

# Scheduling Resource Allocation with Timeslot Penalty for Changeover

Amrinder Arora\*      Fanchun Jin      Hyeong-Ah Choi

Department of Computer Science  
The George Washington University  
Washington, DC 20052  
e-mail: {amrinder,jinfc,hchoi}@gwu.edu

## Abstract

Given a time slotted list of resource capacities, we address the problem of scheduling resource allocation considering that a change in allocation results in the changeover penalty of one timeslot. The goal is to maximize the overall allocation of resources. We prove that no 1-lookahead algorithm can be better than  $8/5$ -competitive. We provide improved analysis of Wait Dominate Hold (WDH) algorithm that was previously known to be 4-competitive. We prove that WDH is  $8/3$ -competitive. We also consider  $k$ -lookahead algorithms, and prove lower bound of  $(k+2)/(k+1)$  on their competitiveness and give an online algorithm that is 2-competitive.

*Keywords:* Online scheduling, changeover cost, resource allocation, wireless scheduling

## 1 Introduction

We consider an online scheduling problem with changeover costs defined as follows.

**Problem**  $\mathcal{TRAC}$  - Time slotted Resource Allocation with Changeover penalty:

**Given:** Sequence of resource capacities  $\mathbf{C} = [c(1), c(2), c(3), \dots, c(n)]$

**To Find:** Allocation  $\mathbf{X} = [x(1), x(2), x(3), \dots, x(n)]$ , such that:

$x(i) \leq c(i)$  for  $1 \leq i \leq n$       (*Capacity constraint*)

If  $x(i) \neq x(i+2)$ , then  $x(i+1) = 0$       (*Timeslot penalty for Changeover*)

**Objective Function:** To maximize  $\sum_{i=1}^n x(i)$

An example problem instance of the  $\mathcal{TRAC}$  problem and a feasible solution are shown in Table 1. We can easily observe that due to the timeslot penalty for changeover, a feasible solution to the problem must contain a 0 between unequal resource allocations.

---

\*Corresponding Author: 801 22nd St. NW., Room 730, Washington DC 20052. Phone: 202 994 4217, Fax: 202 994 4875 Email: amrinder@gwu.edu

Table 1: A specific problem instance of  $\mathcal{TRAC}$  and a feasible solution

$C:$	23	25	4	7	16	33	66	9	8	7	6	1
$X:$	23	23	0	0	0	33	33	0	7	7	0	1

## 1.1 Motivation

The problem is directly motivated from the study of wireless networks. The capacities of the wireless channels can change frequently, and the end points of the wireless channel can adjust the transmission parameters to adapt to the channel condition. However, such an adjustment for transmission parameters needs the end points to communicate to adjust the data transfer configuration, which causes a loss of timeslots (the “changeover” cost). The end points may decide to transmit data at lesser than available capacity, if by doing so, the changeover costs decrease. The interesting practical problem is to find a trade off between the benefit from the adjustment and the penalty caused by adjustment.

## 1.2 Background and Previous Work

Scheduling is one of the fundamental computer science problems. Basic setting of scheduling involves resources (machines) and jobs (tasks) that need to be scheduled, such that some metric (usually the makespan, that is the total length of the schedule) is minimized. Many of the results in online scheduling can be attributed back to the 1966 paper by Graham [1], which was one of the first papers to consider the online scheduling problem, and to consider competitive analysis. The technique of competitive analysis, now commonly used in the context of online algorithms, compares the performance of online algorithm to that of an optimal offline algorithm. If the performance of an online algorithm  $\mathcal{A}$  is at most  $c$ -times “worse” than the performance of an optimal offline algorithm, then the algorithm  $\mathcal{A}$  is said to be  $c$ -competitive. In his landmark paper, Graham presented the list scheduling algorithm and proved that the competitive ratio of list scheduling algorithm is  $2 - \frac{1}{m}$ , where  $m$  is the number of machines. Graham’s work was finally improved after a hiatus of almost 30 years by Bartal, Fiat, Karloff, and Vohra’s construction of an algorithm presented in [2]. Bartal et al’s algorithm is 1.986 competitive, and was the first known algorithm with competitive ratio bounded less than 2 for all machines. That result was improved by Karger, Phillips and Torng in [3], who presented an algorithm with competitive ratio of 1.945. In [4], Albers further improved the bound to 1.923-competitive by presenting an algorithm based on a different strategy, and also improved the lower bound to 1.852.

Since Graham’s statement of scheduling problem, many different variations of online scheduling have also been considered, such as preemptive scheduling, precedence constraints, release times, deadlines, conflicting jobs, unknown running times etc. A complete taxonomy of different variations of scheduling algorithms can be found in the textbook [5]. Optimal algorithm for online scheduling of parallel jobs with dependencies has been presented in [6]. Some significant papers in different variations of online scheduling include [7], [8] and [9].

With respect to scheduling in wireless networks, Andrews and Zhang have presented excellent results on admissibility of flows in [10]. Kalyanasundaram, Pruhs and Velauthapillai have presented algorithm for minimizing average response time in [11]. Other significant papers for the applications of scheduling problem in wireless networks include [12], [13], [14] etc.

### 1.3 Our Results

We focus entirely on the technique of competitive analysis. Our current work is an extension of results earlier presented in [15], [16] and [17]. In those papers, it was shown that no online algorithm with finite lookahead can be optimal. A dynamic programming algorithm for the offline solution was given, that executes in  $O(n^3)$  time, where  $n$  is the number of timeslots. A 1-lookahead online algorithm (Wait Dominate Hold algorithm) was also presented, and it was shown to be 4-competitive.

In this paper, we improve the results significantly, and also consider new problem variations. After observing that an algorithm with no lookahead cannot be  $c$ -competitive for any  $c$ , we focus on 1-lookahead algorithms. We prove that no 1-lookahead algorithm can be better than  $8/5$ -competitive. The proof uses an adaptive offline adversary, and the central idea is that adversary increases the capacity if the online algorithm does not use a timeslot, and keeps it constant if the adversary uses a timeslot. Without considering the boundary condition, it can be easily shown that the adversary can always achieve twice the total allocation as the online algorithm. Since the resource capacity keeps increasing in the adversary’s constructed example, we focus on the boundary condition closely, and prove that even including the boundary condition, no 1-lookahead algorithm can be better than  $8/5$ -competitive.

We provide improved analysis of Wait Dominate Hold (WDH) algorithm. We prove that WDH is  $8/3$ -competitive. We use a novel technique of tracking internal states of the algorithm in terms of a finite state automata, and analyze all different sentences that can be possibly generated by the WDH algorithm. We hope that the same technique can be used to analyze other algorithms as well.

