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

Anupam Gupta* Ravishankar Krishnaswamy* Kirk Pruhs†

Abstract

We give a principled method to design online algorithms (for potentially non-linear problems) using a mathematical programming formulation of the problem, and also to analyze the competitiveness of the resulting algorithm using the dual program. This method can be viewed as an extension of the online primal-dual method for linear programming problems, to nonlinear programs. We show the application of this method to two online speed scaling problems: one involving scheduling jobs on a speed scalable processor so as to minimize energy plus an arbitrary sum scheduling objective, and one involving routing virtual circuit connection requests in a network of speed scalable routers so as to minimize the aggregate power or energy used by the routers. This analysis shows that competitive algorithms exist for problems that had resisted analysis using the dominant potential function approach in the speed scaling literature, and provides alternate cleaner analysis for other known results. This represents another step towards a principled design and analysis of primal-dual algorithms for online problems.

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

†Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA. kirk@cs.pitt.edu. Supported in part by an IBM Faculty Award, and NSF grants CCF-0830558 and 1115575.

1 Introduction

Speed scalable devices are now a ubiquitous energy management technology. Such devices can be run in high speed and power modes that are energy inefficient, or in low speed and power modes that are more energy efficient, where energy efficiency is the resulting processing speed divided by power investment. The resulting optimization problems involve determining when this improvement in the quality of service provided by running at high speed justifies the resulting inefficient use of significant energy. As the relationship between speed and power in current (and any conceivable) technologies is non-linear, so are the resulting optimization problems. This non-linearity explains in part why we have generally not been able to show that online algorithms are competitive in such settings by reasoning directly about optimal solutions. The dominant algorithm analysis tool in speed scaling settings has been potential functions. But one has to often guess the "right" potential function; moreover, there are situations where the use of potential functions is problematic, most notably when there does not seem to be a simple algebraic expression for the "right" potential for an arbitrary configuration. One motivation of our research is to see if one can use duality to analyze online algorithms for speed scaling problems for which algorithm analysis using potential functions seems problematic.

The first problem we consider involves scheduling a speed scalable processor online, with an objective of the form $\mathcal{E} + \beta\mathcal{S}$, where $\mathcal{E}$ is the energy used by the processor, $\mathcal{S}$ is a scheduling objective that is the sum over jobs of a scheduling cost of the individual jobs, and β expresses the relative value of saving energy versus decreasing the scheduling objective. The input consists of a collection of jobs that arrive over time. The j^{th} job arrives at time r_j, and has size/work p_j. There is a convex function $P(s) = s^\alpha$ specifying the dynamic power used by the processor as a function of speed s, which may be any nonnegative real number. The value of α is typically around 3 for CMOS based processors. A *fractional sum scheduling objective* $\mathcal{S}$ is 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. The corresponding *integer* sum scheduling objective is of the form $\sum_j \sum_t y_{jt} C_{jt}$, where y_{jt} indicates whether or not job j was completed at time t. For linear sum scheduling objectives (where the cost C_{jt} for finishing a job j at time t is a linear function of the flow time $t - r_j$), a potential function based on an algebraic expression for the future cost of the online scheduling algorithm (starting from a particular configuration) has proved to be widely applicable for analyzing natural speed scaling scheduling algorithms [IMP11]. However, the seeming lack of simple algebraic expressions for future online costs of natural online algorithms for nonlinear sum scheduling objectives (like the sum of the squares of flow time) explains in part why, despite some effort, we have not been able to analyze algorithms for such problems.

The second problem that we consider involves online routing of virtual circuit connection requests in a network of speed scalable routers with the objective of minimizing the aggregate power used by the routers. The j^{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), where element e use powers ℓ_e^α, where the load ℓ_e is the sum of the flows through the element. The objective of total aggregate power is then $\sum_e \ell_e^\alpha$. This problem was introduced in [AFZZ10], where a poly-log-approximate *offline* polynomial-time algorithm is given for unsplittable routing; this algorithm classifies the requests by geometrically increasing demands, and randomly rounds a convex program for each demand class.

