta^{\alpha/(\alpha-1)})$ .  $\blacksquare$

As observed, e.g., in [AAG<sup>+</sup>95], this  $O(\alpha)^\alpha$  result is the best possible, even for the (fractional) OnGAP problem without any assignment costs. In Section 5, we show how to obtain an  $O(\alpha)^\alpha$ -competitive algorithm for *integer* assignments by a very similar greedy algorithm, and dual-fitting, albeit with a more careful analysis.

### 3 Application to Speed Scaling

We now discuss the application of our results for OnGAP to some well-studied speed scaling problems. In these problems a collection of jobs arrive over time. The  $j^{\text{th}}$  job arrives at time  $r_j$ , and has size/work  $p_j$ . These jobs must be scheduled on a speed scalable processor that can run at any non-negative speed. There is a convex function  $P(s) = s^\alpha$  specifying the dynamic power used by the processor as a function of speed  $s$ . The value of  $\alpha$  is typically around 3 for CMOS based processors. Commonly, one considers objectives  $\mathcal{G}$  of the form  $\mathcal{S} + \beta\mathcal{E}$ , where  $\mathcal{S}$  is a scheduling objective, and  $\mathcal{E}$  is the energy used by the system. Moreover, the scheduling objective  $\mathcal{S}$  is a *fractional* sum objective of the form  $\sum_j \sum_t \frac{x_{jt}}{p_j} C_{jt}$ , where  $C_{jt}$  is the cost of completing job a unit of work of job  $j$  at time  $t$ , and  $x_{jt}$  is the amount of work completed at time  $t$ , or the corresponding *integer* sum objective  $\sum_j \sum_t y_{jt} C_{jt}$ , where  $y_{jt}$  indicates whether or not job  $j$  was completed at time  $t$ . Fractional scheduling objectives are interesting in their own right (for example, in situations where the client gains some benefit from the early partial completion of a job), and are often used in an intermediate step in the analysis of algorithms for integer scheduling objectives.

Normally one thinks of the online scheduling algorithm as having two components: a *job selection policy* to determine the job to run, and a *speed scaling policy* to determine the processor speed. However, one gets a different view when one thinks of the online scheduler as solving online the following mathematical programming formulation of the problem (which is an instance of the OnGAP problem):

$$\begin{aligned} \min \quad & \sum_t \left( \sum_j x_{jt} \right)^\alpha + \sum_j \sum_t C_{jt} x_{jt} \\ \text{subject to} \quad & \sum_t x_{jt} \geq 1 \quad j = 1, \dots, n \end{aligned}$$

Here the variables  $x_{jt}$  specify how much work from job  $j$  is run at time  $t$ . Because we are initially concerned with fractional scheduling objectives, we can assume without loss of generality that all jobs have unit length. The arrival of a job  $j$  corresponds to the arrival of a constraint specifying that job  $j$  must be completed. Greedily raising the primal variables corresponds to committing to complete the work of job  $j$  in the cheapest possible way, given the previous commitments.

**Two Special Cases.** A well-studied speed-scaling problem in this class is when the scheduling objective is energy minimization subject to deadline feasibility [YDS95, BKP07, BBCP11, BCPK09]: for each job  $j$  there is a deadline  $d_j$ , and  $c_{jt} = 0$  for  $t \in [r_j, d_j]$  and is infinite otherwise. Our algorithm for OnGAP is essentially equivalent to the algorithm *Optimal Available (OA)*, introduced in [YDS95] and shown to be  $\alpha^\alpha$ -competitive in [BKP07]—specifically, the speeds set by both OA and our algorithm are the same at all times, but the jobs that are run may be different, since OA uses Earliest Deadline First for scheduling. Our analysis of the online greedy algorithm for OnGAP is an alternate, and simpler, analysis of OA than the potential function analysis in [BKP07]. In this instance, our duality based analysis is tight, as OA is no better than  $\alpha^\alpha$ -competitive [YDS95, BKP07].

Another well-studied special case is when the objective is total flow [AF07, BPS09, BCP09, LLTW08, ALW10, CEL<sup>+</sup>09, CLL10]. That is,  $c_{jt}$  equals  $(t - r_j)$  for  $t \geq r_j$  and infinite otherwise. All prior algorithms for this objective assume some variation of the *balancing* speed scaling algorithm that sets the power equal to the (fractional) number/weight of unfinished jobs. Unlike the earlier example above, OnGAP behaves differently than these balancing algorithms for the following reason. When a job arrives, OnGAP only focuses on choosing assignments which minimize the rate of increase of the objective, and this rule determines both the scheduling policy (in fact, the entire schedule of job  $j$  is decided upon arrival) and the power usage over time. However, in the balancing algorithms the speed profile (and hence power usage) is entirely determined by the scheduling policy, and the scheduling policies used are typically the ones optimal for fractional flow like SJF. Hence it is likely that our algorithms will actually have a different work profile from balancing algorithms.

