International Journal of Foundations of Computer Science
Vol. 16, No. 3 (2005) 581–598

World Scientific
www.worldscientific.com

ONLINE SCHEDULING OF UNIT JOBS WITH BOUNDED IMPORTANCE RATIO

STANLEY P. Y. FUNG*
Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong
Email: pyfung@cs.hku.hk

FRANCIS Y. L. CHIN*
Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong
Email: chin@cs.hku.hk

HONG SHEN†
Graduate School of Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, Japan
Email: shen@jaist.ac.jp

Received 16 November 2003
Accepted 20 December 2004
Communicated by Sartaj Sahni

We consider the following online scheduling problem. We are given a set of jobs, each having an integral release time and deadline, unit processing length, and a nonnegative real weight. In each time unit one job is to be scheduled, and the objective is to maximize the total value (weight) obtained by scheduling the jobs. This problem arises in the scheduling of packets in network switches supporting quality-of-service (QoS). Previous algorithms for this problem are 2-competitive.
In this paper we propose a new algorithm that achieves an improved competitive ratio when the importance ratio is bounded. Specifically, for job weights within the range $[1..B]$, our algorithm is $2 - 1/(\lceil \lg B \rceil + 2)$-competitive, and the bound is tight.

Keywords: Online scheduling, competitive analysis, importance ratio, quality of service.

1. Introduction

We consider the following simple online scheduling problem. A *job* q is specified by a *release time* $r(q)$, a *deadline* $d(q)$, both of which are integers, and a nonnegative real *weight* $w(q)$, which is the value obtained for processing q. All jobs are of unit length, i.e., require one unit of time to finish execution. Scheduling is done in

*Supported by RGC Grant HKU7142/03E.
†Supported by Japan society for the Promotion of Science (JSPS) under its General Research Scheme B Grant No 14380139.

a uniprocessor setting, and one job can be processed at every unit of time. The objective is to maximize the total value obtained in scheduling the jobs.

We sometimes denote a job by a 3-tuple (r, d, w) representing its release time, deadline and weight. The span of a job, denoted by $span(q)$, is the time interval $[r(q), d(q)]$. A job is *pending* if it is released, not completed and its deadline has not been reached. The scheduling algorithm needs to schedule the jobs *online*, i.e., the jobs are only known when they arrive and the algorithm cannot make changes to the schedule in the past. When a job arrives, all its details are known. Online algorithms are very commonly analyzed in terms of their competitive ratios, introduced in [14]. An online algorithm is *c-competitive* if for any instance of jobs, the value produced by the online algorithm is at least $1/c$ that of the optimal offline algorithm. Competitive analysis and various forms of online scheduling are discussed in detail in [4, 13].

This unit job scheduling problem was first studied in [11] in relation to the transmission of packets in network switches supporting Quality-of-Service (QoS). In those networks, different users can have different guaranteed level of service, and each packet has a weight representing its QoS value. Network switches use this information to determine the priority of packet transmission.

One possible algorithm for this scheduling problem is a greedy one, which always schedules the heaviest job. Another possible algorithm is that, at any time step, schedule according to the optimal schedule of the pending jobs assuming no further jobs will arrive. This optimal schedule can be computed as follows: starting from the heaviest pending job, allocate each job to the latest possible timeslot within its span. For example, given three jobs $q_1(0, 1, 1), q_2(0, 2, 2)$ and $q_3(1, 2, 3)$, the greedy algorithm would schedule $[q_2, q_3]$ in this order, while the current-optimal algorithm just mentioned would schedule $[q_1, q_3]$.

In fact, these two algorithms are described in [5], known as FIRSTFIT (FF) and ENDFIT (EF) respectively. They are proposed in a slightly different context known as *scheduling with partial merits*. In that problem, jobs can have arbitrary lengths, and a partially processed job receives a partial merit proportional to its length processed. This is unlike traditional scheduling [3, 12] in which incomplete jobs get no merit. The partial merit scheduling problem first arises in multimedia content transmission over a network with low bandwidth [5], but it also has other natural applications. For more related results in the partial merit model, see [6, 7, 8].

In [7] we proved a lower bound of $(\sqrt{5}+1)/2 \approx 1.618$ on the competitive ratio for the 'non-timesharing' version of the partial merit model, which can be carried over to this unit job scheduling problem. (The same bound was proved independently in [1] and [10].) The simple greedy algorithm that always schedules the heaviest job is 2-competitive [11, 10]. Since then there are continued efforts to improve the upper bound, but none of them can give competitive ratio $2 - \epsilon$ for some constant ϵ[a]. Hence different special cases have been studied. For example, when all jobs have

[a]Very recently, Chrobak et al. [9] gives a 1.94-competitive algorithm, the first one to break the bound of 2 on general instances. Still, our algorithm has better competitive ratio for reasonable values of B.

bounded span s, the competitive ratio is at most $2 - 2/s + o(1/s)$ [2]. In particular, when $s \leq 3$ there is an optimal 1.618-competitive algorithm [2]. On the other hand, if we allow randomization, a $e/(e-1) \approx 1.58$-competitive algorithm exists [2].

In this paper, we consider the problem from another direction: the case of bounded *importance ratio*, where the importance ratio is defined as the maximum ratio of job weights. We propose an algorithm FIT based on FIRSTFIT and ENDFIT. Both FF and EF are shown to be 2-competitive and their bounds are tight [5]. FF performs poorly when it schedules a job with far-away deadline that is only slightly heavier than another job which is reaching its deadline. On the other hand, EF performs poorly when a very light job is scheduled (because of its earlier deadline) instead of another much heavier job, but this heavy job is eventually discarded because of the future arrival of some other heavy jobs. In fact, these two algorithms seem complementary to each other, in the sense that ENDFIT performs well on those instances which FIRSTFIT performs poorly and vice versa. This suggests that a combination of those two algorithms might be a promising approach for a new algorithm with better performance. The FIT algorithm is a combination of FF and EF. It specifies a condition for choosing either FF or EF at every time step. We show that in the general case, FIT is unfortunately also 2-competitive, and thus no better than other algorithms. However, we are able to show that FIT performs better when the importance ratio B is bounded.

The paper is organized as follows. In Section 2, we describe our new algorithm FIT, and show that it is 2-competitive. In Section 3, we refine the analysis and show that FIT is $(2 - 1/(\lceil \lg B \rceil + 2))$-competitive[b]. This competitiveness bound is tight and can be much less than 2, e.g., FIT is 1.75-competitive when $B = 4$. Section 4 concludes the paper.