2 Our Contributions

Very much in the spirit of the online primal dual technique for linear programs [BN07], we give a principled method to design online algorithms (for potentially nonlinear problems) using a mathematical programming formulation of the problem, and also analyze the competitiveness of the resulting algorithm using the Lagrangian dual program. We start by considering a mathematical program for the offline problem. We then interpret the online algorithm as solving this mathematical program online, where the constraints arrive one-by-one; 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. We consider the most natural online greedy algorithm: one that raises the primal variables so that the increase in the primal objective is minimized. For analysis purposes, we set the Lagrangian dual variable corresponding to the new constraint to be proportional to the rate of increase in the primal objective that the online algorithm incurred at the time that the constraint was satisfied. So the setting of the dual variables is determined (up to a multiplicative constant) by the primal programming formulation. By the weak duality property, each feasible value of the dual is a lower bound to the optimal primal solution [BV04]. So we are left to analyze the dual. However, due to nonlinearity, analyzing the dual for a nonlinear program is more complicated than for a linear program. Indeed, in the dual for a nonlinear program, one can not disentangle the objective and the constraints (as one can in the linear case); 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 in the nonlinear case not only involve setting the Lagrangian dual variables, but also involve showing that *every setting* of the copies of the primal variables in the dual still gives a good lower bound. So the analysis has to relate the settings of the copies of the primal variables in the dual with the actual primal variables in the primal. For the speed scaling scheduling and routing problems that we consider, this analysis allows us to conclude that the natural online greedy algorithm is $O_\alpha(1)$-competitive (the subscript means that the constant depends on the constant α). We can also show that the natural online greedy algorithm is $O_\alpha(1)$-competitive for the integer versions of these problems.

Before we give more details about the specific speed scaling problems that we consider, we would like to emphasize that once we formulate the primal non-linear program in the obvious way, the design of the online algorithm and the dual variable settings are naturally derived from this program. The problem specific aspects are confined to the analysis of the dual. Consequently, we feel that our work represents another step towards a principled design and analysis of primal-dual algorithms for online problems. Looking toward the future, it would be ideal if this principled approach could be applied to a wider class of nonlinear online problems.

2.1 Applications to Speed Scaling Scheduling and Routing

Our first observation is that speed scaling scheduling problems with a sum scheduling objective can be reduced to the following more general problem, that we call *Online Generalized Assignment Problem (OnGAP)*. In OnGAP, jobs arrive one-by-one online, and the algorithm must fractionally assign these jobs to one of m machines. When a job j arrives, the online algorithm learns ℓ_{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 α^{th} powers of the machine loads, plus the total assignment cost. (See (4.1) for the convex programming formulation.)

Speed scaling problems with sum scheduling objectives are a special case of OnGAP where each machine is one unit of time when jobs can be scheduled, the assignment cost c_{je} models the scheduling cost for scheduling a unit of job j at time e. This captures most of the speed scaling problems

considered in the literature. If 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, then this is the speed scaling scheduling problem with the objective of minimizing energy subject to deadline feasibility constraints, studied in [YDS95, BKP07, BBCP11, BCPK09], where various algorithms are shown to be $O_\alpha(1)$-competitive using potential functions. If c_{jt} equals $(t - r_j)$ for $t \geq r_j$ and infinite otherwise, then this is the speed scaling scheduling scheduling problem with the objective of total flow plus energy, studied in [AF07, BPS09, BCP09, LLTW08, ALW10, CEL+09, CLL10], where various algorithms are shown to be $O_\alpha(1)$-competitive using potential functions. To capture the objective of sum of fractional flow/response times squared plus energy, we can set the assignment cost c_{je} to be $(e - r_j)^2$ for all times e that are at least the release time r_j of job j, and infinite otherwise.

In Section 4 we apply our primal-dual approach to OnGAP. We observe that the dual for the natural mathematical programming formulation of OnGAP can be computed by a simple greedy algorithm. This allows us show that the natural online greedy algorithm for OnGAP is α^α-competitive with respect to the dual with the specified settings of the dual variables, and hence with respect to optimal. As a consequence, the corresponding online greedy algorithm is α^α-competitive for speed scaling scheduling problems with *any* sum scheduling objective. Recall that previously, competitive analyses were only known for linear sum scheduling objectives. Also for some speed scaling problems, our duality analysis gives a cleaner analysis than the potential function analyses in the literature; compare for example the α^α-competitive analysis of the greedy Optimal Available (OA) algorithm for energy minimization with deadline feasibility constraints given in [BKP07] to our α^α-competitive analysis of the philosophically-similar online greedy algorithm.