**A Note about our Approximation Guarantees.** A closer examination of our analysis (especially equation (2.11)) shows that our algorithm has a *Lagrangian multiplier preserving* property: we get that our convex cost  $+ \alpha^\alpha$  times the linear term is at most  $\alpha^\alpha$  times the dual. This separation between the linear and non-linear terms in the objective happens because the constraints are linear, and when we compute the dual, the dual variables are involved in the linear terms whereas the convex terms in the minimizer are identical to their expressions in the primal. In order to argue about the dual minimizer, this somehow forces us to be exact on the linear terms. This can perhaps explain why our analysis is tight for the deadline feasibility and load balancing problems [BKP07, AAF<sup>+</sup>97, AAG<sup>+</sup>95] (where there are no linear terms), but is an  $O(\alpha^\alpha)$ -factor worse than the algorithm of [BCP09] for fractional flow+energy (which has non-trivial linear terms).

**Comparison to Previous Techniques.** In most potential function-based analyses of speed scaling problems in the literature, the potential function is defined to be the future cost for the online algorithm to finish the jobs if the remaining sizes of the jobs were the lags of the jobs, which is how far the online algorithm is behind on the job [IMP11]. A seemingly necessary condition to apply this potential function technique is that there must be a relatively simple algebraic expression for the future cost for the online scheduling algorithm starting from an arbitrary state. As it is not clear how to obtain such an algebraic expression for the most obvious candidate algorithms for nonlinear scheduling objectives,<sup>2</sup> this to date has limited the application of this potential function method to speed scaling problems with linear scheduling objectives. However, our dual-based analysis for OnGAP yields an online greedy speed scaling algorithm that is  $O_\alpha(1)$ -competitive for any sum scheduling objective.

Our algorithm for OnGAP has the advantage that, at the release time of a job, it can commit to the client exactly the times that each portion of the job will be run. One can certainly imagine situations when this information would be useful to the client. Also the OnGAP analysis applies to a wider class of machine environments than does the previous potential function based analyses in the literature. For example, our analysis of the OnGAP algorithm can handle the case that the processor is unavailable at certain times, without any modification to the analysis. Although this generality has the disadvantage that it gives sub-optimal bounds for some problems, such as when the scheduling objective is total flow.

By speeding up the processor by a  $(1+\epsilon)$  factor, one can obtain an online speed scaling algorithm that one can show, using known techniques [BLMSP06, BPS09], has competitive ratios at most  $\min(\alpha^\alpha(1+\epsilon)^\alpha, \frac{1}{\epsilon})$  for the corresponding integer scheduling objective.

---

<sup>2</sup> For  $L_\alpha$  norms of flow on a single fixed speed processor, no potential function is required to prove scalability of natural online algorithms [BP10]. For more complicated non-work conserving machine environments, there are analyses that use a potential function that is a rough approximation of future costs [GIK<sup>+</sup>10, IM10]. But despite some effort, it has not been clear how to extend these potential functions to apply to  $L_\alpha$  norms of flow in the speed scaling setting.



## 4 Routing with Speed Scalable Routers

In this section we consider the following online routing problem. Routing requests in a graph/network arriving over time. The  $j^{\text{th}}$  request consists of a source  $s_j$ , a sink  $t_j$ , and a flow requirement  $f_j$ . In the unsplittable flow version of the problem the online algorithm must route  $f_j$  units of flow along a single  $(s_j, t_j)$ -path. In the splittable flow version of this problem, the online algorithm may partition the  $f_j$  units of flow among a collection of  $(s_j, t_j)$ -paths. In either case, we assume speed scalable network elements (routers, or links, or both) that use power  $\ell^\alpha$  when they have load  $\ell$ , where the load is the sum of the flows through the element. We consider the objective of minimizing the aggregate power. We show that an intuitive online greedy algorithm is  $\alpha^\alpha$ -competitive using the dual function of a mathematical programming formulation as a lower bound to optimal.

**High Level Idea:** The proof will follow the same general approach as for OnGAP: we define dual variables  $\hat{\lambda}_j$  for the demand pairs, but now the minimization problem (which is over flow paths, and not just job assignments) is not so straight-forward: the different edges on a path  $p$  might want to set  $f(p)$  to different values. So we do something seemingly bad: we *relax* the dual to decouple the variables, and allow each (edge, path) pair to choose its own “flow” value  $f(p, e)$ . And which of these should we use as our surrogate for  $f(p)$ ? We use a convex combination  $\sum_{e \in p} h_e f(p, e)$ —where the multipliers  $h(e)$  are chosen *based on the primal loads(!)*, hence capturing which edges are important and which are not.