2. Algorithm FIT

Intuitively, either q_{max}, the heaviest job, or q_{EF}, the job planned in the first position of the ENDFIT schedule[c], is scheduled, based on their weights. q_{max} is scheduled when $w(q_{max})$ is much larger than $w(q_{EF})$, i.e., $w(q_{max}) > r \cdot w(q_{EF})$ for some parameter $r > 1$, otherwise q_{EF} is scheduled. Formally, for a fixed $r > 1$, our algorithm FIT_r behaves as follows:

> At every integer time, let q_{max} be the heaviest job, and q_{EF} be the first job in an EF schedule. If $w(q_{EF}) < w(q_{max})/r$, q_{max} is scheduled. Otherwise q_{EF} is scheduled.

2.1. Preliminaries

Time is divided into *timeslots* (or simply *slots*) of unit length. For any two slots s and s', $s < s'$ denotes s is earlier than s'. We also use s to denote an instance of

[b]Throughout this paper lg denotes log to base 2.

[c]For jobs with the same weight, we assume that ties are broken in a consistent manner; see [5]. Note that there may be no job planned in the first position in the EF schedule, in which case q_{EF} becomes a null job $(-\infty, \infty, 0)$.

time: we say 'time s' to refer to the moment at the immediate beginning of slot s.

Let OPT and FIT denote the optimal offline schedule and the schedule produced by FIT_r, respectively. $H(s)$ denotes the job in slot s in schedule H; in this paper $H = OPT$ or FIT. An FF-slot is a slot in FIT where q_{max} is scheduled, and similarly an EF-slot is a slot in FIT where q_{EF} is scheduled. (If the two jobs are the same job then it does not matter.)

Let $EFplan$ denote the EF schedule that we compute at every time step. It would be convenient to our analysis to sometimes think of the FIT algorithm as a 2-stage process. Everytime FIT is invoked, it first goes into the EF-stage to compute $EFplan$ for all pending jobs (including the jobs just released), determines q_{EF}, and then goes into the FF-stage to determine whether q_{max} is heavy enough to be chosen instead. We use $EFplan(t, s)$ with $s \geq t$ to denote the job in slot s in this plan generated at time t. A pending job in $EFplan$ is said to be *planned.* In the proofs we sometimes need the notation $EFplan^+(t, s)$, which denotes the $EFplan$ we immediately recompute for the pending jobs after the FF-stage of time t, but before the new jobs from time $t + 1$ arrive. This may be different from $EFplan(t, s)$ since the job chosen in FF-stage is originally in $EFplan$ but not so after the FF-stage.

The main idea of the competitiveness proof is to 'charge' the jobs in OPT to jobs in FIT. One straightforward charging scheme is, for every slot s, charge $OPT(s)$ to $FIT(s)$ if $OPT(s)$ is not too much heavier than $FIT(s)$, otherwise charge $OPT(s)$ to itself in the FIT schedule. In the following we shall show that when the latter case happens, $OPT(s)$ must be in an FF-slot in the FIT schedule.

Lemma 1 *If a new job arrives, the weight of jobs at any slot in EFplan would not decrease.*

Proof. Suppose a job x is planned in a slot s and after some new jobs arrive, job y is planned in s instead, where $w(y) < w(x)$. Since EF plans y in s, and x has higher priority in EF, x must already be planned in some other slot s'. s' cannot be earlier than s because EF plans x in as late a slot as possible. s' cannot be later than s because if this was possible, then before the arrival of new jobs this should also be possible, and the original $EFplan$ would not have planned x in s. Hence contradiction. □

It is easy to see that if a slot s_1 is an EF-slot, $FIT(s_1) = q$, then all slots within $span(q)$ are planned with jobs not lighter than q at time s_1. Lemma 2 shows that this remains true for all later times, even after the arrival of new jobs and/or the existence of some FF-slots in $span(q)$.

Lemma 2 *Let q be a job, $[s_1..s_k]$ be k consecutive slots within $span(q)$, and $FIT(s_1) = q$. If s_1 is an EF-slot, then for all slots s in $[s_1..s_k]$, $w(EFplan(t, s)) \geq w(q)$ for all $t \geq s_1$.*

Proof. We prove by induction on k. The base case $k = 2$ is easy. Suppose the claim is true for an interval of $2, 3, \ldots, k-1$ slots. Now consider $[s_1..s_k]$ where s_1 is an EF-slot and $FIT(s_1) = q$. By induction hypothesis, all slots s in $[s_1..s_{k-1}]$ have $w(EFplan(t, s)) \geq w(q)$ for all $t \geq s_1$. Thus we only need to prove that

$$w(EFplan(t, s_k)) \geq w(q) \text{ for all } t \geq s_1 \tag{1}$$

As explained before, at time s_1, $w(EFplan(s_1, s_k)) \geq w(q)$. Let S be the set of jobs in $EFplan(s_1, s)$ for all $s \geq s_1$. All jobs in S have weights $< r \cdot w(q)$, since otherwise the heaviest one would be scheduled in s_1.

Now (1) is true when $t = s_1$. Consider the next time unit s_2. We have $w(EFplan(s_2, s_k)) \geq w(EFplan(s_1, s_k)) \geq w(q)$ by Lemma 1. If s_2 is an FF-slot, then the job $FIT(s_2)$ cannot be from S since they are not heavy enough ($w(EFplan(s_2, s_2)) \geq w(q)$ by induction hypothesis while jobs in S are lighter than $r \cdot w(q)$). Then $w(EFplan^+(s_2, s_k)) \geq w(EFplan(s_1, s_k)) \geq w(q)$. Hence $w(EFplan(s_3, s_k)) \geq w(EFplan^+(s_2, s_k)) \geq w(q)$ (Lemma 1). We can therefore go on to slot s_3 and so on, each step still having a job no lighter than $w(q)$ in $EFplan$ in slot s_k. Thus (1) holds for any number of consecutive FF-slots after s_1.

If we eventually arrive at some EF-slot $s_j < s_k$, then consider the interval $[s_j..s_k]$: by induction hypothesis $w(EFplan(t, s_k)) \geq w(EFplan(s_j, s_j))$ for all $t \geq s_j$, which is $\geq w(q)$ by induction hypothesis. Thus (1) holds for $t \geq s_j$.

If all slots $[s_2..s_{k-1}]$ are FF-slots, then none of these slots contain jobs in S, thus these jobs are still not scheduled. Thus $w(EFplan(s_k, s_k)) \geq w(EFplan(s_1, s_k)) \geq w(q)$. Thus (1) holds for $t = s_k$.

Therefore in all cases (1) holds. □

The following lemma is the contrapositive version of Lemma 2 (applied to a particular slot s_k). It shows that under certain conditions in s_k, s_1 must be an FF-slot. Since it will be used frequently later, we state it explicitly.