Then in Section 5, we make some further comments about the application of these results to speed scaling problems. We can also show that the natural online greedy algorithm is $O_\alpha(1)$-competitive online for the *integer* version of OnGAP, see Appendix A for details. Lower bounds in [AAF+97, AAG+95, BKP07] imply that no deterministic online algorithm can be better than α^α-competitive for OnGAP, and no deterministic online algorithm can be better than α^α-competitive for speed scaling scheduling to minimize energy with feasibility constraints.

Finally in Section 6, we apply our primal-dual approach to the splittable routing problem in a network of speed scalable routers. In this case, the worst-case settings of the copies of the primal variables in the dual are not easy to reason about. To facilitate this, we relax the dual problem in a novel way, by allowing the copies of the primal variables in the dual to take on different values for different edges. To overcome relaxing the flow-constraints, we alter the relaxed objective function (based on the edge loads of our online algorithm!) to ensure that we can still recover enough dual value. This allows us to show the online greedy algorithm is α^α-competitive with respect to the relaxed dual with the specified settings of the dual variables, and hence with respect to optimal. Again we can also show that the natural online greedy algorithm is $O_\alpha(1)$-competitive unsplittable routing, see Appendix A for details.

3 Related Work

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]. A relatively recent survey of the algorithmic power management literature in general, and the speed scaling literature in particular, can be found in [Alb10].

While our reduction from speed scaling scheduling problems to load balancing problems is natural in hindsight, to the best of our knowledge this has not been observed before. This reduction potentially allows the application of techniques from the load balancing literature to speed scaling problems. The

version of OnGAP without assignment costs was studied by [AAF+97, AAG+95], where the online greedy algorithm is shown to be $O_\alpha(1)$-competitive. In their analysis the online cost is bounded by an algebraic expression involving the product of the online cost and the optimal cost, which is disentangled by use of the Cauchy-Schwartz inequality. While this analysis shares some commonalities with both potential function analysis and duality analysis, it is probably best considered a distinct technique. Upon some reflection, one can see that their potential function technique can be used to obtain an alternate analysis that achieves the same bounds as we achieve in this paper by duality. Caragiannis [Car08] gives some refinements to the analysis in [AAF+97, AAG+95] for OnGAP without assignment costs. An offline $O(1)$-approximation (independent of α) was given by [AE05] and [AKMPS09], via solving the convex program and then rounding the solution in a correlated fashion. Finally, offline poly-log-approximation algorithms for the virtual circuit routing problem, when routers have a static power component, can be found in [AAZ10, AFZZ10].

[GIK+10, IM11a, IM11b] show that various online algorithms are competitive, using potential function analysis, for various scheduling problems with fixed speed processors and for the ℓ_k norms of flow objective. The potential functions used in these analyses are motivated the desire to have an algebraic expression for the future costs for the online algorithm, but required some ad-hoc features in order for the algebra to work out. Despite some efforts by the authors of these papers, it is not clear how to extend these potential functions to work in a speed scaling setting.

Independently and concurrently with this work, Anand, Garg and Kumar [AGK11] obtained results that are similar in spirit to 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. Additionally, [AGK11] showed how duality based analyses could be given for several scheduling algorithms that were analyzed in the literature using potential functions. Our results are somewhat different in spirit, with our emphasis being more on a principled methodology for algorithm design and setting of the dual variables. For instance their algorithm for the speed scaling problem in [AGK11] is not derived from the mathematical programming formulation, and the emphasis in [AGK11] is more on obtaining a "dual-fitting" analysis for (in some sense) pre-existing algorithms.

4 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:

$$\begin{aligned} \min \quad & \sum_e \left(\sum_j \ell_{je} x_{je} \right)^\alpha + \sum_e \sum_j c_{je} x_{je} \\ \text{subject to} \quad & \sum_e x_{je} \geq 1 \qquad j = 1, \ldots, n \end{aligned} \tag{4.1}$$

The dual of the primal relaxation is then

$$g(\lambda) = \min_{x \geq 0} \left(\sum_j \lambda_j + \sum_e \left(\sum_j \ell_{je} x_{je} \right)^\alpha + \sum_{j,e} c_{je} x_{je} - \sum_{j,e} \lambda_j \, x_{je} \right) \tag{4.2}$$

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 λ_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 is a lower bound to the optimal primal solution; this is *weak duality* [BV04].