We also consider  $k$ -lookahead algorithms for the resource allocation problem, and prove lower bound of  $(k + 2)/(k + 1)$  using the adversary approach. As expected, this lower bound approaches 1 as  $k$  increases.

We give a simple online algorithm that is 2-competitive when there is at least 2 lookahead available. Before giving that algorithm, we present two general families of online  $k$ -lookahead algorithms - the Optimal Block Algorithm (OPTB) and Optimal Sliding Window Algorithm (OPTSW). We believe that OPTB and OPTSW can be considered as general frameworks for many online problems that use the concept of “timeslot” or “step” and where the lookahead is available.

A summary of our results is presented in Table 2.

Table 2: Summary of results presented in this paper

<b>Lookahead</b>	1	2	$k > 2$
<b>Lower Bound</b>	8/5	4/3	$(k + 2)/(k + 1)$
<b>Upper Bound</b>	8/3	2	2

## 1.4 Structure of the Paper

This paper is organized as follows. This section covers the problem statement, motivation and the previous work done in the related fields. Section 2 presents the lower bound for all one lookahead algorithms for  $\mathcal{TRAC}$  problem. In the Section 3, we present an improved analysis of the Wait Dominate Hold algorithm and prove that it is 8/3-competitive, improving the earlier known analysis of 4-competitive. We consider the  $k$ -lookahead variation of the problem in Section 4 and present lower and upper bounds. Our conclusions in Section 5 complete the paper.

## 2 Lower Bound Analysis for 1-Lookahead Algorithms

Firstly, we note that for online scheduling problem with one timeslot penalty as changeover cost, an algorithm with 0-lookahead cannot be  $c$ -competitive for any value of  $c$ . As an example, suppose that capacity of the first timeslot is 1. If an online algorithm with no lookahead does not use resource at all in this timeslot, then the adversary can simply terminate and use the resource fully. If the online algorithm uses the resource (either partially, or fully) in this timeslot, then the adversary can use a high value of resource capacity in the second timeslot, which the algorithm will be unable to use. In either case, the competitive ratio is not bounded by any constant.

In this section, we prove a lower bound on all 1-lookahead algorithms. Consider a deterministic 1-lookahead online algorithm  $\mathcal{A}$ . For the purpose of this discussion, let  $a(t)$  denote the output selected by algorithm  $\mathcal{A}$  during timeslot  $t$ , let  $x(t)$  denote the output selected by the constructed offline solution, and let  $|A|$  denote the total allocation achieved by  $\mathcal{A}$ .

Consider input of  $1, 2, c(3)$ , where value of  $c(3)$  is controlled by an adaptive online adversary.

If algorithm  $\mathcal{A}$  chooses  $a(1) > 0$  in the first slot, adversary sets the value of  $c(3) = a(1) - \epsilon$ . In this case, the algorithm can only achieve a total allocation of  $2a(1)$ , while optimal total allocation is  $3a(1) - 3\epsilon$ . If algorithm chooses  $a(1) = 0$  in the first slot, adversary sets the value of  $c(3) = 1$ . In this case, the algorithm can only achieve a total allocation of 2, while optimal total allocation is 3. Thus, no 1-lookahead online algorithm can achieve a total allocation of more than  $2/3$  times that of optimal.

This counterexample proves a lower bound of  $3/2$  on competitive ratio of 1-lookahead algorithms. Next, we formulate the following adversary strategy to prove a better lower bound:

**Adversary Strategy:** Start with an input of  $c(1) = 1$  and  $c(2) = 2$ . Set the value of  $c(t)$ , for all  $t > 2$  based on choice of online algorithm as per following rules:

*Adversary Rule 1:* If  $a(t-2) = 0$ , set  $c(t) = 2c(t-1)$

*Adversary Rule 2:* If  $a(t-2) > 0$ , set  $c(t) = c(t-1)$

Based on online algorithm  $\mathcal{A}$ , a sample input and algorithm can be like:

$$\begin{array}{cccccccccccc} c(t): & 1 & 2 & \dots & 2^{k_1} & 2^{k_1+1} & \dots & 2^{k_1+1} & 2^{k_1+1} & 2^{k_1+1} & 2^{k_1+2} \\ a(t): & 0 & 0 & \dots & a_1 & \dots & \dots & a_1 & 0 & & \end{array}$$

**Constructing an offline solution:** Consider an offline solution, that uses the full capacity on any timeslot that succeeds a timeslot where algorithm's choice is not 0. That is, if  $a(t) > 0$ , then set  $x(t+1) = c(t+1)$ .

**Lemma 1** *Offline solution as defined above is a valid solution.*

**Proof:** To see that the offline solution as defined above is a valid solution, we only need to prove that if for any two consecutive timeslots, offline solution allocates non zero outputs, then both the values are the same. Suppose, during timeslots  $t+1$  and  $t+2$ , the offline allocation is non-zero. Therefore, by definition,  $a(t) > 0$  and  $a(t+1) > 0$ . Since  $a(t) > 0$ , by definition of capacity matrix,  $c(t+2) = c(t+1)$ . Thus, by construction of offline solution,  $x(t+2) = c(t+2) = c(t+1) = x(t+1)$ . Therefore, the offline solution as constructed above is a valid solution. ■

**Theorem 1** *Excluding boundary condition, offline solution as constructed above achieves a total value of twice that of the one step lookahead online algorithm  $\mathcal{A}$ .*

**Proof:** We prove this claim by demonstrating a bijection between the allocations made by  $\mathcal{A}$  and the constructed offline solution. Specifically, we prove using induction that for each timeslot  $t$ , for which  $a(t) > 0$ ,  $x(t+1) \geq 2a(t)$ .

*Induction Base:* Clearly, this claim is true for  $t = 1$ , as  $c(1) = 1$  and  $c(2) = 2$ . If  $a(1) > 0$ , then  $x(2) = 2 \geq 2a(1)$ .

*Induction Hypothesis:* Let us assume that the claim is true for all  $t < T$ .

*Induction Step:* Suppose  $a(T) > 0$ , then by definition,  $x(T+1) = c(T+1)$ , and there are two cases: If  $a(T-1) = 0$ , then  $c(T+1) = 2c(T)$ , and thus  $x(T+1) \geq 2a(T)$ .