Lemma 3 *Suppose s_1 and s_k are slots ($s_k > s_1$), $FIT(s_1) = OPT(s_k) = q$, and $q' = FIT(s_k)$. If s_k is an EF-slot and $w(q) > w(q')$, or s_k is an FF-slot and $w(q) \geq w(q')/r$, then s_1 must be an FF-slot.*

Lemma 3 shows that when $OPT(s)$ is not too light compared to $FIT(s)$, and $OPT(s)$ is scheduled in FIT earlier, then it must be in an FF-slot. But it does not guarantee the existence of the slot in the first place. The next lemma establishes the existence of a FIT slot for charging jobs in OPT when, at a certain slot s, $OPT(s)$ is much heavier than $FIT(s)$.

Lemma 4 *For a slot s, if $w(OPT(s))/w(FIT(s)) > r$, then there exists an FF-slot $s' < s$ such that $FIT(s') = OPT(s)$.*

Proof. If $OPT(s)$ is not in FIT before s, then it is still pending in FIT at time s, thus the FF-stage of FIT would schedule it, or some heavier job, in s rather than the current $FIT(s)$. Therefore $OPT(s)$ must be in a slot s' in FIT before s. That it is an FF-slot follows from Lemma 3, since we have $w(FIT(s')) = w(OPT(s)) > r \cdot w(FIT(s))$, and therefore satisfy the conditions (in fact, more than enough) in Lemma 3, no matter s is an FF-slot or an EF-slot. □

2.2. Charging Groups

We want to 'charge' the jobs in OPT to jobs in FIT. To achieve a good competitive ratio we need to guarantee that the jobs in OPT are not too much heavier than those in FIT. Naturally, we can charge the OPT job to the FIT job in the same slot. If the OPT job is much heavier, Lemma 4 tells us that this heavy job

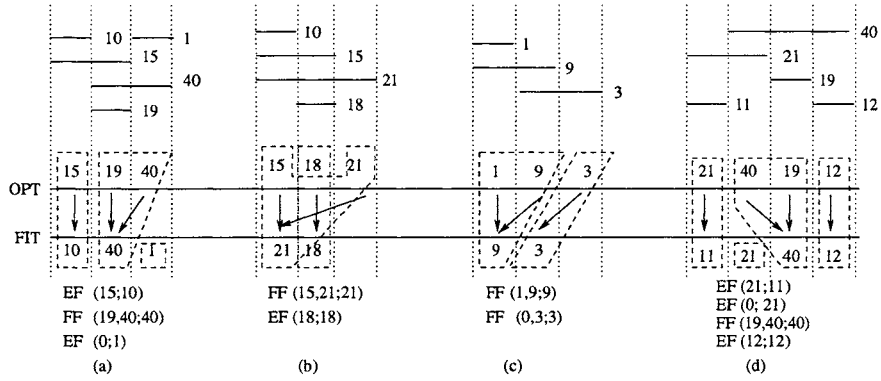

Fig. 1. Charging groups. Numbers represent job weights and we assume $r = 2$.

appeared in FIT in an earlier FF-slot, and we may charge to there instead. This leads us to define the following *charging rule*:

Charging Rule.
For every slot s, if $OPT(s)$ appears in an FF-slot s' in FIT, charge $OPT(s)$ to $FIT(s')$ (note that s' may be earlier than, same as or later than s). Otherwise charge $OPT(s)$ to $FIT(s)$.

Fig. 1 shows some possible scenarios. It is easy to see that all jobs in OPT are charged. Each EF-slot in FIT is charged at most once, each FF-slot is charged at most twice, and charging to a different slot is possible only if the slot being charged is an FF-slot.

A *charging group* of a slot s consists of the job $FIT(s)$ and the jobs in OPT that charge to it. An *FF charging group* of s (or FF-group for short) consists of a job y in an FF-slot s, the same job y in OPT in (possibly) another slot s', and may also include a job x in $OPT(s)$ charging to y in FIT. We denote this by $(x, y; y)$. An *EF charging group* of s (or EF-group) consists of a job y in an EF-slot s and the job x in $OPT(s)$ charging to y. We denote this by $(x; y)$. If the group does not have some charging jobs, we denote it by e.g. $(0, y; y)$ or $(0; y)$. When no confusion arises, the x and y inside the symbols $(x, y; y)$ and $(x; y)$ may also denote the weight of a job instead of the job itself. Fig. 1 shows some charging groups. Define the *charging ratio* of a charging group to be the sum of job weights in OPT to the job weight in FIT of that charging group, i.e., $(w(x) + w(y))/w(y)$ for an FF-group and $w(x)/w(y)$ for an EF-group.

Lemma 5 *The charging ratio of any charging group is at most* $\max(2, r)$.

Proof. Consider any slot s in FIT, and let $x = OPT(s), y = FIT(s)$.

(i) If s is an EF-slot and

(a) $w(x)/w(y) \leq r$ (e.g., Fig. 1(a) first slot), the charging group is $(x; y)$ and the claim holds.

(b) $w(x)/w(y) > r$ (e.g., Fig. 1(a) third slot), then x must exist in an FF-slot $s' < s$ in FIT (Lemma 4). x is therefore charged to $FIT(s')$ and not y.

Thus the charging group is $(0; y)$, with charging ratio $= 0$.

(ii) if s is an FF-slot and

(a) $w(x)/w(y) \leq 1$ (e.g., Fig. 1(a) second slot), the charging group is $(x, y; y)$ with charging ratio $= (w(x) + w(y))/w(y) \leq 2$.

(b) $w(x)/w(y) > 1$ (e.g., Fig. 1(c) second slot), then x must exist in a slot $s' < s$ in FIT (otherwise y is not the heaviest pending job at time s), and s' must be an FF-slot (Lemma 3). Therefore x is charged to $FIT(s')$ and not to y. Thus the charging group is $(0, y; y)$, giving a charging ratio of 1. □

Note from above that an FF-group $(x, y; y)$ cannot have $w(x) > w(y)$, because in this case x would be charged to another FF-group. Similarly an EF-group $(x; y)$ cannot have $w(x) > r \cdot w(y)$.

Theorem 1 *FIT_2 is 2-competitive and the bound is tight. In fact FIT_r is no better than 2-competitive for any value of r.*

Proof. By setting $r = 2$, all charging groups have their charging ratios bounded by 2 (Lemma 5). Thus FIT_2 is 2-competitive.

Suppose $r > 2$. We consider the following instance for large k:

2^k copies of $(0, 2^{k+1}, 1)$;

for $i = k, k-1, \ldots, 0$, 2^i copies of $(2^{k+1} - 2^{i+1}, 2^{k+1}, r^{k+1-i})$;

1 copy of $(2^{k+1} - 1, 2^{k+1}, r^{k+1})$.