### 4.1 The Algorithm and Analysis

We first consider the splittable flow version of the problem. Therefore, we can assume without loss of generality that all flow requirements are unit, and all sources and sinks are distinct (so we can associate a unique request  $j(p)$  with each path  $p$ ). This will also allow us to order paths by the order in which flow was sent along the paths. We now model the problem by the following primal optimization formulation:

$$\begin{aligned} \min \quad & \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p) \right)^\alpha \\ \text{subject to} \quad & \sum_{p \in P_j} f(p) \geq 1 \quad j = 1, \dots, n \end{aligned}$$

where  $P_j$  is the set of all  $(s_j, t_j)$  paths, and  $f(p)$  is a non-negative real variable denoting the amount of flow routed on the path  $p$ . In this case, the dual function is:

$$g(\lambda) = \min_{f(p)} \left( \sum_j \lambda_j + \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p) \right)^\alpha - \sum_{j, p \in P_j} \lambda_j f(p) \right)$$

One can think of the dual function as a routing problem with the same instance, but without the constraints that at least a unit of flow must be routed for each request. In the objective, in addition to energy costs, a fixed cost of  $\lambda_j$  is paid for each request  $j$ , and a payment of  $\lambda_j$  is received for each unit of flow routed from  $s_j$  to  $t_j$ .

**Description of the Online Greedy Algorithm:** To route flow for request  $j$ , flow is continuously routed along the paths that will increase costs the least until enough flow is routed to satisfy the request. That is, flow is routed along all  $(s_j, t_j)$  paths  $p$  that minimize  $\sum_{e \in p} \alpha \cdot \left( \sum_{q \leq p: q \ni e} f(q) \right)^{\alpha-1}$ . For analysis purposes, after the flow for request  $j$  is routed, we define

$$\hat{\lambda}_j = \alpha \delta \left( \sum_{e \in p} \sum_{q \leq p: q \ni e} f(q) \right)^{\alpha-1}$$

where  $p$  is any path along which flow for request  $j$  was routed, and  $\delta$  is a constant (later set to  $\frac{1}{\alpha^{\alpha-1}}$ ).

**The Analysis:** Unfortunately, unlike the previous section for load balancing, it is not so clear how to compute the dual function  $g(\hat{\lambda})$  or its minimizer since the variables cannot be nicely decoupled as we did there (per machine). In order to circumvent this difficulty, we consider the following *relaxed* function  $\hat{g}(\hat{\lambda}, h)$ , which does not enforce the constraint that flow must be routed along paths. This enables us to decouple variables and then argue about the objective value. Indeed, let  $\tilde{f}(p)$  be the final flow on path  $p$  for the routing produced by the online algorithm. Let  $h(e) = \alpha \sum_{p \ni e} \tilde{f}(p)^{\alpha-1}$  be the incremental cost of routing additional flow along edge  $e$ , and  $h(p) = \sum_{e \in p} h(e)$  be the incremental cost of routing additional flow along path  $p$ . We then define:

$$\hat{g}(\hat{\lambda}, h) = \min_{f(p,e)} \left( \sum_j \hat{\lambda}_j + \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p,e) \right)^\alpha - \sum_j \hat{\lambda}_j \sum_{p \in P_j} \sum_{e \in p} \frac{h(e)}{h(p)} f(p,e) \right)$$

Conceptually,  $f(p,e)$  can be viewed as the load placed on edge  $e$  by request  $j(p)$ . In  $\hat{g}(\hat{\lambda}, h)$ , the scheduler has the option of increasing the load on individual edges  $e \in p \in P_j$ , but the income from edge  $e$  will be a factor of  $\frac{h(e)}{h(p)}$  less than the income achieved in  $g(\hat{\lambda})$ . In Lemma 4.1 we prove that  $\hat{g}(\hat{\lambda}, h)$  is a lower bound for  $g(\hat{\lambda})$ . We then proceed as in the analysis of OnGAP. Lemma 4.2 shows how the minimizer and value of  $\hat{g}(\hat{\lambda}, h)$  can be computed, and Lemma 4.3 shows how to bound some of the dual variables in terms of the final online primal solution.