If  $a(T-1) > 0$ , then  $a(T) = a(T-1)$  and  $c(T+1) = c(T)$ , and thus  $x(T+1) = x(T)$ . Using induction hypothesis,  $x(T) \geq 2a(T-1)$ . Thus,  $x(T+1) \geq 2a(T)$ .

Thus, having proved a bijection, we know that excluding boundary conditions,  $\sum_t x(t) \geq 2\sum_t a(t)$ . ■

## 2.1 Fixing Boundary Condition

The discussion in the preceding sections presents an intuitive idea for adversary. However, it does not suggest a suitable end point for the algorithm. For example, if the online algorithm continues to allocate  $a(t) = 0$ , the adversary continues to set  $c(t+1) = 2c(t)$  ad infinitum. In this section, we present a method to find a suitable end point. We intend to prove that no online algorithm can be better than  $c$ -competitive, where  $c = 8/5 - \epsilon$ .

To prove that, we first generalize the previous strategy, by replacing the constant 2 with a constant  $\alpha$ , where  $\alpha \in (5/3, 2]$ . Next, we make two observations. Both of these observations are in the context of the adversary strategy presented above.

**Lemma 2** *If the online algorithm  $\mathcal{A}$  uses a contiguous block of  $\lceil \frac{5}{3\alpha-5} \rceil$  elements, then the adversary can achieve a ratio of  $5/3$ .*

**Proof:** Consider an input sequence as follows:

$$\begin{array}{l} c(t): \quad 1 \quad \alpha \quad \dots \quad \alpha \quad \alpha \quad 0 \\ a(t): \quad 1 \quad 1 \quad \dots \quad 1 \end{array}$$

If algorithm uses a contiguous block of  $k$  elements, the adversary can stop by putting a 0 in the last timeslot. In this case,  $\mathcal{A}$  achieves total allocation of  $k + 1$ , and adversary achieves total allocation of  $\alpha k$ , that is, a ratio of  $\alpha k / (k + 1)$ , which is more than  $5/3$  if  $k \geq 5 / (3\alpha - 5)$ .

**Lemma 3** *If the online algorithm  $\mathcal{A}$  does not use any time slot in a contiguous block of  $k$  elements (assuming  $k$  is odd), then the adversary can achieve a ratio of  $\frac{5}{3} - \frac{1}{3 \cdot \alpha^k}$ .*

**Proof:** Consider an input sequence as follows:

$$\begin{array}{l} c(t): \quad 1 \quad \alpha \quad \dots \quad \alpha^{k-1} \quad \alpha^k \quad \alpha^{k-1} \\ a(t): \quad 0 \quad 0 \quad \dots \quad 0 \end{array}$$

In this situation,  $\mathcal{A}$  can achieve maximum allocation of  $\alpha^k$ . Optimal offline algorithm can achieve  $3 \cdot \alpha^{k-1}$  from the last 3 timeslots and  $\sum_{i=0}^{k-3} \alpha^i$  by using alternate timeslots from 1 to  $k - 2$ . That is, optimal offline algorithm can achieve a value of  $5/3 \cdot \alpha^k - 1/3$ . Thus,  $\frac{\text{OPT}}{|A|} \geq \frac{5}{3} - \frac{1}{3 \cdot \alpha^k}$ . ■

**Corollary 1** *If the online algorithm  $\mathcal{A}$  does not use a time slot indefinitely, then the adversary can achieve a ratio of  $\frac{5}{3} - \epsilon$ , for any  $\epsilon > 0$ .*

**Proof:** Immediate from the previous result, by using an appropriate value of  $k = \lceil -\log_\alpha(3\epsilon) \rceil$ . ■

We define one more concept that will be helpful in the main theorem.

$$\begin{aligned} h(\alpha, i) &\stackrel{\text{def}}{=} \text{Optimum offline allocation in sequence } 1, \alpha, \alpha^2, \dots, \alpha^{i-1} \text{ not using the last timeslot} \\ &= 0 \text{ if } i = 1 \\ &\geq \frac{\alpha^i - 1}{\alpha^2 - 1} \text{ if } i \text{ is even and } > 1 \\ &\geq \frac{\alpha^i - \alpha}{\alpha^2 - 1} \text{ if } i \text{ is odd and } > 1 \end{aligned}$$

Using these lemmas and concepts, we can prove the following result.

**Theorem 2** *For any 1-lookahead online algorithm  $\mathcal{A}$  that allocates a non-zero usage in the first timeslot, the adversary can achieve a ratio of  $\frac{8}{5} - \epsilon$ , for any  $\epsilon > 0$ .*

Table 3: Capacity sequence as generated by adversary strategy and online algorithm  $\mathcal{A}$  usage: There are  $x_1$   $a_1$ s,  $x_2$   $a_2$ s, etc.

$$\begin{array}{cccccccccccccccc}
c(t): & 1/\alpha_1 & 1 & \dots & 1 & 1 & \alpha_1 & \dots & \alpha_1^{y_1-1} & \alpha_1^{y_1} & \dots & \beta_1 & \beta_1 & \beta_1\alpha_2 & \dots & \longrightarrow \\
a(t): & a_1 & \dots & a_1 & 0 & 0 & \dots & 0 & a_2 & \dots & a_2 & 0 & 0 & \dots & 0 & \longrightarrow \\
\longrightarrow & \beta_1\alpha_2^{y_2-2} & \beta_1\alpha_2^{y_2-1} & \beta_2 & \dots & \dots & \beta_{k-1} & \beta_{k-1} & \beta_{k-1}\alpha_k & \dots & \beta_{k-1}\alpha_k^{y_k-1} & \beta_k & x \\
\longrightarrow & a_3 & \dots & a_k & 0 & 0 & \dots & 0 & 0 & a_{k+1} & y & z
\end{array}$$

**Proof:** Let us assume that the adversary starts with the same strategy as presented above. In that case, say after  $n$  slots, the capacity and algorithm usage is as shown in Table 3.

We define a block  $B_i$  to be  $x_i$  consecutive slots used by algorithm followed by  $y_i$  0s.