Fig. 2 shows the instance and the schedules for $k = 2$. FIT chooses q_{EF} in all slots. The competitive ratio is

$$\frac{2^k r + \cdots + 4r^{k-1} + 2r^k + r^{k+1} + r^{k+1}}{2^k + \cdots + 4r^{k-2} + 2r^{k-1} + r^k + r^{k+1}} \approx \frac{r + r(1 + \frac{2}{r} + \frac{4}{r^2} + \cdots)}{r + (1 + \frac{2}{r} + \frac{4}{r^2} + \cdots)} = \frac{r + r(\frac{1}{1-2/r})}{r + (\frac{1}{1-2/r})} = 2$$

Suppose $r < 2$. Consider the following instance: (Fig. 3)

for $i = k, k-1, \ldots, 0$, 2^i copies of $(2^{k+1} - 2^{i+1}, 2^{k+1}, r^{k-i})$;

1 copy of $(2^{k+1} - 1, 2^{k+1}, r^k)$;

for $0 \leq i \leq 2^{k+1} - 1$, $(i, i + 2^{k+1} + 1, r^{f(i)} + \epsilon)$ where $f(i) = k + 1 - \lceil \lg(2^{k+1} - i) \rceil$.

FIT chooses q_{max} in all slots. The competitive ratio is (neglecting the small ϵ)

$$\frac{2(r^k + 2r^{k-1} + 4r^{k-2} + \cdots + 2^k) + r^k + r^{k+1}}{(r^k + 2r^{k-1} + 4r^{k-2} + \cdots + 2^k) + r^{k+1}} = \frac{2 - 2(2/r)^{k+1} - 1 - 2/r + r}{1 - (2/r)^{k+1} + r - 2}$$

when k is large and $r < 2$, $(2/r)^k$ dominates, giving a ratio of 2.

When $r = 2$, both schedules gives a competitive ratio 2 when k is large enough. □

3. Competitiveness of FIT with Bounded B

Let B denote the maximum ratio of job weights, i.e., all job weights w satisfy $1 \leq w \leq B$. This is called the *importance ratio*. The constructions in Theorem 1 produce instances with unbounded importance ratio. In real applications, the importance ratio tends to be a small constant or may be known in advance; hence B can be

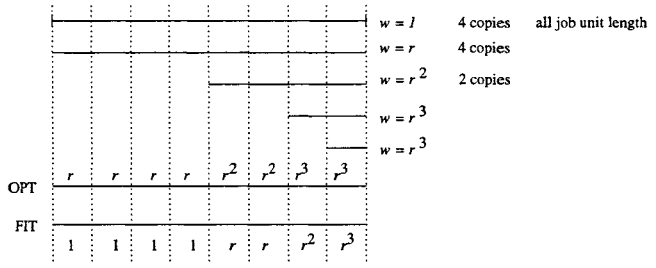

Fig. 2. The instance for Theorem 1 ($r > 2$) and the schedules.

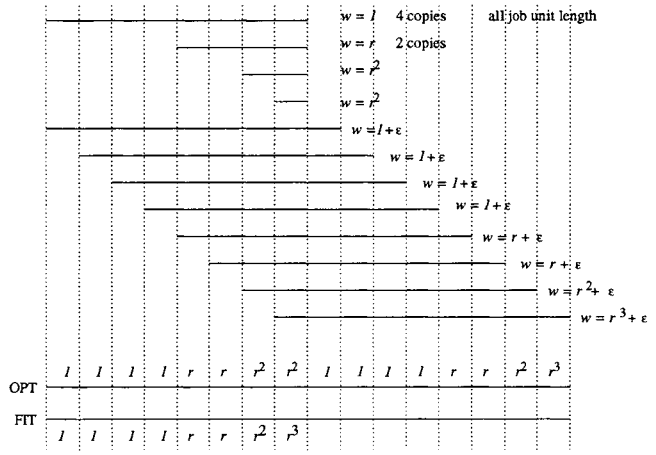

Fig. 3. The instance for Theorem 1 ($r < 2$) and the schedules.

considered as part of the input and bounded. The following subsections show that charging groups can be 'linked' together, and prove that such linkages can be used to lower the competitive ratio in the analysis.

3.1. Links

When we consider each individual charging group, the charging ratio 2 is tight only in case (i)(a) and (ii)(a) of Lemma 5. In those cases where the OPT job, charging to the FIT job in the same slot, is sufficiently large, the OPT job is actually planned in a later slot in $EFplan$ at that time. For example, in Fig. 1(a),(b), the job with weight 15 is originally planned in the second slot, only to be discarded later due to arrival of a heavier job. This heavier job belongs to another charging group. In the following, we show that when a charging group has charging ratio larger than a certain value, a 'link' can be generated to connect it to another charging group (later we will see that this value is 1 for EF-group and $1 + 1/r$ for FF-group). We shall make sure that the charging group being connected to is 'heavier' than the previous group. The links may continue (i.e., this group may link to yet another group), but this linking cannot be continued indefinitely because there is a bound on B. It will terminate when it reaches a group that has a small charging ratio. We show that, when we consider the linked charging groups together (rather than

each one separately), the overall charging ratio is smaller.

Formally, a *link* $x_i \to y_j$ points from the OPT job of a charging group ('x_i' in a FF-group $(x_i, y_i; y_i)$ or EF-group $(x_i; y_i)$) to the FIT job y_j of another charging group at a later slot. Initially, it represents the situation that the OPT job is planned in a later slot in FIT while OPT schedules it in the current slot. These planned jobs, however, may later be dropped out of the schedule because of the arrival of heavier jobs. To model this, we require all links to satisfy the following two *link constraints*:

- **(weight constraint)** *For any link* $x_i \to y_j$, *if* y_j *is in an EF-slot, then* $w(y_j) \geq w(x_i)$; *if* y_j *is in an FF-slot, then* $w(y_j) \geq r \cdot w(x_i)$.
- **(span constraint)** *For all links* $x_i \to y_j$, y_j *must be in a slot within* $span(x_i)$.

We construct the links incrementally. We simulate the operation of FIT slot-by-slot and sweep the OPT and FIT schedules from beginning to end. We maintain that at any time, all links satisfy the link constraints. As we will see shortly, some links point to unswept slots (i.e. in $EFplan$) during this process, and therefore these links have to satisfy the weight constraints for EF-slots.

When a slot s is first swept, the following *linking rules* are used to determine when links are generated and where they initially point to. (This refers to the time after the EF-stage. The location may need to be changed subsequently when we come to the FF-stage, or when we further sweep the schedules; see Lemma 7.)

Linking Rules. For each FIT slot s, a link is generated if and only if:

(i) s is an EF-slot in a charging group $(x; y)$, $1 < w(x)/w(y) \leq r$.
(ii) s is an FF-slot in a charging group $(x, y; y)$, $1/r \leq w(x)/w(y) \leq 1$.