Online Greedy Algorithm Description: Let δ 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 δ, 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

$$\delta \cdot \alpha \cdot \ell_{je} \left(\sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{je} \tag{4.3}$$

are increased until all the work from job j is scheduled, i.e., $\sum_e x_{je} = 1$. Notice that $\alpha \cdot \ell_{je} \big(\sum_{i \leq j} \ell_{ie} x_{ie} \big)^{\alpha-1}$ 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 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[1]. 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 $\widehat{\lambda}_j$ 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

$$\widehat{\lambda}_j := \delta \cdot \alpha \cdot \ell_{je} \left(\sum_{i \leq j} \ell_{ie} x_{ie} \right)^{\alpha-1} + c_{je} \tag{4.4}$$

Intuitively, $\widehat{\lambda}_j$ is a surrogate for the total increase in objective value due to our fractional assignment of job j (we assign a total of 1 unit of job j, and λ_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 $\widetilde{x}$ denote the final value of the x_{je} variables for the online algorithm.

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

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

Observe that the values $\widehat{x}$ could be very different from the values $\widetilde{x}$, and indeed the next few Lemmas try to characterize these values. Lemma 4.1 notes that $\widehat{x}$ only has one job $\varphi(e)$ on each machine e, and Lemma 4.2 shows how to determine $\varphi(e)$ and $\widehat{x}_{\varphi(e)e}$. Then, in Lemma 4.3, 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 4.1 *There is a minimizing solution $\widehat{x}$ such that if $\widehat{x}_{je} > 0$, then $\widehat{x}_{ie} = 0$ for all $i \neq j$.*

[1] 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 4.3

Proof. Suppose for some machine e, there exist distinct jobs i and k such that $\widehat{x}_{ie} > 0$ and $\widehat{x}_{ke} > 0$. 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 $(\sum_j \ell_{je}\widehat{x}_{je})^\alpha$ term fixed and not increase the linear term $\sum_j(\widehat{\lambda}_j - c_{je})\,\widehat{x}_{je}$. This allows us to either set $\widehat{x}_{ie}$ or $\widehat{x}_{ke}$ to zero without increasing the objective. ■

Lemma 4.2 *Define* $\varphi(e) = \arg\max_j \frac{(\widehat{\lambda}_j - c_{je})}{\ell_{je}}$. *Then* $\widehat{x}_{\varphi(e)e} = \frac{1}{\ell_{\varphi(e)e}}\left(\frac{\widehat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha\ell_{\varphi(e)e}}\right)^{1/(\alpha-1)}$ *and* $\widehat{x}_{je} = 0$ *for* $j \neq \varphi(e)$. *Moreover, the contribution of machine* e *towards* $g(\widehat{\lambda})$ *is exactly* $(1-\alpha)\left(\frac{\widehat{\lambda}_{\varphi(e)} - c_{\varphi(e)e}}{\alpha\ell_{\varphi(e)e}}\right)^{\alpha/(\alpha-1)}$.

Proof. By Lemma 4.1 we know that in $\widehat{x}$ there is at most one job (say j, if any) run on machine e. Then the contribution of this machine to the value of $g(\widehat{\lambda})$ is

$$(\ell_{je}\widehat{x}_{je})^\alpha - (\widehat{\lambda}_j - c_{je})\widehat{x}_{je} \tag{4.5}$$

Since $\widehat{x}$ is a minimizer for $g(\widehat{\lambda})$, we know that the partial derivative of the above term evaluates to zero. This gives $\alpha\ell_{je}\cdot(\ell_{je}\widehat{x}_{je})^{\alpha-1} - \left(\widehat{\lambda}_j - c_{je}\right) = 0$, or equivalently, $\widehat{x}_{je} = \frac{1}{\ell_{je}}\left(\frac{\widehat{\lambda}_j - c_{je}}{\alpha\ell_{je}}\right)^{1/(\alpha-1)}$. Substituting into this value of $\widehat{x}_{je}$ into equation (4.5), the contribution of machine e towards the dual $g(\widehat{\lambda})$ is

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

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 $(\widehat{\lambda}_j - c_{je})/\ell_{je}$ since $\alpha > 1$. This is precisely the job $\varphi(e)$ and the proof is hence complete. ■

Lemma 4.3 *For all machines* e, *job* $\psi(e)$ *is feasible choice for* $\varphi(e)$.