We further define:

$$\alpha_i \stackrel{\text{def}}{=} \text{Constant used by adversary in the } i\text{-th block} \quad (1)$$

$$\beta_i \stackrel{\text{def}}{=} \prod_{j=1}^i \alpha_j^{y_j} \quad [\text{For convenience, we can define } \beta_0 = 1.] \quad (2)$$

In the capacity array, term 1, i.e.,  $\beta_0$  appears  $x_1 + 1$  times. Term  $\beta_1$  appears for  $x_2 + 1$  times. Similarly, term  $\beta_i$  appears for  $x_{i+1} + 1$  times. By definition,  $x_i \geq 1$  and  $y_i \geq 1$ . Due to capacity constraints,  $a_1 \leq 1/\alpha_1$ ,  $a_2 \leq \alpha_1^{y_1-1}$  and  $a_3 \leq \beta_1\alpha_2^{y_2-1}$ . In general,  $\forall i \geq 1, a_{i+1} \leq \beta_{i-1}\alpha_i^{y_i-1} = \beta_i/\alpha_i$ .

Each 0 in the algorithm allocation array makes the entry in the capacity array increase by a factor of  $\alpha_i$ , for some  $i$ .

The algorithm  $\mathcal{A}$  and optimal achieve a total of:

$$|A| = \sum_{i=1}^k a_i x_i + \text{opt}[a_{k+1}, y, z] \quad (3)$$

$$\begin{aligned}
\text{OPT} &= \sum_{i=1}^k \{\beta_{i-1} x_i + \beta_{i-1} h(\alpha_i, y_i)\} + \text{opt}[\beta_k/\alpha_k, \beta_k, x] \\
&\geq \sum_{i=1}^k \{\alpha_{i-1} a_i x_i + \beta_{i-1} h(\alpha_i, y_i)\} + \text{opt}[\beta_k/\alpha_k, \beta_k, x]
\end{aligned} \quad (4)$$

Using results presented above, we observe that if for any  $i$ ,  $x_i \geq \lceil \frac{5}{3\alpha_i-5} \rceil$ , then the adversary could terminate at that point. Also, if  $y_i \geq \lceil -\log_\alpha(3\epsilon) \rceil$ , the adversary could terminate. Thus, we assume that for all  $i \in \{1..k\}$ ,  $x_i < \lceil \frac{5}{3\alpha_i-5} \rceil$  and  $y_i < \lceil -\log_\alpha(3\epsilon) \rceil$ . This also allows us to assume that the adversary can force the algorithm to have as many blocks as it wants, since each block size is bounded.

Consider adversary's choice of  $\alpha_i = 2$  for  $i \in \{1, \dots, k-1\}$  and  $\alpha_k = 5/3 + \gamma_1$ .

Suppose  $S = \sum_{i=1}^k a_i x_i$ . Then, we have that:

$$\begin{aligned} |A| &= S + 2a_{k+1} \\ \text{OPT} &\geq 2S + \beta_{k-1}h(\alpha_k, y_k) + 3(a_{k+1} - \gamma_2) \\ \text{OPT} &\geq 2S + \beta_{k-1}(\alpha_k^{y_k} - \alpha_k)/(\alpha_k^2 - 1) + 3(a_{k+1} - \gamma_2) \end{aligned}$$

Now consider  $a_{k+1}$ , it is the value chosen by the algorithm in the  $(k+1)$ -th block. It is clear from the equation, that the best competitive ratio is obtained by the algorithm when  $a_{k+1}$  is as large as possible. Thus, say  $a_{k+1} = \beta_{k-1}\alpha_k^{y_k-1}$ . We thus obtain that:

$$\begin{aligned} |A| &= S + 2\beta_{k-1}\alpha_k^{y_k-1} \\ \text{OPT} &\geq 2S + \beta_{k-1}(\alpha_k^{y_k} - \alpha_k)/(\alpha_k^2 - 1) + 3\beta_{k-1}\alpha_k^{y_k-1} - 3\gamma_2. \end{aligned}$$

Three cases arise on value of  $y_k$ :

- Case I:  $y_k = 1$

$$\begin{aligned} |A| &= S + 2\beta_{k-1} \\ \text{OPT} &\geq 2S + 3\beta_{k-1} - 3\gamma_2. \\ \Rightarrow \text{OPT} &\geq 8/5 |A| + 0.2(2S - \beta_{k-1}) - 3\gamma_2. \end{aligned}$$

Knowing that  $2S \geq \beta_{k-1}$ , we obtain that  $\text{OPT} \geq 8/5 |A| - \epsilon$ , where  $\epsilon = 3\gamma_2$ .

- Case II:  $y_k = 2$

$$\begin{aligned} |A| &= S + 10/3\beta_{k-1} \\ \text{OPT} &\geq 2S + 6\beta_{k-1} - 3\gamma_2. \\ \text{OPT} &\geq 9/5 |A| - 3\gamma_2. \end{aligned}$$

- Case III:  $y_k > 2$

$$\begin{aligned} |A| &= S + 2\beta_{k-1}\alpha_k^{y_k-1} \\ \text{OPT} &\geq 2S + \beta_{k-1}\alpha_k^{y_k} - \alpha_k/25 - \gamma_4 + 3\beta_{k-1}\alpha_k^{y_k-1} - 3\gamma_2. \\ \text{OPT} &\geq 2S + 18/5\beta_{k-1}\alpha_k^{y_k-1} - 3/5\beta_{k-1} - 3\gamma_2. \\ \text{OPT} &\geq 9/5 |A| - 3\gamma_2 \end{aligned}$$

Taking the minimum value of competitive ratio from the three cases, we obtain that:

$$\text{OPT} \geq 8/5 |A| - \epsilon$$

■

Theorem 2 can be easily extended to all online algorithms by combining results of the Lemma 3.

**Theorem 3** *For any online algorithm  $\mathcal{A}$ , the adversary can achieve a ratio of  $\frac{8}{5} - \epsilon$ , for any  $\epsilon > 0$ .*

**Proof:** If  $\mathcal{A}$  does not use any of the first  $\lceil -\log_\alpha(3\epsilon) \rceil$  time slots, then by Lemma 3, the adversary can achieve a ratio of  $5/3 - \epsilon$  and stop. If the algorithm does use a timeslot, then the adversary can simply ignore all prior timeslots, and then using Theorem 2, the adversary can achieve a ratio of  $\frac{8}{5} - \epsilon$ , for any  $\epsilon > 0$ . ■

### 3 Wait-Dominate-Hold Algorithm

In this section, we analyze the Wait-Dominate-Hold online algorithm (WDH) as presented in [15].

We use the following two definitions as part of our analysis.

**Strongly increasing sequence:** A sequence of numbers is said to be *strongly* increasing if each number is at least twice as much as the previous number.

**Almost equal:** Two numbers are said to be *almost equal* if neither is more than or equal to twice the other. We represent it as  $x \approx y$ .