In both cases, link x to $FIT(s'')$, where s'' is a FIT slot such that $EFplan(s, s'') = x$.

The following lemma shows that charging groups that have a charging ratio 'too large' must be able to generate a link according to the above rules.

Lemma 6 *For any EF-group with charging ratio* > 1, *or any FF-group with charging ratio* $\geq 1 + 1/r$, *a link can always be generated according to the above rules, while satisfying the link constraints.*

Proof. It is clear that any charging group with ratio larger than the specified limits belongs to either one of the cases in the linking rules, and that the link generated satisfies the link constraints. It remains to prove that x is in $EFplan$.

(i) s is an EF-slot in a charging group $(x; y)$, $1 < w(x)/w(y) \leq r$. If x is not pending at time s, i.e., it exists in FIT in a slot $s' < s$, then s' must be an FF-slot (Lemma 3). Hence x is charged to $FIT(s')$ instead, contradicting $(x; y)$ being a charging group. Thus x is still pending, and must be planned in a slot $s'' > s$ (otherwise y will not be scheduled in s since $w(x) > w(y)$). Thus there must be such a slot s'' to be linked. (e.g., Fig. 1(a) first group.)

(ii) s is an FF-slot in a charging group $(x, y; y)$, $1/r \leq w(x)/w(y) \leq 1$. The proof is similar to (i): If x is not pending at time s, i.e., it exists in a slot $s' < s$ in FIT, then s' must be an FF-slot (Lemma 3). Hence x is charged to $FIT(s')$

instead, contradicting $(x, y; y)$ being a charging group. If x is scheduled in slot s in FIT (i.e. $x = y$), the charging group is actually $(0, y; y)$ with charging ratio $= 1$ (and hence no link). Otherwise, x is still pending, and x must be planned in a slot $s'' > s$, since $w(x) \geq w(y)/r > (r \cdot w(EFplan(s, s)))/r = w(EFplan(s, s))$. Thus there must be such a slot s'' to be linked. (e.g., Fig. 1(b) first group.)

All links generated as above satisfy the weight constraint (since both ends of the link are actually the same job) and the span constraint (since the OPT job must be planned in a feasible slot). □

A *linking slot* is a slot s such that $OPT(s)$ has a link pointing out. Other slots are called *non-linking slots*. When $EFplan$ changes (e.g. updated to $EFplan^+$ after the FF-stage, or a new slot is swept), links may need to be rearranged to satisfy the weight constraints. We only change links that point to slots after the currently-swept slot. In the following, we show that links can be rearranged when sweeping the schedules to maintain the following invariants:

- Only the current slot is pointed to by at most two links; all other future slots are pointed to by at most one link.
- All links satisfy the link constraints.

For each slot s being swept, we invoke the link-rearranging algorithm **Link-Arrange**, described as follows:

Case 1. If s is an EF-slot and generates a new link $x \to y_j$, and if there is an existing link $z \to y_j$, then relink z to $EFplan(s, z)$, or to s if z is not planned at time s. Repeat this relink process if there is another link pointing to $EFplan(s, z)$, until finally such a link is relinked to slot s.

Case 2. If s is an FF-slot, perform the same operation as if s is an EF-slot, based on $EFplan$ (not $EFplan^+$, the updated $EFplan$ after the FF-stage). After we update $EFplan$ to $EFplan^+$, it is possible that some links now violate the weight constraint for $EFplan^+$. For every such link $z \to y_k$, it can be shown that z must be planned in a slot after y_k (Claim 1 below). If no link points to this slot, move the link to point to this slot (i.e., $z \to y_k$ becomes $z \to z$). Otherwise we have a link $z' \to z$. Swap the links to get $z \to z$ and $z' \to y_k$. Repeat the process if this new link $z' \to y_k$ violates the weight constraint.

We first observe the following, which follows from the definition of links:

Observation 1 *If an OPT slot contains the 'y'-job in an FF-group $(x, y; y)$, no links are generated from this OPT slot.*

Claim 1 *If a link $z \to y_k$ violates the weight constraint, z must be planned in a slot after y_k.*

Proof. Consider where z is in FIT. There are only four possibilities, the first three satisfying the weight constraints:

(i) If z is already scheduled in FIT, it must be in an EF-slot (Observation 1), and by Lemma 2 , $w(z) \leq w(y_k)$;

(ii) If z is planned in $EFplan^+$ earlier than (or same as) the position of y_k, then the slot of y_k is within the span of z by span constraint. By the property of

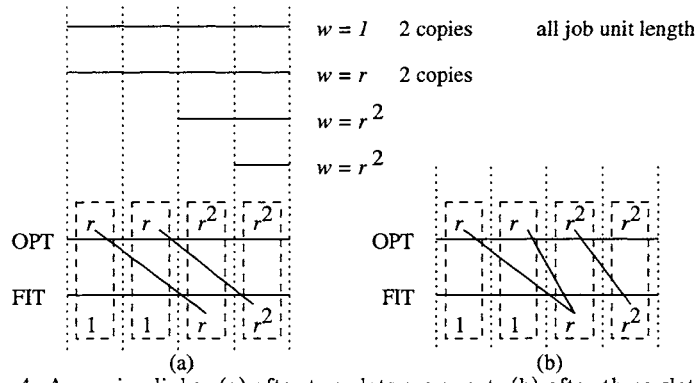

Fig. 4. Arranging links: (a) after two slots are swept; (b) after three slots are swept.

EF, $w(z) \leq w(y_k)$;

(iii) If z is not in $EFplan^+$, then by the property of EF, $w(z) \leq w(y_k)$;

(iv) z is planned in a slot after y_k. □

Fig. 4 shows an example of arranging links.

Lemma 7 *By the above algorithm, the links are arranged so that after each slot is swept, each linking slot is pointed to by at most two links, and each non-linking slot is pointed to by at most one link. All link constraints are satisfied.*

Proof. It is easy to see that the first invariant is maintained by the algorithm after sweeping every slot. It can also be seen from the algorithm that a non-linking slot will not be pointed to by more than one link. These imply the first part of the lemma. The second part of the lemma is true if we can maintain the second invariant. Suppose the invariant holds before we sweep slot s. Now we come to slot s. Let $x = OPT(s)$.

Case 1: s is an EF-slot. It is easy to see that the link constraints are satisfied if x does not generate a new link, or generates a link to a slot not pointed by other links. We just keep the link positions unchanged, although $EFplan$ has possibly changed (thus links may point to some other jobs). By Lemma 1 the job weights at any slot will not decrease, thus all links still satisfy the weight constraint.

