General Perfectly Periodic Scheduling

Extended Abstract

Zvika Brakerski zvika@eng.tau.ac.il
Aviv Nisgav aviv@eng.tau.ac.il
Boaz Patt-Shamir boaz@eng.tau.ac.il
Dept. of Electrical Engineering
Tel Aviv University
Tel Aviv 69978
Israel

Abstract

In a perfectly-periodic schedule, time is divided into time-slots, and each client is scheduled precisely every some predefined number of slots, called the period of that client. Periodic schedules are useful in wireless communication and other settings. The quality of a schedule is measured by the proportion between the requested and the granted periods: either the maximum over all jobs, or the average. There exist good scheduling algorithms for the average measure in the unit-length single-server model in which all jobs are one slot long, and at most one job is served in each time unit. In this paper we study the general model, where each job may have a different length, and m jobs can be served in parallel for some given m. We give a lower bound for this model which demonstrates the inherent difficulty of multiple lengths, and present a sequence of algorithms, culminating in an algorithm for the general case which is asymptotically optimal under the maximum ratio measure (and hence also the average ratio measure). The new algorithms utilize new techniques which are rather different from the known algorithms used for the unit-length model. Some of the algorithms improve on the best known bounds for the unit-length model.

1 Introduction

Consider a system that comprises a resource and some clients sharing it by means of time multiplexing: in any given time, a different client may use the resource. Many application domains (e.g., real-time tasks, multimedia applications, communication with guaranteed quality-of-service) require that clients are served at some prescribed rate, and this rate should be as smooth as possible even in small time-windows. Practically, this means that the time axis is divided into small equal-size quanta called time slots, which are allocated to clients in more-or-less equally spaced intervals. The allocation of time slots to clients is governed by a scheduling algorithm. More precisely, the scheduling algorithm is given a set of *requested shares*, and its goal is to produce an assignment of time slots to clients, while trying to optimize two different measures:

- **Approximation:** a schedule is said to have good approximation if the fraction of time-slots allocated to each client (called *granted share* below) is close to the requested share of the client, according to some given metric.
- **Smoothness:** a schedule is said to have good smoothness if the time-slots allocated to each client are as evenly spaced as possible (under some other given metric).

The metrics may vary, according to the application at hand; in any case, the intuition is that the best possible approximation is when the granted rates are exactly the requested rates, and the best possible smoothness is when each client is scheduled exactly every p time slots, for some p called the *period* of that client. Note that it is easy to optimize one objective while neglecting the other: approximation can be trivially achieved to any desired degree by taking long sequences of time slots and partitioning them to intervals whose lengths are proportional to the requested shares, with some rounding. The longer the sequence is, the better approximation can be guaranteed; but clearly, the longer the sequence is, the worst smoothness we get. On the other end of the spectrum we have the round-robin schedule, where each client gets one time slot in turn, regardless of its requested share. This scheme features the best possible smoothness, but suffers from poor approximation in general. Most prior work on this scheduling problem concentrated on the variant where the chief goal is to obtain good approximation, while smoothness was only of secondary importance. In this paper, we are interested in exploring the other extreme: *we insist on maintaining strict smoothness while relaxing the approximation requirement*. More precisely, we call a schedule

0

perfectly periodic, if each client i is scheduled *exactly* every p_i time slots, for some p_i called the period of i. In our setting, the granted period may be different from the requested period. Our goal is to optimize the approximation measure under the perfect periodicity constraint.

Perfect schedules are attractive from a few viewpoints, all due to the fact that mathematically, they are very simple to describe: the schedule of a client is completely specified by two numbers (period and offset). This inherent simplicity gives rise to several pleasing consequences; let us list a few.

Wireless communication with portable devices. One of the major power consumers in portable devices (such as PDAs) is their radio, used for wireless communication. This is a critical issue, since a portable device may weigh only few grams, leaving very little room for batteries. Perfect schedules help to significantly reduce the power requirement of a mobile client while it is waiting for its turn: instead of "busy waiting" (constantly listening to the radio channel), the device can actually shut off its radio until its turn arrives. This feature exists in modern wireless technologies [11]. For example, in Bluetooth, which is a new technology for wireless communication of small devices, the standard defines *sniff mode* [1]. A device in sniff mode is obliged to listen to the network only in time windows defined in a strictly periodic fashion. Another example is the concept of broadcast disks [2], where the server continuously broadcasts a "database." A client that wishes to access a certain "page" in the database waits until that item is scheduled. If the schedule is perfectly periodic, it is extremely easy for the client to compute when will be the next occurrence of its desired item. Moreover, if the schedule is perfectly periodic, then it is known [12, 14] how to interleave "index pages" between the data pages so that a randomly arriving client does not need to continuously listen until its desired data page is broadcasted. The index pages reduce significantly the active listening time of the client. There is no such scheme for non-perfect schedules.