#### 3.1 Background and definition of WDH

The WDH algorithm is a one step lookahead online algorithm. Before allocating usage for a time slot, it uses the following information: allocation in the previous timeslot  $x(t-1)$ , current capacity  $c(t)$  and next capacity  $c(t+1)$ .

WDH algorithm uses the following set of rules for choosing  $x(t)$ :

(Pre-calculation) If  $x(t-1) = 0$ , set  $u(t) = c(t)$ , else  $u(t) = \min\{x(t-1), c(t)\}$ .

( $\mathcal{R}1$ ) If  $c(t+1) > 2u(t)$ , set  $x(t) = 0$ .

( $\mathcal{R}2$ ) If  $u(t) \geq 2c(t+1)$ , set  $x(t) = u(t)$  (note that  $x(t+1)$  will be assigned to 0 in the next slot schedule).

( $\mathcal{R}3$ ) If  $c(t+1)/2 \leq u(t) < 2c(t+1)$  and  $x(t-1) > 0$ , set  $x(t) = u(t)$ .

( $\mathcal{R}4$ ) If  $c(t+1)/2 \leq u(t) < 2c(t+1)$  and  $x(t-1) = 0$ , set  $x(t) = \min\{u(t), c(t+1)\}$ .

Note that ( $\mathcal{R}1$ ) corresponds to *Wait*, ( $\mathcal{R}2$ ) corresponds to *Dominate*, and ( $\mathcal{R}3$ ) and ( $\mathcal{R}4$ ) correspond to *Hold*.

Table 4: Wait-Dominate-Hold: A 1-Slot Lookahead Online Algorithm

```

double waitDominateHold (double prevUsage, double currCapacity, double nextCapacity)
{
    // Calculates the upper bound for usage in this cycle
    double ub = 0;
    if (prevUsage > currCapacity)
        return 0;
    else if (prevUsage  $\stackrel{?}{=} 0$ )
        ub = currCapacity;
    else
        ub = prevUsage;

    // Waits (next capacity is very high)
    if (2 * ub < nextCapacity)
        return 0;

    // Dominates (next capacity is very low)
    if (ub  $\geq$  2 * nextCapacity)
        return ub;

    // If previous slot was used, it is not allowed to Hold.
    if (prevUsage > 0)
        return ub;

    // Hold and wait
    return min (ub, nextCapacity);
}

```

The precise definition of the algorithm is presented in Table 4.

An example usage of the Wait-Dominate-Hold algorithm is shown in Table 5.

Table 5: Example usage of Wait-Dominate-Hold algorithm.

<i>C</i> :	23	25	4	7	16	33	66	9	8	7	6	1
<i>X</i> :	23	23	0	0	0	33	33	0	7	7	0	1

## 3.2 Competitive Analysis of WDH Algorithm

In [15], it was proven that Wait-Dominate-Hold algorithm is 4-competitive. The analysis was against the total capacity, not against the optimal offline solution.

We present an improved analysis. This novel analysis technique tracks the internal states of the WDH algorithm.

### 3.2.1 Definition of Return States

By analyzing Table 4, we can associate each “return” statement with a corresponding return state. Following states can be identified:

- *S*: Suffer: Allocation of previous timeslot was more than capacity of current timeslot

- *W*: Wait: Capacity of next timeslot is more than twice the maximum possible usage in current timeslot
- *D*: Dominate: Capacity of next timeslot is less than half the maximum possible usage in current timeslot
- *H*: Hold: Capacity of next timeslot is approximately (within a factor of two) of current maximum possible usage
- *C*: Continue: Would like to dominate/hold, but have to maintain allocation of previous timeslot

### 3.2.2 Rules governing the internal states of WDH algorithm

Rules:

- *C* can only be followed by *S*.
- *D* can only be followed by *S*.
- *H* can be followed by *W, D, H* or *C*, but not by *S*.
- *W* can be followed by *W, D* or *H*, but not by *C* or *S*.
- *S* can be followed by *W, D* or *H*, but not by *C* or *S*.

The automata representing the state transitions of WDH algorithm is shown in Figure 1.

### 3.2.3 Defining the sentences

From the automata shown in Figure 1, it is clear that the sentences that can be generated using WDH are as follows:

1.  $W^*DS$
2.  $W^*H^+CS$
3.  $W^*H^+DS$
4.  $W^*H^+$

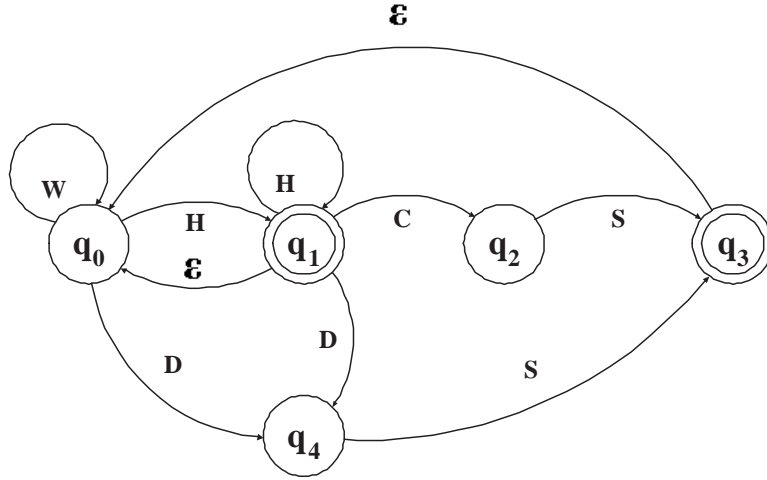


Figure 1: Automata

Thus, possible sentences are:  $\{W^*DS|W^*H^+CS|W^*H^+DS|W^*H^+\}^*$ .

We observe an interesting element from these sentences that when  $H$  is followed by  $W$ , it could be either because the capacity for that timeslot was lesser than half of the next capacity, or because the possible allocation was lesser than half of the next capacity. We further observe that any sentence that follows  $W^*H^+$  sentence must have at least one  $W$ . To account for that, we consider the possible sentence  $W^*H^+$  to include a  $W$  at the end, that is,  $W^*H^+W$ . As we will observe in Section 3.2.8, we analyze this sentence in two separate cases and can then decide if the last  $W$  in that sentence should be analyzed as part of that sentence or a different sentence.

### 3.2.4 Useful Lemma

We use the following lemma in some of the proofs. This lemma is similar to the definition of  $h(\alpha, i)$  given in Section 2 in the lower bound analysis, using a constant value of 2 for  $\alpha$ .