**Lemma 4.1** *For the above setting of  $h(\cdot)$ ,  $\hat{g}(\hat{\lambda}, h) \leq g(\hat{\lambda})$ .*

**Proof.** We show that there is a feasible value of  $\hat{g}(\hat{\lambda}, h)$  that is less than  $g(\hat{\lambda})$ . Let the value of  $f(p,e)$  in  $\hat{g}(\hat{\lambda}, h)$  be the same as the value of  $f(p)$  in  $g(\hat{\lambda})$ . Plugging these values for  $f(p,e)$  into the expression for  $\hat{g}(\hat{\lambda}, h)$ , and simplifying, we get:

$$\begin{aligned} \hat{g}(\hat{\lambda}, h) &\leq \sum_j \hat{\lambda}_j + \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p) \right)^\alpha - \sum_j \hat{\lambda}_j \sum_{p \in P_j} f(p) \sum_{e \in p} \frac{h(e)}{h(p)} \\ &= \sum_j \hat{\lambda}_j + \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p) \right)^\alpha - \sum_j \hat{\lambda}_j \sum_{p \in P_j} f(p) \\ &= g(\hat{\lambda}) \end{aligned}$$

The first equality holds by the definitions of  $h(e)$  and  $h(p)$ , and the second equality holds by the optimality of  $f(p)$ . ■

**Lemma 4.2** *There is a minimizer  $\hat{f}$  of  $\hat{g}(\hat{\lambda}, h)$  in which for each edge  $e$ , there is a single path  $p(e)$  such that  $\hat{f}(p,e)$  is positive, and  $\hat{f}(p(e), e) = \left( \frac{\hat{\lambda}_{j(p(e))} h(e)}{\alpha \cdot h(p(e))} \right)^{1/(\alpha-1)}$ .*

**Proof.** The argument that  $\hat{g}(\hat{\lambda}, h)$  has a minimizer where each edge only has flow from one request follows the same reasoning as in the proof of Lemma 2.1. Once we know only one request sends flow on any edge we can use calculus to identify the minimizer and the value which achieves it. Indeed, we get that  $p(e) = \arg \max_{p \ni e} \frac{\hat{\lambda}_{j(p)} h(e)}{h(p)}$ . and the value of  $\hat{f}(p(e), e)$  is set so that the incremental energy cost would just offset the incremental income from routing the flow, that is  $\alpha \hat{f}(p(e), e)^{\alpha-1} = \hat{\lambda}_{j(p(e))} \frac{h(e)}{h(p(e))}$ . Solving for  $\hat{f}(p(e), e)$ , the result follows. ■

**Lemma 4.3**  $\hat{\lambda}_{j(p(e))} \leq \delta \cdot h(p(e))$

**Proof.**  $\hat{\lambda}_{j(p(e))}$  is  $\delta$  times the rate at which the energy cost was increasing for the online algorithm when it routed the last bit of flow for request  $j(p(e))$ .  $h(p(e))$  is the rate at which the energy cost would

increase for the online algorithm if additional flow was pushed along  $p(e)$  after the last request was satisfied. If  $p(e)$  was a path on which the online algorithm routed flow, then the result follows from the fact the online algorithm never decreases the flow on any edge. If  $p(e)$  was not a path on which the online algorithm routed flow, then the result follows from the fact that at the time that the online algorithm was routing flow for request  $j(p(e))$ ,  $p(e)$  was more costly than the selected paths, and the cost can't decrease subsequently due to monotonicity of the flows in the online solution. ■

**Theorem 4.4** *The online greedy algorithm is  $\alpha^\alpha$  competitive.*

**Proof.** We will show that  $\widehat{g}(\widehat{\lambda}, h)$  is at least the online cost ON divided by  $\alpha^\alpha$ , which is sufficient since  $\widehat{g}(\widehat{\lambda}, h)$  is a lower bound to  $g(\widehat{\lambda})$  by Lemma 4.1, and since  $g(\widehat{\lambda})$  is a lower bound to optimal.

$$\widehat{g}(\widehat{\lambda}, h) = \min_{f(p,e)} \left( \sum_j \widehat{\lambda}_j + \sum_e \left( \sum_j \sum_{p \ni e: p \in P_j} f(p, e) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{p \in P_j} \sum_{e \in p} \frac{h(e)}{h(p)} f(p, e) \right) \quad (4.13)$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left( \frac{\widehat{\lambda}_{j(p(e))} h(e)}{\alpha \cdot h(p(e))} \right)^{\alpha/(\alpha-1)} \quad (4.14)$$

$$\geq \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left( \frac{\delta \cdot h(e)}{\alpha} \right)^{\alpha/(\alpha-1)} \quad (4.15)$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \quad (4.16)$$

$$= \sum_j \widehat{\lambda}_j \sum_{p \in P_j} \widetilde{f}(p) - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \quad (4.17)$$

$$= \delta \alpha \sum_j \sum_{p \in P_j} \widetilde{f}(p) \left( \sum_{e \in p} \sum_{q \leq p: q \ni e} \widetilde{f}(q) \right)^{\alpha-1} - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \quad (4.18)$$

$$\geq \delta \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \quad (4.19)$$

$$= \frac{1}{\alpha^\alpha} \sum_e \left( \sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \quad (4.20)$$

$$\geq \text{ON} / \alpha^\alpha \quad (4.21)$$

The equality in line (4.13) is the definition of  $\widehat{g}(\widehat{\lambda}, h)$ . The equality in line (4.14) follows from Lemma 4.2. The inequality in line (4.15) follows from Lemma 4.3. The equality in line (4.16) follows from the definition of  $h(e)$ . The equality in line (4.17) follows from the feasibility of  $\widetilde{f}$ . The equality in line (4.18) follows from the definition of  $\widehat{\lambda}$ . The equality in line (4.19) follows from the definition of  $\delta$ . ■