The invariant is also maintained when links are redirected to other slots as long as the job is in $EFplan$. However if the redirection finishes at slot s because a link $z \to y_j$ cannot be redirected, i.e., z is not in $EFplan$, then either

a. z is already scheduled in FIT before s. It is not an FF-slot since no link is generated from this job in a FF-group (Observation 1). Thus it is an EF-slot, then by Lemma 2, $w(FIT(s)) \geq w(z)$; or

b. z is pending but not in $EFplan$. Then we also have $w(FIT(s)) \geq w(z)$, or else z would be scheduled in s since this is within its span.

In both cases $w(FIT(s)) \geq w(z)$, thus satisfying the weight constraint.

Case 2: s is an FF-slot. The proof is similar to that of Case 1, except that the weight of the job in s is at least r times heavier and the link(s) to s should also satisfy the weight constraint. We now show that the redirection (swapping) of links

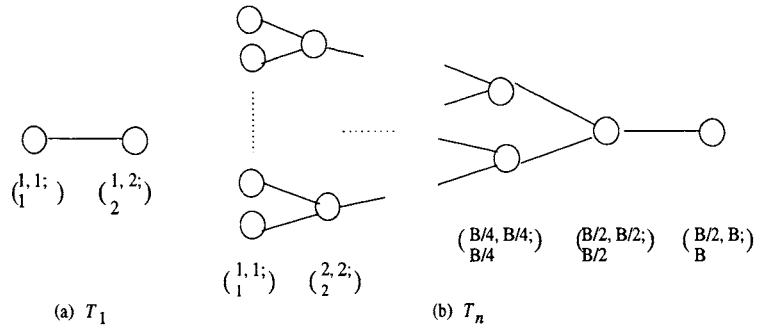

Fig. 5. The 'worst' charging tree. (a) T_1; (b) T_n.

in the extra step for Case 2 of **Link-Arrange** ensures that the link constraints are satisfied and that it must terminate. If no link points to z, moving the link $z \to y_k$ to $z \to z$ maintains the invariants. Otherwise suppose $z' \to z$. We swap the links $z \to y_k$ and $z' \to z$ to become $z \to z$ and $z' \to y_k$. Both links satisfy the span constraint. If $w(z') \leq w(y_k)$, we are done. Otherwise **Link-Arrange** similarly finds where z' occurs in FIT, and continues moving/swapping links until the weight constraint is satisfied. This process must terminate since there is a finite number of jobs in *EFplan* and each move/swap increase the number of links of the form $z \to z$. □

3.2. Charging Trees

A *charging tree* is a binary tree whose nodes are either EF-groups (called *EF-nodes*) or FF-groups (called *FF-nodes*). Nodes in the tree are connected by links, that connect charging groups as described in the previous subsection. That is, each non-root EF-node is represented by $(x_i; y_i)$ with $y_i < x_i \leq ry_i$ and non-root FF-node by $(x_i, y_i; y_i)$ with $y_i/r \leq x_i \leq y_i$. The links have to satisfy the weight constraint: a link $x_i \to y_j$ from node i to node j will ensure that $x_i \leq y_j$ if j is an EF-node, and $rx_i \leq y_j$ if j is an FF-node. The *charging ratio* of a charging tree is defined as

$$\frac{\Sigma_{FFnode}(x_i + y_i) + \Sigma_{EFnode}(x_i)}{\Sigma_{FFnode}(y_i) + \Sigma_{EFnode}(y_i)}$$

summing over all nodes in the tree.

Given a pair of OPT and FIT schedules, the charging groups are linked together to form a forest of charging trees, as described in the previous subsection. Each such charging tree is a binary tree since each charging group can be pointed by at most two links (Lemma 7). However the root cannot have two children because, by Lemma 7, it would then generate another link itself and thus not a root. If we can bound the charging ratio of each charging tree, then the competitive ratio of FIT is also bounded by the same ratio.

From now on we set $r = 2$, and for simplicity assume $B = 2^n$ for some positive integer n.

Consider the charging tree T_n for $B = 2^n$ in Fig. 5. T_n is a full binary tree

(except at the root). All nodes in the same level are identical. All nodes are FF-nodes, 'doubling' the job weights at all levels (except the root), and has $n+1$ levels of nodes. We will see later that T_n is indeed realizable, i.e., there exists an instance of jobs such that the OPT and FIT schedules produce charging groups and links corresponding to this tree. We are going to show that any charging tree with importance ratio bounded by B will not have a charging ratio larger than that of T_n, thus providing an upper bound of the competitive ratio.

The following two lemmas, with proof in Appendix, help us identify charging trees with the maximum possible charging ratio.

Lemma 8 *Given any charging tree T with importance ratio B, there always exists a full charging tree (except at the root), with the same nodes at each level, such that its importance ratio is no more than B and has charging ratio no less than that of T.*

Lemma 9 *To find the charging tree with the maximum charging ratio, we only need to consider those having the following properties:*
Property 1: *All nodes are FF-nodes. For any FF-node $(x,y;y)$ with $y \leq B/2$, we have $x = y$.*
Property 2: *There are at least two levels of nodes, with leaves $(1,1;1)$ and parents-of-leaves $(x,2;2)$ for some x.*
Property 3: *It has $2^{n-1} = B/2$ leaves.*

Hence, we denote a charging tree by a sequence of nodes $[(x_1,y_1;y_1)\ldots(x_k,y_k;y_k)]$, with $(x_1,y_1;y_1)$ on the leaves side and $(x_k,y_k;y_k)$ being the root.

Lemma 10 *The charging tree T_n has the maximum possible charging ratio among all charging trees with importance ratio $B = 2^n$.*

Proof. We only need to consider those charging trees with properties as stated in Lemmas 8 and 9. We prove the lemma by induction on n. For the base case $n = 1$, i.e. $B = 2$, we show that $T_1 = [(1,1;1)\ (1,2;2)]$ is a worst tree (Fig. 5(a)), as follows. Consider a tree $[(1,1;1)\ (x,2;2)\ \ldots]$. We have $x \geq 1$ or otherwise the importance ratio is greater than 2. If these are the only two nodes, then we have $x = 1$, and we are done. So suppose there are more nodes. We still have $x = 1$, since if $x > 1$ the next FF-node must be $(x',y;y)$ with $y > 2 = B$. The next node must be of the form $(x',2;2)$ again, since it must be an FF-node with job weights restricted by importance ratio and weight constraints. We can similarly show that $x' = 1$. Hence the tree is $[(1,2;2)\ldots(1,2;2)]$, which has the same charging ratio as T_1.