**Lemma 4** *Given a strongly increasing sequence of capacities, the maximum allocation that can be achieved is no more than  $4/3$  times the capacity of the last timeslot.*

**Proof:**

**Given:**  $\mathbf{C} : c(1), c(2), \dots, c(k)$ , such that:

$$2c(i) \leq c(i+1) \quad \forall i \in 1 \dots k-1.$$

**To Prove:** Optimal allocation,  $opt(\mathbf{C}) \leq 4/3c(k)$

Case I:  $k$  is even.

$$\begin{aligned}
opt(c(1), c(2), \dots, c(k)) &\leq opt(c(1), c(2)) + \dots + opt(c(k), c(k-1)) \\
&\leq c(2) + \dots + c(k) \\
&\leq 4/3 c(k)
\end{aligned}$$

Case II:  $k$  is odd.

$$\begin{aligned}
opt(c(1), c(2), \dots, c(k)) &\leq c(1) + opt(c(2), c(3)) + \dots + opt(c(k), c(k-1)) \\
&\leq c(1) + c(3) + \dots + c(k) \\
&\leq 4/3 c(k)
\end{aligned}$$

■

Next, we analyze the 4 different sentences independently.

### 3.2.5 Analysis of $W^*DS$ sentence

#### Input

$$C : c(1), \dots, c(k), c(k+1), c(k+2)$$

$$2c(i) \leq c(i+1) \forall i \in \{1..k\}$$

$$2c(k+2) \leq c(k+1)$$

#### WDH Output

$$X : 0, 0, \dots, 0, c(k+1), 0$$

#### Optimal Solution

$$c(1), 0, c(3), 0, c(5), \dots, 0, c(k+1), 0 \text{ [If } k \text{ is even]}$$

$$0, c(2), 0, c(4), 0, c(6), \dots, 0, c(k+1), 0 \text{ [If } k \text{ is odd]}$$

$$\begin{aligned}
|OPT| &= c(k+1) + c(k-1) + c(k-3) + \dots \\
&\leq c(k+1) + 1/4c(k+1) + 1/16c(k+1) + \dots \\
&\leq 4/3c(k+1)
\end{aligned}$$

#### Result

Competitive Ratio:  $4/3$

### 3.2.6 Analysis of $W^*H^+CS$ sentence

#### Input

$$C : c(1), \dots, c(k), c(k+1), c(k+2), \dots, c(k+n+1), c(k+n+2)$$

$$2c(i) \leq c(i+1) \forall i \in \{1..k\}$$

$$c(k+i) \approx c(k+j) \forall i, j \in \{1..n+1\}$$

$$c(k+i) > c(k+n+2) \forall i \in \{1..n+1\}$$

### WDH Output

$$X : 0, 0, \dots, 0, \underbrace{y, y, \dots, y}_{n+1}, 0$$

$$\text{Note that } y = \min\{c(k+1), c(k+2), \dots, c(k+n+1)\}$$

### Optimal Solution

$$\text{Suppose that } z = \max\{c(k+1), c(k+2), \dots, c(k+n+1)\}$$

$$\text{Note that } y > c(k+n+2).$$

$$\begin{aligned} |OPT| &\leq 4/3c(k) + \max\{nz, (n+1)y\} + c(k+n+2) \\ &\leq 4/3y + \max\{nz, (n+1)y\} + y \\ &\leq (2n+1+4/3)y \end{aligned}$$

$$|OPT|/|X| \leq (2n+1+4/3)/(n+1) \leq 13/6$$

### Result

Competitive Ratio: 13/6

## 3.2.7 Analysis of $W^*H^+DS$ sentence

### Input

$$C : c(1), \dots, c(k), c(k+1), c(k+2), \dots, c(k+n+1), c(k+n+2)$$

$$2c(i) \leq c(i+1) \forall i \in \{1..k\}$$

$$c(k+i) \approx c(k+j) \forall i, j \in \{1..n+1\}$$

$$c(k+i) > 2c(k+n+2) \forall i \in \{1 \dots n+1\}$$

### WDH Output

$$X : 0, 0, \dots, 0, \underbrace{y, y, \dots, y}_{n+1}, 0$$

$$\text{Note that } y = \min\{c(k+1), c(k+2), \dots, c(k+n+1)\}$$

### Optimal Solution

$$\text{Suppose that } z = \max\{c(k+1), c(k+2), \dots, c(k+n+1)\}$$

$$\text{Note that } y > 2c(k+n+2).$$

$$\begin{aligned} |OPT| &\leq 4/3c(k) + \max\{nz, (n+1)y\} + c(k+n+2) \\ &\leq 4/3y + \max\{nz, (n+1)y\} + 1/2y \\ &\leq (2n+1/2+4/3)y \end{aligned}$$

$$|OPT|/|X| \leq (2n+1/2+4/3)/(n+1) \leq 23/12$$

### Result

Competitive Ratio: 23/12

### 3.2.8 Analysis of $W^*H^+W$ sentence

#### Input

$$C : c(1), \dots, c(k), c(k+1), c(k+2), \dots, c(k+n), c(k+n+1)$$

$$2c(i) \leq c(i+1) \quad \forall i \in \{1 \dots k\}$$

$$c(k+i) \approx c(k+j) \quad \forall i, j \in \{1 \dots n\}$$

$$2c(k+i) \geq c(k+n+1) \quad \forall i \in \{1 \dots n\}$$

#### WDH Output

$$X : 0, 0, \dots, 0, \underbrace{y, y, \dots, y}_n, 0$$

$$\text{Note that } y = \min\{c(k+1), c(k+2), \dots, c(k+n)\}$$

#### Optimal Solution

$$\text{Suppose that } z = \max\{c(k+1), c(k+2), \dots, c(k+n)\}$$

$$\text{Note that } 2y \geq c(k+n+1).$$

- **Case 1:**  $n = 1$

We consider two cases on the values of  $c(k+1)$  and  $c(k+2)$ .

- $y = c(k+1) < c(k+2)$

Note that  $y > 2c(k)$ .

$$\begin{aligned} |OPT| &\leq 4/3c(k) + 2y \\ &\leq 2/3y + 2y \\ &\leq 8/3y \end{aligned}$$

- $y = c(k+2) \leq c(k+1)$

In this case, note that WDH output is  $X : 0, 0, \dots, 0, y, 0$  where  $y = c(k+2)$ , such that  $x_{k+2} = 0$  because of  $2c(k+2) < c(k+3)$ , where  $c(k+3)$  is located in the next sentence.