While the above algorithm only gives a splittable routing, i.e., a fractional routing, we note that the ideas of the next section, Section 5, can be used to obtain an  $O_\alpha(1)$ -competitive algorithm for *integer flow* as well by using a slightly modified primal program (we have to handle non-uniform demands, and also strengthen the basic convex program to prevent some trivial integrality gaps. The next section described how we can handle these issues for the load balancing problem.

## 5 Online Load Balancing: Integral Assignments

For simplicity, let us consider online integer load balancing *without* assignment costs; it is easy to see the extension to the other problems that we consider. In this problem each job has the values  $\ell_{je}$ , and the goal is to integrally assign it to a single machine so as to minimize the sum  $\sum_e (\sum_j X_{je} \ell_{je})^\alpha$  where

$X_{je}$  is the indicator variable for whether job  $j$  is assigned to machine  $e$ . The most natural reduction to our general model OnGAP is to set all  $c_{je}$ 's to 0. However, the convex relaxation for this setting has a large integrality gap with respect to integral solutions. For example, consider the case of just a single job which splits into  $m$  equal parts (where  $m$  is the number of machines)—the integer primal optimal pays a factor of  $m^{\alpha-1}$  times the fractional primal optimal. To handle this case, we add a fixed assignment cost of  $c_{je} = \ell_{je}^\alpha$  for assigning job  $j$  to machine  $e$ . It is easy to see that the cost of an optimal integral solution at most doubles in this relaxation. This is the convex program that we use for the rest of this section.

### 5.1 Approach I: Integer Assignment

In this section, we show that an algorithm which gives an  $O(\alpha)^\alpha$ -competitive ratio. Consider the following greedy algorithm: when job  $j$  arrives, it picks the machine  $e$  that minimizes

$$\delta \cdot \alpha \cdot \ell_{je} \left( \sum_{i < j} \ell_{ie} x_{ie} \right)^{\alpha-1} + \ell_{je}^\alpha,$$

and set  $x_{je} = 1$ . Moreover, set  $\hat{\lambda}_j$  for job  $j$  to be precisely the quantity above. Let  $\tilde{x}_{ie}$  be the final settings of the primal variables.

For the analysis, we again need to set the  $\hat{x}_j$ 's. For machine  $e$ , again consider  $\varphi(e)$  and  $\psi(e)$  as defined in the previous proofs. We can no longer claim that  $\psi(e)$  is a feasible setting for  $\varphi(e)$ . However, we can claim that the load on machine  $e$  that is seen by  $\varphi(e)$  (*and indeed, by any job  $k$* ) is at most the load seen by  $\psi(e)$  when it arrived, *plus* the length  $\ell_{\psi(e)e}$ . Hence

$$\hat{\lambda}_{\varphi(e)} \leq \delta \alpha \ell_{\varphi(e)e} \left( \sum_{i < \psi(e)} \ell_{ie} \tilde{x}_{ie} + \ell_{\psi(e)e} \right)^{\alpha-1} + \ell_{\varphi(e)e}^\alpha \implies \frac{\hat{\lambda}_{\varphi(e)} - \ell_{\varphi(e)e}^\alpha}{\alpha \ell_{\varphi(e)e}} \leq \delta \left( \sum_{i \leq \psi(e)} \ell_{ie} \tilde{x}_{ie} \right)^{\alpha-1}. \quad (5.22)$$

But since  $\psi(e)$  is the last job on machine  $e$ , this last expression is exactly  $\delta \left( \sum_i \ell_{ie} \tilde{x}_{ie} \right)^{\alpha-1}$ . Now recall the dual from (2.5). Observing that  $\alpha > 1$ , we can use the calculations we just did to get

$$\begin{aligned} g(\hat{\lambda}) &\geq \sum_j \hat{\lambda}_j + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \sum_e \left( \sum_i \ell_{ie} \tilde{x}_{ie} \right)^\alpha \\ &= \alpha \delta \sum_{e,j} \ell_{je} \tilde{x}_{je} \left( \sum_{i:i < j} \ell_{ie} \tilde{x}_{ie} \right)^{\alpha-1} + \sum_{j,e} \ell_{je}^\alpha \tilde{x}_{je} + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \left( \sum_{j,e} \ell_{je} \tilde{x}_{je} \right)^\alpha \\ &= \sum_e \left( \alpha \delta \sum_{j \in S_e} \ell_{je} \left( \sum_{i \in S_e: i < j} \ell_{ie} \right)^{\alpha-1} + \sum_{j \in S_e} \ell_{je}^\alpha + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \left( \sum_{j \in S_e} \ell_{je} \right)^\alpha \right) \\ &\geq \frac{1}{e(e(\alpha+1))^\alpha} \times \sum_e \left( \sum_{j \in S_e} \ell_{je} \right)^\alpha \end{aligned}$$