Proof. The line of reasoning is the following:

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

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

$$\widehat{\lambda}_k \le \delta\cdot\alpha\cdot\ell_{ek}(\sum_{i\le k}\ell_{ie}x_{ie})^{\alpha-1} + c_{ke} \implies \frac{\widehat{\lambda}_k - c_{ke}}{\ell_{ke}} \le \delta\,\alpha\,(\sum_{i\le k}\ell_{ie}x_{ie})^{\alpha-1}\,.$$

The expression on the right is monotone increasing in $\sum_{i\le k}\ell_{ie}x_{ie}$, 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. ■

Theorem 4.4 *The online greedy algorithm is* α^α*-competitive.*

Proof. By weak duality it is sufficient to show that $g(\widehat{\lambda}) \ge \mathrm{ON}/\alpha^\alpha$. Applying Lemma 4.2 to the expression for $g(\widehat{\lambda})$ (equation (4.2)) and substituting the contribution of each machine towards the dual, we get that

$$g(\widehat{\lambda}) = \left(\sum_j \widehat{\lambda}_j + \sum_e (1-\alpha)\left(\frac{\widehat{\lambda}_{\psi(e)} - c_{\psi(e)e}}{\alpha\ell_{\psi(e)e}}\right)^{\alpha/(\alpha-1)}\right) \tag{4.6}$$

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

$$\sum_j \lambda_j = \sum_{j,e} \widehat{\lambda}_j \widetilde{x}_{je} \tag{4.7}$$

$$= \sum_e \widetilde{x}_{je} \Big(\sum_j \delta \cdot \alpha \cdot \ell_{je} \Big(\sum_{i \le j} \ell_{ie} \widetilde{x}_{ie} \Big)^{\alpha-1} + c_{je} \Big) \tag{4.8}$$

$$= (\delta \cdot \alpha) \sum_e \sum_j \ell_{je} \widetilde{x}_{je} \Big(\sum_{i \le j} \ell_{ie} \widetilde{x}_{ie} \Big)^{\alpha-1} + \sum_{j,e} \widetilde{x}_{je} c_{je} \tag{4.9}$$

$$\ge \delta \sum_e \Big(\sum_j \ell_{je} \widetilde{x}_{je} \Big)^{\alpha} + \sum_{j,e} \widetilde{x}_{je} c_{je} \tag{4.10}$$

Now consider the second term of equation (4.6). Note that if we substitute the value of $\widehat{\lambda}_{\psi(e)}$, it evaluates to $(1-\alpha)\delta^{\alpha/(\alpha-1)} \sum_e \big(\sum_j \ell_{je} \widetilde{x}_{je} \big)^{\alpha}$ Putting the above two estimates together, we get

$$g(\widehat{\lambda}) \ge \delta \sum_e \Big(\sum_j \ell_{je} \widetilde{x}_{je} \Big)^{\alpha} + \sum_{j,e} \widetilde{x}_{je} c_{je} + (1-\alpha)\delta^{\alpha/(\alpha-1)} \sum_e \Big(\sum_j \ell_{je} \widetilde{x}_{je} \Big)^{\alpha} \tag{4.11}$$

$$= \Big(\delta + (1-\alpha)\delta^{\alpha/(\alpha-1)} \Big) \sum_e \Big(\sum_j \widetilde{x}_{je} \ell_{je} \Big)^{\alpha} + \sum_{j,e} \widetilde{x}_{je} c_{je} \tag{4.12}$$

$$\ge \mathrm{ON}/\alpha^{\alpha} \tag{4.13}$$

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

As observed, e.g., in [AAG+95], this $O(\alpha)^{\alpha}$ result is the best possible, even for the (fractional) OnGAP problem without any assignment costs. In Section A, 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.

5 Application to Speed Scaling

We now discuss the application of our results for OnGAP to some well-studied speed scaling problems. 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):

$$\begin{aligned} \min \quad & \sum_t \Big(\sum_j x_{jt} \Big)^{\alpha} + \sum_j \sum_t C_{jt}\, x_{jt} \\ \text{subject to} \quad & \sum_t x_{jt} \ge 1 \qquad j = 1, \ldots, n \end{aligned}$$

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. This greedy algorithm

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. The speed scaling algorithms analyzed in the literature for total (possibly weighted) flow scheduling objectives, are some variation of the *balancing* speed scaling algorithm that sets the power equal to the (fractional) number/weight of unfinished jobs; so for these algorithms, when a job is run generally depends on jobs that arrive in the future.

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.

6 Routing with Speed Scalable Routers

Our analysis will follow the same general approach as for OnGAP: we define dual variables $\widehat{\lambda}_j$ 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.