If we take off  $c(k+2)$  from and the current sentence, and append it in front of the next sentence, it will not affect the analysis of the next sentence, since newly generated  $W^*$  in the next sentence is still strongly increasing. Now we analyze the current sentence without  $c(k+2)$ .

$$\begin{aligned} |OPT| &\leq 4/3c(k) + y \\ &\leq 4/3y + y \\ &\leq 7/3y \end{aligned}$$

- **Case 2:**  $n > 1$

$$\begin{aligned} |OPT| &\leq 4/3c(k) + \max\{nz, (n+1)y\} \\ &\leq 4/3y + \max\{nz, (n+1)y\} \\ &\leq (2n + 4/3)y \end{aligned}$$

$$|OPT|/|X| \leq (2n + 4/3)/n \leq 8/3 \text{ when } n > 1.$$

**Result**

Competitive Ratio: 8/3

**3.2.9 Putting it all together**

**Theorem 4** *Wait-Dominate-Hold algorithm is 8/3-competitive.*

**Proof:** Immediate from results of preceding sections, as we take the maximum value of competitive ratios of all 4 sentences. ■

**3.3 Tightness of Competitive Analysis of WDH Algorithm**

Next, we present an example where optimal allocation is twice the allocation of WDH algorithm.

$C$	1	2	2	2	...	2
WDH	1	1	1	1	...	1
OPT	0	2	2	2	...	2

In this case, the optimal allocation approaches twice the allocation of WDH algorithm, and can be made arbitrarily close to 2 by changing the length of the input.

**4 Problem Variation:  $k$ -lookahead Algorithms**

Having proved both upper and lower bounds for the 1-lookahead online problem, we now focus on the  $k$ -lookahead variation. As outlined in [18], considering  $k$ -lookahead is a natural extension of online problems, both in the theoretical, and in the practical sense (the authors in that paper pose quite eloquently “What is it worth to know a part of the future”). Online scheduling problem with  $k$ -lookahead was also considered in [19] in the context of web caching. In [18], two models of lookahead for online algorithms were presented, weak  $k$ -lookahead (in which next  $k$  requests are known) and strong  $k$ -lookahead (in which  $k$  *distinct* requests are known). In this paper, we focus on the weak  $k$ -lookahead, as distinctness of capacity values does not fundamentally change our problem definition.

We consider online algorithms that have  $k$ -timeslots lookahead information available, where  $k > 1$ . In such a case, the problem can be stated as:

**Given:**  $x(t - 1), c(t), c(t + 1), \dots c(t + k)$

**To find:**  $x(t)$

We refer to  $x(t-1)$  as the previous allocation,  $c(t)$  as the current capacity and  $c(t+1) \dots c(t+k)$  as the lookahead capacities. We observe that if  $k = 0$ , then no competitive ratio is possible, and that if  $k = 1$ , then the results presented in Sections 2 and 3 are the best known results.

#### 4.1 A lower bound on all $k$ -lookahead algorithms

**Theorem 5** *No  $k$ -lookahead online algorithm can be better than  $\left(\frac{k+2}{k+1}\right)$ -competitive.*

**Proof:** Let us assume that there are a total of  $k + 2$  timeslots, and the adversary sets  $c(i) = 1 \forall i \in \{1..k+1\}$ .

Suppose the online algorithm  $\mathcal{A}$  allocates value of  $a$  in the first timeslot. Let us denote the maximum allocation achieved by  $\mathcal{A}$  by  $|A|$ . We consider two cases on the value of  $a$ :

- **Case I:**  $a \geq \frac{k}{k+1}$  In this case, adversary can set  $c(k+2) = a - \epsilon$ .

In this case algorithm  $\mathcal{A}$  must miss at least one timeslot, as its current allocation exceeds the capacity available in the last timeslot.

$$|A| \leq a(k+1), \text{ while } |OPT| = (k+2)(a - \epsilon)$$

$$\text{Thus } \frac{|OPT|}{|A|} \geq \frac{k+2}{k+1}.$$

- **Case II:**  $a < \frac{k}{k+1}$  In this case, adversary can set  $c(k+2) = 1$ .

Algorithm  $\mathcal{A}$  can continue to use  $a$  for all timeslots in which case it achieves a total allocation of  $a(k+2)$  or it can miss one timeslot and allocate value of 1 for all other timeslots. In either case,  $|A| \leq k+1$ , while  $|OPT| = k+2$

$$\text{Thus } \frac{|OPT|}{|A|} \geq \frac{k+2}{k+1}.$$

■

#### 4.2 Two families of $k$ -lookahead online algorithms

In this section, we present two families of  $k$ -lookahead algorithms for all online problems for which an optimal offline algorithm is known.

##### 4.2.1 OPTB: Optimal Block Algorithm

Given the current state of the algorithm, and the  $k$ -lookahead information, the OPTB algorithm proceeds as follows.



### 4.2.3 Applying Optimal Block Algorithm to our problem

Next, we consider the natural application of OPTB to our problem, where we use the dynamic programming offline optimal algorithm given in [15].

1. As per OPTB, we divide the input into  $m$  blocks of size  $k$  each. Say the block  $B_i$  consists of timeslots  $ik + 1, ik + 2, \dots, ik + k$ . Blocks are numbered as  $B_0, B_1, \dots, B_{m-1}$ .
2. We use the known optimal offline algorithm to find the optimal schedule for the block  $B_i$ .
3. Set allocation for timeslots  $ik + 1, ik + 2, \dots, ik + k - 1$  as per the optimal schedule.
4. Before setting the allocation for  $ik + k$ , calculate optimal schedule for block  $B_{i+1}$ . We observe that as the algorithm is  $k$ -lookahead, the resource capacities for block  $B_{i+1}$  are available before a decision needs to be made for  $ik + k$  timeslot.
5. For timeslots  $ik + k$  and  $ik + k + 1$ , if the values  $x(ik + k)$  and  $x(ik + k + 1)$  are equal, set both of them, otherwise set the smaller of them to 0.