where the last inequality is obtained by applying Lemma 5.1 to the expression for each machine  $e$ . This implies the  $O(\alpha)^\alpha$  competitive ratio for the integral assignment algorithm.

**Lemma 5.1** *Given non-negative numbers  $a_0, a_1, a_2, \dots, a_T$  and  $\delta = (e(\alpha+1))^{\alpha-1}$ , we get*

$$\alpha \delta \sum_{j \in [T]} a_j \left( \sum_{i < j} a_i \right)^{\alpha-1} + \sum_{j \in [T]} a_j^\alpha + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \left( \sum_{j \in [T]} a_j \right)^\alpha \geq \frac{1}{e(e(\alpha+1))^\alpha} \times \left( \sum_{j \in [T]} a_j \right)^\alpha. \quad (5.23)$$

**Proof.** First, consider the case when  $\alpha \geq 2$ : in this case, we bound the LHS of (5.23) when the sequence of numbers is non-decreasing, and then we show the non-decreasing sequence makes this LHS the smallest. We then consider the (easier) case of  $\alpha \in [1, 2]$ .

Suppose  $a_0 \leq a_1 \leq \dots \leq a_T$ , then  $\sum_{j=0}^T a_j (\sum_{i < j} a_i)^{\alpha-1} \geq \sum_{j=0}^{T-1} a_j (\sum_{i \leq j} a_i)^{\alpha-1}$ , and it suffices to (lower) bound the following term

$$\alpha \delta \sum_{j=0}^{T-1} a_j \left( \sum_{i \leq j} a_i \right)^{\alpha-1} + \sum_{j=1}^T a_j^\alpha - (\alpha-1) \delta^{\alpha/(\alpha-1)} \left( \sum_{j=1}^T a_j \right)^\alpha \quad (5.24)$$

There are two cases, depending on the last term: whether  $a_T \leq \frac{1}{\alpha} \sum_{j=0}^{T-1} a_j$ , or not.

- If  $a_T \leq \frac{1}{\alpha} \sum_{j=0}^{T-1} a_j$ , we get  $\sum_{j=0}^T a_j \leq (1 + 1/\alpha) \sum_{j=0}^{T-1} a_j$ . Now, consider the first term in (5.24):

$$\alpha \delta \sum_{j=0}^{T-1} a_j \left( \sum_{i \leq j} a_i \right)^{\alpha-1} \geq \delta \left( \sum_{j=0}^{T-1} a_j \right)^\alpha \geq \frac{\delta}{(1 + 1/\alpha)^\alpha} \left( \sum_{j=0}^T a_j \right)^\alpha \geq \frac{\delta}{e} \left( \sum_{j=0}^T a_j \right)^\alpha.$$

(The last inequality used the fact that  $(1 + 1/\alpha)^\alpha$  approaches  $e$  from below.) Finally, plugging this back into (5.24), ignoring the second sum, and using  $\delta = (e(\alpha + 1))^{1-\alpha}$ , we can lower bound the expression of (5.24) by  $(\sum_{j=0}^T a_j)^\alpha$  times

$$\frac{\delta}{e} - (\alpha-1) \delta^{\alpha/(\alpha-1)} = \frac{1}{e(e(\alpha+1))^{\alpha-1}} - \frac{\alpha-1}{(e(\alpha+1))^\alpha} = \frac{(\alpha+1) - (\alpha-1)}{(e\alpha)^\alpha (1 + 1/\alpha)^\alpha} \geq \frac{2}{e(e\alpha)^\alpha}$$

- In case  $a_T \geq \frac{1}{\alpha} \sum_{j=0}^{T-1} a_j$ , we get  $\sum_j a_j = \sum_{j < T} a_j + a_T \leq (1 + \alpha) a_T$ . Now using just the single term  $a_T^\alpha$  from the first two summations in (5.24), we can lower bound it by

$$a_T^\alpha - (\alpha-1) \delta^{\alpha/(\alpha-1)} (1 + \alpha)^\alpha a_T^\alpha = a_T^\alpha \left( 1 - \frac{(\alpha-1)(1 + \alpha)^\alpha}{(e(\alpha+1))^\alpha} \right) = a_T^\alpha \left( 1 - \frac{\alpha-1}{e^\alpha} \right).$$

This is at least  $a_T^\alpha/2 \geq \frac{1}{2(1+\alpha)^\alpha} (\sum_j a_j)^\alpha \geq \frac{1}{2e\alpha^\alpha}$ .