Suppose T_p is the tree having the maximum charging ratio for $B = 2^p$. We want to find the worst charging tree with importance ratio $2B = 2^{p+1}$. We know by Lemma 9 that the leaves of this tree are $(1,1;1)$, and parents of leaves are $(x,2;2)$. Since $n > 1$, we can apply Property 1 of Lemma 9 to show that $x = 2$. Consider the subtree T' of this tree by ignoring the leaves. It is easy to see that no jobs in T' have weights smaller than 2, in order for T' to satisfy the weight constraint. T' must itself be a worst tree, i.e., having the maximum charging ratio among all trees with job weights between 2 and $2B$. This is because, if there is another such tree T'' with a larger charging ratio, then T' can be replaced by T'' as long as both

having the same number of leaves. This increases the overall charging ratio. That they have the same number of leaves is guaranteed by Property 3 of Lemma 9.

By induction hypothesis, the worst tree with job weights between 1 and B is T_p. Therefore, the worst tree with job weights between 2 and $2B$ is a scaled-up version of this, i.e. $[(2,2;2) \ldots (B,B;B)(B,2B;2B)]$. Combining with the leaves, the worst tree with job weights between 1 and $2B$ is $[(1,1;1)\ (2,2;2) \ldots (B,B;B)(B,2B;2B)]$, which is T_{p+1}. Thus the claim is true. □

Lemma 11 *T_n has a charging ratio of $(2\lg B+3)/(\lg B+2)$ when $B=2^n$.*

Proof. T_n consists of one root node $(B/2,B;B)$, one node $(B/2,B/2;B/2)$, two nodes $(B/4,B/4;B/4)$, ..., 2^{k-1} nodes $(B/2^k,B/2^k;B/2^k)$, ..., and 2^{n-1} leaf nodes (1,1;1). Thus the charging ratio of T_n is

$$c=\frac{\sum_{k=1}^{n}2^{k-1}(B/2^k+B/2^k)+(B/2+B)}{\sum_{k=1}^{n}2^{k-1}(B/2^k)+B}=\frac{nB+3B/2}{nB/2+B}=\frac{2n+3}{n+2}=\frac{2\lg B+3}{\lg B+2}$$

□

Theorem 2 *FIT_2 gives a competitive ratio of $2-1/(\lceil\lg B\rceil+2)$, and the bound is tight.*

Proof. The competitive ratio of FIT is bounded by the charging ratio of T_n. When B is not an exact power of 2, we can replace B by the smallest power of 2 that is larger than B. Thus the result follows from Lemma 11. The bound is tight, as the worst-case charging tree T_n corresponds to the instance in Fig. 3 by putting $r=2$. □

3.3. Comparisons

Comparing to the FIRSTFIT and ENDFIT algorithms, FIT_2 performs better in the bounded importance ratio case. It is easy to see that FF remains 2-competitive even when the importance ratio B is very small, but EF might perform better when B is bounded. However we still have:

Lemma 12 *The competitive ratio of EF is at least $2-1/(B+1)$.*

Proof. Consider three jobs $q_1=(0,1,1), q_2=(0,2,B)$ and $q_3=(1,2,B)$. OPT schedules q_2 and q_3 giving total value $2B$ while EF schedules q_1 and q_3 giving total value $1+B$. □

Fig. 6 shows a comparison between EF and FIT_2 for bounded B. FIT_2 outperforms EF in the competitive ratio when $B>11$.

4. Concluding Remarks

In this paper we give a new algorithm for the online scheduling of unit length jobs. We show that it is 2-competitive in general, and has improved competitive ratio $2-1/(\lceil\lg B\rceil+2)$ when the importance ratio B of the jobs is bounded. There were no deterministic algorithm having competitive ratio $2-\epsilon$ for constant $\epsilon>0$, until a very recent paper that gives a 1.94-competitive algorithm [9]. The best lower bound for this problem is 1.618. How to close this gap remains an open problem.

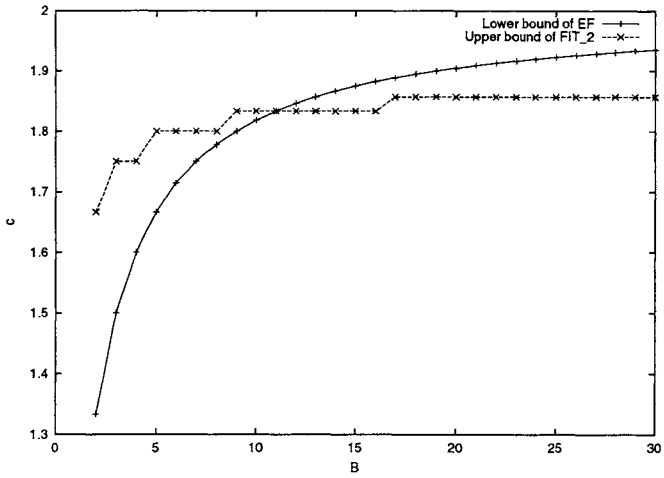

Fig. 6. Comparing the competitive ratios of EF and FIT_2.

References

1. N. Andelman, Y. Mansour and A. Zhu, "Competitive queueing policies for QoS switches", in *Proceedings of 14th ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 761–770.
2. Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall and T. Tichý, "Online competitive algorithms for maximizing weighted throughput of unit jobs", in *Proceedings of 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, 2004, pp. 187–198.
3. S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha and F. Wang, "On the competitiveness of on-line real-time task scheduling", *Real-Time Systems* **4** (1992) 125–144.
4. A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.
5. E.-C. Chang and C. Yap, "Competitive online scheduling with level of service", in *Proceedings of 7th International Computing and Combinatorics Conference*, volume 2108 of *Lecture Notes in Computer Science*, 2001, pp. 453–462.
6. F. Y. L. Chin and S. P. Y. Fung, "Improved competitive algorithms for online scheduling with partial job values, in *Proceedings of 9th International Computing and Combinatorics Conference*, volume 2697 of *Lecture Notes in Computer Science*, 2003, pp. 425–434.
7. F. Y. L. Chin and S. P. Y. Fung, "Online scheduling with partial job values: does timesharing or randomization help?", *Algorithmica*, **37**(3) (2003) 149–164.
8. M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý and N. Vakhania, "Preemptive scheduling in overloaded systems", *Journal of Computer and System Sciences* **67**(1) (2003) 183–197.
9. M. Chrobak, W. Jawor, J. Sgall and T. Tichý, "Improved online algorithms for buffer management in QoS switches", in *Proceedings of 12th European Symposium on Algorithms*, 2004, pp. 204–215.
10. B. Hajek, "On the competitiveness of on-line scheduling of unit-length packets with

hard deadlines in slotted time", in *Proceedings of Conference on Information Sciences and Systems*, 2001.

11. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber and M. Sviridenko, "Buffer overflow management in QoS switches", in *Proceedings of 33th ACM Symposium on Theory of Computing*, 2001, pp. 520–529.
12. G. Koren and D. Shasha, "D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems", *SIAM Journal on Computing* **24** (1995) 318–339.
13. J. Sgall, "Online scheduling", in *Online Algorithms: the State of the Art*, (edited by A. Fiat and G. J. Woeginger), volume 1442 of *Lecture Notes in Computer Science*, Springer-Verlag, 1998, pp.196–227.
14. D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules", *Communications of the ACM* **28**(2) (1985) 202–208.

Appendix A: Proofs of Lemmas

Lemma 8 Proof. We show that any charging tree can be transformed to a full binary tree with the same nodes at each level. For any subtree rooted at an internal node v, consider the left and right subtrees (including empty subtrees). If the two subtrees are not identical, replace the left subtree with an identical copy of the right subtree, or vice versa; one of these operations will increase the charging ratio. The reason is as follows. Suppose a node contributes x and y to the numerator and denominator, respectively, of the charging ratio. The left subtree contribute a (resp. b) to the numerator (resp. denominator) to the charging ratio, and similarly c, d for the right subtree. (An empty subtree contributes 0 to both numerator and denominator.) Charging ratio $= \frac{x+a+c}{y+b+d}$. Replace right subtree by left subtree changes the ratio to $\frac{x+2a}{y+2b}$. Without loss of generality, assume $\frac{x+2a}{y+2b} \geq \frac{x+2c}{y+2d}$. Then by simple property of ratios we have $\frac{x+2a}{y+2b} \geq \frac{x+2a+x+2c}{y+2b+y+2d} \geq \frac{x+2c}{y+2d}$, and hence $\frac{x+2a}{y+2b} \geq \frac{x+a+c}{y+b+d}$. Thus this replacement will not decrease the charging ratio. Apply this process to all nodes level-by-level, starting from the leaves, gives a full binary tree (except at the root) with same nodes in the same level. □

Lemma 9 Proof. We show that any charging tree not satisfying the properties can be transformed into another one that satisfies them, such that the charging ratio of the whole charging tree would not decrease, and no link constraints are violated. We use the fact that whenever we add 2δ or more to the numerator of the charging ratio and δ to the denominator, for any positive δ, the charging ratio would not decrease. (Note that the charging ratio is, by Theorem 1, at most 2.)

Property 1: First, we show that the root is an FF-node. Suppose the root is an EF-node $(x; y)$. Without loss of generality we can change it to $(y; y)$. Denote its single child node v by $(z; w)$ or $(z, w; w)$, where $z \leq y$. Again we can increase z to y. Now, replace the root with an FF-node $(y/2, y; y)$ and change v to $(y/2; w)$ or $(y/2, w; w)$. This does not change the charging ratio. All links are still valid. If the charging ratio of v is too small, i.e. $w > y/2$ for an EF-node or $w > y$ for an FF-node, we increase the value of y to restore them. It can be verified that the charging ratio would not decrease. If the charging ratio of v is too large, i.e.

$w < y/4$ for an EF-node or $w < y/2$ for an FF-node, we increase the value of w and change its children v'. For example, if its children are $(p;q)$, we increase w by δ so that the charging ratio of v is within limit, and increase p by $\delta/2$. Similar change applies when v' are FF-nodes. It can be verified that this does not decrease the charging ratio. We may need to change v' in turn if its charging ratio is too large, and so on.

Second, we change the rest of EF-nodes to FF-nodes. Because of weight constraint and the fact that the root is an FF-node, all EF-nodes $(x_i;y_i)$ in the tree have $y_i \leq B/2$. Each of them can be changed into an FF-node $(x_i, 2y_i; 2y_i)$. It is easy to see that this does not violate any weight or span constraints.

All nodes now become FF-nodes. Finally we show that node $(x,y;y)$ can be changed to $(y,y;y)$ if $y \leq B/2$. Consider a node $v_1(x_1,y_1;y_1)$ in a charging tree, with parent $v_2(x_2,y_2;y_2)$ $(y_1 \leq B/2)$. We show how to transform v_1 and v_2 such that v_1 becomes $(y_1,y_1;y_1)$.

(a) Suppose v_2 has two v_1 as children. Increase x_1 to y_1, then increase y_2 to $2y_1$ if $y_2 < 2y_1$ (otherwise do nothing to y_2). Addition to numerator $= 2(y_1 - x_1) + (2y_1 - y_2)$. Addition to denominator $= (2y_1 - y_2)$. Since $2x_1 \leq y_2$, we have $2(y_1 - x_1) \geq 2y_1 - y_2$. This implies addition to numerator at least twice the addition to denominator. (This is also trivially true for the case of y_2 not increased.) Thus this will not decrease the charging ratio.

(b) Suppose v_2 only has one v_1 as a child, i.e. v_2 is the root. In this case, we can change x_2 without affecting any links. Increase x_1 to y_1, then increase y_2 to $2y_1$ and x_2 to y_1 if $y_2 < 2y_1$ (otherwise do nothing to x_2 and y_2). Addition to numerator $= (y_1 - x_1) + (2y_1 - y_2) + (y_1 - x_2)$. Addition to denominator $= 2y_1 - y_2$. Since $2x_2 \leq y_2, 2x_1 \leq y_2$, we have $x_1 + x_2 \leq y_2$. This implies the addition to numerator is at least twice the addition to denominator.

Property 2: Suppose it only has one level (i.e. a root node). The maximum charging ratio is attained by the node (1,2;2), and thus we can add a leaf (1,1;1) to this node.

Suppose the leaves are $(x_1,y_1;y_1)$. By changing them to $(x_1,x_1;x_1)$, the charging ratio is increased. Next consider the parents of the leaves, $(x_2,y_2;y_2)$ where $y_2 \geq 2x_1$ because of the weight constraint. If $y_2 > 2x_1$, we can, without decreasing the charging ratio, scale up the leaves from $(x_1,x_1;x_1)$ to $(y_2/2, y_2/2; y_2/2)$, and then scale down the whole tree so that the lowest two levels are (1,1;1), $(x$,2;2) (because of the fact that the leaves have charging ratio exactly 2, while all other nodes have charging ratio at most 2).

Property 3: If a charging tree has more than 2^{n-1} leaves, then by Lemma 8 and Properties 1, 2 above, it must have at least 2^n leaves and the importance ratio will be larger than B.

Suppose a tree T has fewer than 2^{n-1} leaves. We construct another tree having double the number of leaves and the same charging ratio. Let R be the root of T and T' be the tree formed by removing R from T (since R has only one child, T' is still a tree). Consider a tree U formed by root R, root's only child $R' = R$, and

the two subtrees of R' being both T'. This is a valid charging tree satisfying all link constraints, and has double the number of leaves than that of T. Repeat the construction as necessary until the number of leaves is sufficient. □