**Theorem 6** *Algorithm OPTB is 2-competitive, for all  $k \geq 2$ .*

**Proof:** Clearly, the sum of optimal offline solutions constructed for each block is at least as large as the optimal offline.

$$\sum_i B_i \geq OPT$$

We observe that actual allocation made is different from the sum, as the smaller of the two elements:  $x(ik + k)$  and  $x(ik + k + 1)$  is set to 0, unless the two elements are equal.

$$\Rightarrow \sum_i B'_i \geq \sum_i B_i - \sum_i \min\{x_{ik}, x_{ik+1}\} \quad (5)$$

The sum of solutions for each block contains both these elements, among others.

$$\Rightarrow \sum_i B_i \geq \sum_i \{x_{ik} + x_{ik+1}\} \quad (6)$$

From Equations 5 and 6, we can deduce that:

$$\begin{aligned} \Rightarrow \sum_i B'_i &\geq B_i/2 \\ \Rightarrow \sum_i B'_i &\geq OPT/2 \end{aligned}$$

■

**Note on Time Complexity:** Since the time complexity of the dynamic programming algorithm is  $O(k^3)$ , the time complexity of the OPTB algorithm is  $O(k^2)$  per timeslot. As  $k$  increases, this may increase while providing very little additional increase in performance. In that case, a constant value of  $k$  (such as 10) should be adopted as per the simulation results and practical requirements.

## 5 Conclusions

We consider a fundamental resource allocation problem that has a constraint that a change in allocation results in the changeover penalty of one timeslot. The problem is directly motivated from scheduling of capacity in wireless networks.

After observing that an algorithm with no lookahead cannot be  $c$ -competitive for any  $c$ , we focussed on 1-lookahead algorithms. We proved that no 1-lookahead algorithm can be better than  $8/5$ -competitive. The proof uses an adaptive offline adversary, and the central idea is that adversary increases the capacity if the online algorithm does not use a timeslot, and keeps it constant if the adversary uses a timeslot. We proved that no 1-lookahead algorithm can be better than  $8/5$ -competitive.

We have presented improved analysis of Wait Dominate Hold (WDH) algorithm that was previously known to be 4-competitive. We proved that WDH is  $8/3$ -competitive. We use a novel technique of tracking internal states of the algorithm in terms of a finite state automata, and analyzed the different sentences that can be possibly generated by the return states of the WDH algorithm. We hope that the same technique can also be used to analyze other algorithms.

We have also considered  $k$ -lookahead algorithms for the resource allocation problem, and have proven lower bound of  $(k+2)/(k+1)$  using the adversary approach. As expected, this lower bound approaches 1 as  $k$  increases.

We have presented a simple online algorithm that is 2-competitive when there is at least 2 lookahead available. Before giving that algorithm, we present two general families of online  $k$ -lookahead algorithms - the Optimal Block Algorithm (OPTB) and Optimal Sliding Window Algorithm (OPTSW). We believe that OPTB and OPTSW can be considered as general frameworks for many online problems that use the concept of “timeslot” or “step” and where lookahead is available.

Future work in this field can focus on (i) further improving the analysis of WDH algorithm, (ii) finding other algorithms for 1-lookahead problem that have better competitive ratio than WDH,

and (iii) finding other algorithms for  $k$ -lookahead problem that give a better competitive ratio for higher values of  $k$ . Future work may also focus on extending the return state analysis technique and on further developing the two families of algorithms for  $k$ -lookahead problems.

## References

- [1] R. L. Graham, “Bounds for certain multiprocessor anomalies,” *Bell Systems Technical Journal*, vol. 45, pp. 1563–1581, November 1966.
- [2] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, “New algorithms for an ancient scheduling problem,” *Journal of Computer Systems Science*, vol. 51, no. 3, pp. 359–366, 1995.
- [3] D. R. Karger, Steven J. Phillips, and Eric Torng, “A better algorithm for an ancient scheduling problem,” in *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 1994, pp. 132–140, Society for Industrial and Applied Mathematics.
- [4] S. Albers, “Better bounds for online scheduling,” *SIAM Journal on Computing*, vol. 29, no. 2, pp. 459–473, 1999.
- [5] P. Brucker, *Scheduling Algorithms*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
- [6] A. Feldmann, M. Y. Kao, J. Sgall, and S. H. Teng, “Optimal online scheduling of parallel jobs with dependencies,” in *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, New York, NY, USA, 1993, pp. 642–651, ACM Press.
- [7] D. B. Shmoys, Joel Wein, and David P. Williamson, “Scheduling parallel machines on-line,” in *Proceedings of the 32nd annual symposium on Foundations of computer science*, 1991, pp. 131–140.
- [8] L. A. Hall and D. B. Shmoys, “Approximation schemes for constrained scheduling problems,” in *Proceedings of the 30th Ann. IEEE Symp. on Foundations of Computer Science*, April 1989, pp. 134–139.
- [9] G. Rote and G. J. Woeginger, “Minimizing the number of tardy jobs,” *Acta Cybernetica*, vol. 13, no. 4, pp. 423–430, 1998.
- [10] Matthew Andrews and Lisa Zhang, “Scheduling over a time-varying user-dependent channel with applications to high-speed wireless data,” *J. ACM*, vol. 52, no. 5, pp. 809–834, 2005.
- [11] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai, “Scheduling broadcasts in wireless networks,” in *ESA '00: Proceedings of the 8th Annual European Symposium on Algorithms*, London, UK, 2000, pp. 290–301, Springer-Verlag.
- [12] H. Luo, S. Lu, and V. Bharghavan, “A new model for packet scheduling in multihop wireless networks,” in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, NY, USA, 2000, pp. 76–86, ACM Press.

- [13] S. Lu, V. Bharghavan, and R. Srikant, “Fair scheduling in wireless packet networks,” in *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, New York, NY, USA, 1997, pp. 63–74, ACM Press.
- [14] A. Khattab and K. Elsayed, “Channel-quality dependent earliest deadline due fair scheduling schemes for wireless multimedia networks,” in *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, New York, NY, USA, 2004, pp. 31–38, ACM Press.
- [15] A. Arora and H. A. Choi, “Channel aware scheduling in wireless networks,” Tech. Rep. 002, The George Washington University, 2006.
- [16] G. Sahin, F. Jin, A. Arora, and H.-A. Choi, “Predictive scheduling in multi-carrier wireless networks with link adaptation,” in *Proc. IEEE Vehicular Technology Conference*, September 2004.
- [17] F. Jin, G. Sahin, A. Arora, and H.-A. Choi, “The effects of the sub-carrier grouping on multi-carrier channel-aware scheduling,” in *Proceedings of BroadNets 2004, Broadband Wireless Networking Symposium*, October 2004.
- [18] S. Albers, “A competitive analysis of the list update problem with lookahead,” *Theoretical Computer Science*, vol. 197, no. 1–2, pp. 95–109, 1998.
- [19] T. Feder, R. Motwani, R. Panigrahy, and A. Zhu, “Web caching with request reordering,” in *13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 104–105.