So in either case the inequality of the statement of Lemma 5.1 is satisfied.

Now to show that the non-decreasing sequence makes the LHS smallest for  $\alpha \geq 2$ . Only the first summation depends on the order, so focus on  $\sum_{j \in [T]} a_j (\sum_{i < j} a_i)^{\alpha-1}$ . Suppose  $a_k > a_{k+1}$ , then let  $a_k = l$ ,  $a_{k+1} = s$ ; moreover, we can scale the numbers so that  $\sum_{i < k} a_i = 1$ . Now swapping  $a_k$  and  $a_{k+1}$  causes a decrease of

$$(a_k - a_{k+1}) \cdot 1^{\alpha-1} + a_{k+1} (1 + a_k)^{\alpha-1} - a_k (1 + a_{k+1})^{\alpha-1} = (l - s) + s(1 + l)^{\alpha-1} - l(1 + s)^{\alpha-1}.$$

And for  $\alpha \geq 2$  this quantity is non-negative.

Finally, for the case  $\alpha \in [1, 2]$ . Note that  $\alpha \delta \leq 1$  for our choice of  $\delta$ , so the LHS of (5.23) is at least

$$\begin{aligned} & \alpha \delta \sum_{j \in [T]} a_j \left( \left( \sum_{i < j} a_i \right)^{\alpha-1} + a_j^{\alpha-1} \right) + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \left( \sum_{j \in [T]} a_j \right)^\alpha \\ & \geq \alpha \delta \sum_{j \in [T]} a_j \left( \sum_{i \leq j} a_i \right)^{\alpha-1} + (1 - \alpha) \delta^{\alpha/(\alpha-1)} \left( \sum_{j \in [T]} a_j \right)^\alpha \end{aligned}$$

The second inequality used the fact that  $a^\beta + b^\beta \geq (a + b)^\beta$  for  $\beta \in (0, 1)$ . Now the proof proceeds as usual and gives us the desired  $O(e\alpha)^\alpha$  bound. ■

## 5.2 Approach II: Randomized Rounding

We now explain a different way of obtaining integer solutions: by rounding (in an online fashion) the fractional solutions obtained for the problems that we consider into integral solutions. While this has a weaker result, it is a simple strategy that may be useful in some contexts.

Suppose we have a fractional solution w.r.t the above parameters (after including the assignment cost). While it is known that the convex programming formulation we use has an integrality gap of 2 [AE05, AKMPS09], these proofs use correlated rounding procedures which we currently are not able to implement online. Instead we analyze the simple online rounding procedure that independently and randomly rounds the fractional assignment. Indeed, suppose we independently assign each job  $j$  to a machine  $e$  with probability  $\tilde{x}_{je}$ . Denote the integer assignment induced by this random experiment by  $Y_{je}$ . Let  $L_e = \sum_j \ell_{je} Y_{je}$  denote the random load on machine  $e$  after the independent randomized rounding. Note that  $L_e$  is a sum of non-negative and independent random variables. We can now use the following inequality (for bounding higher moments of sums of random variables) due to Rosenthal [Ros70, JSZ85] to get that

$$\mathbb{E}[L_e^\alpha]^{1/\alpha} \leq K_\alpha \max \left( \sum_j \mathbb{E}[\ell_{je} Y_{je}], \left( \sum_j \mathbb{E}[\ell_{je}^\alpha Y_{je}^\alpha] \right)^{1/\alpha} \right),$$

where  $K_\alpha = O(\alpha/\log \alpha)$ . However we know that  $\sum_j \mathbb{E}[\ell_{je} Y_{je}] = \sum_j \ell_{je} \tilde{x}_{je}$ , and that  $\sum_j \mathbb{E}[\ell_{je}^\alpha Y_{je}^\alpha] = \sum_j \mathbb{E}[\ell_{je}^\alpha \tilde{x}_{je}] = \sum_j \ell_{je}^\alpha \tilde{x}_{je}$ . Substituting this back in and using  $(a+b)^\alpha \leq 2^{\alpha-1}(a^\alpha + b^\alpha)$ , we get

$$\mathbb{E}[L_e^\alpha] \leq \left( K_\alpha \max \left( \sum_j \ell_{je} \tilde{x}_{je}, \left( \sum_j \ell_{je}^\alpha \tilde{x}_{je} \right)^{1/\alpha} \right) \right)^\alpha \leq 2^{\alpha-1} K_\alpha^\alpha \left( \left( \sum_j \ell_{je} \tilde{x}_{je} \right)^\alpha + \sum_j \ell_{je}^\alpha \tilde{x}_{je} \right).$$

Summing over all  $e$ , we infer that  $\mathbb{E}[\sum_e L_e^\alpha]$  is at most  $(2K_\alpha)^\alpha$  times the value of the online fractional solution objective, and hence at most  $(2\alpha K_\alpha)^\alpha = O(\alpha^2/\log \alpha)^\alpha$  times the integer optimum by Theorem 2.4. (Note that the results of the previous section, and those of [AAG<sup>+</sup>95, Car08] give  $O(\alpha)^\alpha$ -competitive online algorithms for the integer case.)