6.1 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:

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

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 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 λ_j is payed for each request j, and a payment of λ_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 $\sum_{e\in p} \alpha \cdot \left(\sum_{q\le p: q\ni e} f(q)\right)^{\alpha-1}$. For analysis purposes, after the flow for request j is routed, we define

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

where p is any path along which flow for request j was routed, and δ 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 $g(\widehat{\lambda})$ 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 $\widehat{g}(\widehat{\lambda}, h)$, 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 $\widetilde{f}(p)$ be the final flow on path p for the routing produced by the online algorithm. Let $h(e) = \alpha\sum_{p\ni e}\widetilde{f}(p)^{\alpha-1}$ 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:

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

Conceptually, $f(p,e)$ can be viewed as the load placed on edge e by request $j(p)$. In $\widehat{g}(\widehat{\lambda}, h)$, 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 $g(\widehat{\lambda})$. In Lemma 6.1 we prove that $\widehat{g}(\widehat{\lambda}, h)$ is a lower bound for $g(\widehat{\lambda})$. We then proceed as in the analysis of OnGAP. Lemma 6.2 shows how the minimizer and value of $\widehat{g}(\widehat{\lambda}, h)$ can be computed, and Lemma 6.3 shows how to bound some of the dual variables in terms of the final online primal solution.

Lemma 6.1 *For the above setting of $h(\cdot)$, $\widehat{g}(\widehat{\lambda}, h) \le g(\widehat{\lambda})$.*

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

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

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

Proof. The argument that $\widehat{g}(\widehat{\lambda}, h)$ has a minimizer where each edge only has flow from one request follows the same reasoning as in the proof of Lemma 4.1. 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 $p(e) = \arg\max_{p \ni e} \frac{\widehat{\lambda}_{j(p)} h(e)}{h(p)}$. and the value of $\widehat{f}(p(e), e)$ is set so that the incremental energy cost would just offset the incremental income from routing the flow, that is $\alpha\, \widehat{f}(p(e), e)^{\alpha-1} = \widehat{\lambda}_{j(p(e))} \frac{h(e)}{h(p(e))}$. Solving for $\widehat{f}(p(e), e)$, the result follows. ■

Lemma 6.3 $\widehat{\lambda}_{j(p(e))} \leq \delta \cdot h(p(e))$

Proof. $\widehat{\lambda}_{j(p(e))}$ is δ 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 of 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 6.4 *The online greedy algorithm is α^α competitive.*

Proof. We will show that $\widehat{g}(\widehat{\lambda}, h)$ is at least the online cost ON divided by α^α, which is sufficient since $\widehat{g}(\widehat{\lambda}, h)$ is a lower bound to $g(\widehat{\lambda})$ by Lemma 6.1, and since $g(\widehat{\lambda})$ is a lower bound to optimal.

$$\widehat{g}(\widehat{\lambda}, h) = \min_{f(p,e)} \left(\sum_j \widehat{\lambda}_j + \sum_e \left(\sum_j \sum_{p \ni e : p \in P_j} f(p,e) \right)^\alpha - \sum_j \widehat{\lambda}_j \sum_{P \in P_j} \sum_{e \in p} \frac{h(e)}{h(p)} f(p,e) \right) \tag{6.14}$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left(\frac{\widehat{\lambda}_{j(p(e))} h(e)}{\alpha \cdot h(p(e))} \right)^{\alpha/(\alpha-1)} \tag{6.15}$$

$$\geq \sum_j \widehat{\lambda}_j - (\alpha - 1) \sum_e \left(\frac{\delta \cdot h(e)}{\alpha} \right)^{\alpha/(\alpha-1)} \tag{6.16}$$

$$= \sum_j \widehat{\lambda}_j - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.17}$$

$$= \sum_j \widehat{\lambda}_j \sum_{p \in P_j} \widetilde{f}(p) - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.18}$$

$$= \delta \alpha \sum_j \sum_{p \in P_j} \widetilde{f}(p) \left(\sum_{e \in p} \sum_{q \leq p : q \ni e} \widetilde{f}(q) \right)^{\alpha-1} - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.19}$$

$$\geq \delta \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha - (\alpha - 1) \delta^{\alpha/(\alpha-1)} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.20}$$

$$= \frac{1}{\alpha^\alpha} \sum_e \left(\sum_{p \ni e} \widetilde{f}(p) \right)^\alpha \tag{6.21}$$

$$\geq \mathrm{ON}/\alpha^\alpha \tag{6.22}$$

The equality in line (6.14) is the definition of $\widehat{g}(\widehat{\lambda}, h)$. The equality in line (6.15) follows from Lemma 6.2. The inequality in line (6.16) follows from Lemma 6.3. The equality in line (6.17) follows from the definition of $h(e)$. The equality in line (6.18) follows from the feasibility of $\widetilde{f}$. The equality in line (6.19) follows from the definition of $\widehat{\lambda}$. The equality in line (6.20) follows from the definition of δ. ■