## 6 Conclusion

The online primal-dual technique (surveyed in [BN07]) has proven to be a widely-systematically-applicable method to analyze online algorithms for problems expressible by linear programs. This paper develops an analogous technique to analyze online algorithms for problems expressible by *nonlinear* programs. The main difference is that in the nonlinear setting one can not disentangle the objective and the constraints in the dual, and hence the arguments for the dual have a somewhat different feel to them than in the linear setting. We apply this technique to several natural nonlinear covering problems, most notably obtaining competitive analysis for greedy algorithms for uniprocessor speed scaling problems with essentially arbitrary scheduling objectives that researchers were not previously able to analyze using the prevailing potential function based analysis techniques.

Independently and concurrently with this work, Anand, Garg and Kumar [AGK12] obtained results that are in the same spirit as the results obtained here. Mostly notably, they showed how to use nonlinear-duality to analyze a greedy algorithm for a multiprocessor speed-scaling problem involving minimizing flow plus energy on unrelated machines. More generally, [AGK12] showed how duality based analyses could be given for several scheduling algorithms that were analyzed in the literature using potential functions.



## References

- [AAF<sup>+</sup>97] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.
- [AAG<sup>+</sup>95] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Load balancing in the  $\ell_p$  norm. In *FOCS*, pages 383–391, 1995.
- [AAZ10] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Minimum-cost network design with (dis)economies of scale. In *FOCS*, pages 585–592, 2010.
- [AAZ11] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Energy-aware scheduling algorithms for network stability. In *INFOCOM*, pages 1359–1367, 2011.
- [AE05] Yossi Azar and Amir Epstein. Convex programming for scheduling unrelated parallel machines. In *STOC’05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 331–337. ACM, New York, 2005.
- [AF07] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49, 2007.
- [AGK12] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, 2012.
- [AKMPS09] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. A unified approach to scheduling on unrelated parallel machines. *J. ACM*, 56(5):Art. 28, 31, 2009.
- [Alb10] Susanne Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.
- [ALW10] Lachlan L. H. Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. In *SIGMETRICS*, pages 37–48, 2010.
- [BBCP11] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. *Algorithmica*, 60(4):877–889, 2011.
- [BCP09] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, 2009.
- [BCPK09] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *ICALP (1)*, pages 144–155, 2009.
- [BKP07] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *JACM*, 54(1), 2007.
- [BLMSP06] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.
- [BN07] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):front matter, 93–263 (2009), 2007.
- [BP10] Nikhil Bansal and Kirk Pruhs. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM J. Comput.*, 39(7):3311–3335, 2010.
- [BPS09] Nikhil Bansal, Kirk Pruhs, and Clifford Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.
- [BV04] Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- [Car08] Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 972–981, New York, 2008. ACM.
- [CEL<sup>+</sup>09] Ho-Leung Chan, Jeff Edmonds, Tak Wah Lam, Lap-Kei Lee, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 255–264, 2009.
- [CLL10] Sze-Hang Chan, Tak Wah Lam, and Lap-Kei Lee. Non-clairvoyant speed scaling for weighted flow time. In *ESA (1)*, pages 23–35, 2010.

- [GIK<sup>+</sup>10] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA*, pages 11–20, 2010.
- [IM10] Sungjin Im and Benjamin Moseley. An online scalable algorithm for average flow time in broadcast scheduling. In *SODA*, pages 1322–1333, 2010.
- [IMP11] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011.
- [JSZ85] W. B. Johnson, G. Schechtman, and J. Zinn. Best constants in moment inequalities for linear combinations of independent and exchangeable random variables. *Ann. Probab.*, 13(1):234–253, 1985.
- [LLTW08] T.W. Lam, L.K. Lee, Isaac To, and P. Wong. Speed scaling functions based for flow time scheduling based on active job count. In *European Symposium on Algorithms*, pages 647–659, 2008.
- [PUW08] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.
- [Ros70] Haskell P. Rosenthal. On the subspaces of  $L^p$  ( $p > 2$ ) spanned by sequences of independent random variables. *Israel J. Math.*, 8:273–303, 1970.
- [YDS95] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. IEEE Symp. Foundations of Computer Science*, pages 374–382, 1995.