References

[AAF+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+95] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Load balancing in the l_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.

[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.

[AFZZ10] Matthew Andrews, Antonio Fernández, Lisa Zhang, and Wenbo Zhao. Routing for energy minimization in the speed scaling model. In *INFOCOM*, pages 2435–2443, 2010.

[AGK11] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, 2011.

[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.

[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 Vandenberghe. *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+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+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.

[IM11a] Sungjin Im and Benjamin Moseley. Online scalable algorithm for minimizing ;k-norms of weighted flow time on unrelated machines. In *SODA*, pages 95–108, 2011.

[IM11b] Sungjin Im and Benjamin Moseley. Online scalable algorithm for minimizing ;k-norms of weighted flow time on unrelated machines. In *SODA*, pages 95–108, 2011.

[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.

[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.

A 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 ℓ_{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.

A.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

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

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

For the analysis, we again need to set the $\widehat{x}_j$'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

$$\widehat{\lambda}_{\varphi(e)} \le \delta\,\alpha\,\ell_{\varphi(e)e}(\sum_{i<\psi(e)} \ell_{ie}\widetilde{x}_{ie} + \ell_{\psi(e)e})^{\alpha-1} + \ell_{\varphi(e)e}^\alpha \implies \frac{\widehat{\lambda}_{\varphi(e)} - \ell_{\varphi(e)e}^\alpha}{\alpha\ell_{\varphi(e)e}} \le \delta\,(\sum_{i\le\psi(e)} \ell_{ie}\widetilde{x}_{ie})^{\alpha-1}. \quad \text{(A.23)}$$

But since $\psi(e)$ is the last job on machine e, this last expression is exactly $\delta\,(\sum_i \ell_{ie}\widetilde{x}_{ie})^{\alpha-1}$. Now recall the dual from (4.6). Observing that that $\alpha > 1$, we can use the calculations we just did to get

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

where the last inequality is obtained by applying Lemma A.1 to the expression for each machine e. This implies the $O(\alpha)^\alpha$ competitive ratio for the integral assignment algorithm.

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

$$\alpha\delta \sum_{j\in[T]} a_j \Big(\sum_{i<j} a_i\Big)^{\alpha-1} + \sum_{j\in[T]} a_j^\alpha + (1-\alpha)\delta^{\alpha/(\alpha-1)} \Big(\sum_{j\in[T]} a_j\Big)^\alpha \;\geq\; \frac{1}{e(e(\alpha+1))^\alpha} \times \Big(\sum_{j\in[T]} a_j\Big)^\alpha. \tag{A.24}$$

Proof. First, consider the case when $\alpha \geq 2$: in this case, we bound the LHS of (A.24) 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 $a_0 \leq a_1 \leq \cdots \leq a_T$, 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

$$\alpha\delta \sum_{j=0}^{T-1} a_j \Big(\sum_{i:i\leq j} a_i\Big)^{\alpha-1} \;+\; \sum_{j=1}^{T} a_j^\alpha \;-\; (\alpha-1)\delta^{\alpha/(\alpha-1)} \Big(\sum_{j=1}^{T} a_j\Big)^\alpha \tag{A.25}$$

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 (A.25):

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

  (The last inequality used the fact that $(1+1/\alpha)^\alpha$ approaches e from below.) Finally, plugging this back into (A.25), ignoring the second sum, and using $\delta = (e(\alpha+1))^{1-\alpha}$, we can lower bound the expression of (A.25) 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^α from the first two summations in (A.25), 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 A.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 δ, so the LHS of (A.24) is at least

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

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. ∎

A.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 $\widetilde{x}_{je}$. 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

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

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

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

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 4.4. (Note that the results of the previous section, and those of [AAG+95, Car08] give $O(\alpha)^\alpha$-competitive online algorithms for the integer case.)