Fairness. Another important motivation for perfect periodicity is that in time-sharing systems, one of the main objectives of schedules is that they should be fair: intuitively, fairness means that the number of time slots client i waits should always be inversely proportional to its share. A social example for this requirement is the classical *chairperson assignment problem* [16], that can be illustrated with the following example. A union of several states changes its chairperson every year. The schedule should be fair: Each state gets its share of chairing the union according to its size, say. However, the schedule should also attain this fairness quickly: no state would agree to wait hundreds of years to get its first term of chairing the union. What constitutes a good solution? Several fairness criteria were suggested. For example, in some networks models, each client i has two parameters (w_i, r_i), and the requirement is that in any time window of length $T \geq w_i$, client i gets at least $\lfloor r_i T \rfloor$ time slots [9]. A stricter requirement is the *prefix criterion*, where the requirement is that in any prefix of T slots, each client i gets either $\lfloor \alpha_i T \rfloor$ or $\lceil \alpha_i T \rceil$ slots, where α_i is the share allocated to client i [15]. Since the number of slots is integral, this seems to be the best possible. Indeed, there exists a schedule that meets the prefix fairness requirement [16]. Still, the *gap* between two occurrences of the same client could be as large as twice its average gap. When fairness is very important, perfect schedules provide the best solution.

What's known. Early work on perfectly periodic schedules was motivated by teletex systems [3]. Another variant is the maintenance problem [4, 18]. Minimizing the waiting time for broadcast disks is studied in [13, 5]. Non-perfect schedules are also studied in [15, 18, 6]. The main reason to study non-perfect schedules, apparently, is that perfect schedules are not always feasible: Consider, for example, the case where one client requests period 2 and another requests period 3. There is no way to satisfy both requests: the first client must occupy either all the even-numbered slots or all the odd-numbered slots, but the second client must occupy some even-numbered and some odd-numbered slots. It therefore follows that if perfect periodicity is sought, there are cases where the periods granted will not match the requests. Moreover, it is NP-hard even to decide whether a given set of requests admits a perfectly periodic schedule [5]. Fortunately, it turns out that perfect periodicity is not necessarily expensive in terms of approximation. Specifically, define for each client its approximation ratio to be the proportion between its requested period and its granted period. It is known [10] that if all jobs have unit size, then there exist schedules that guarantee that the *average* approximation ratio (where the weight of each client is its requested bandwidth) is close to optimal. These schedules use a hierarchical round-robin method, called *tree scheduling*. Tree scheduling was further investigated in [7, 8]. In [7], the *maximum* measure is also studied: under this measure, the quality of the schedule is the worst-case approximation ratio over all clients.

Our results and paper organization. In this paper, we extend the concept of perfectly periodic schedules in two ways. First, we consider the *multiple length* model, in which each client i, in addition to to its requested period τ_i, also has a length b_i, and the requirement is that the schedule must allocate b_i time slots for each occurrence of that client. The occurrences must be perfectly periodic as usual. Secondly, we consider the *multiple server* model, where we assume that in each time slot, m clients can be served in parallel, and the total requested bandwidth is m. We investigate these models under both the average and the maximum measures. We start by showing that the multiple length case is inherently different from the unit-length case: in contrast to the unit-length model, even if all lengths and periods are powers of 2, there may be no perfect schedule that satisfies the requests. It turns out that the the ratio between the largest job size and the shortest period is a key quantity. Formally, we define the *extent* of a given instance J to be $R_J \stackrel{\text{def}}{=} \frac{\max\{b_i|i\in J\}}{\min\{\tau_i|i\in J\}}$. Our lower bound shows that in general, the best possible average ratio (and hence, maximum ratio) is at least $1 + R_J - O(1/\min\{\tau_i \mid i \in J\})$. This lower bound is presented in Section 3. We then proceed to provide upper bounds on the approximation ratio, also expressed in terms of R_J. The algorithms are presented in a succession of refinements. In Section 4 we give an algorithm that guarantees a good upper bound for the MAX measure, provided that all periods are powers of 2 times a common multiple. Based on this algorithm, we analyze in Section 5 a simple algorithm that gets an approximation ratio of $\frac{9}{8} + \frac{3}{2}R_J + \frac{1}{2}R_J^2$ for the AVE measure. A generalization of the schedules produced by the algorithm of Section 4 is presented in Section 6. This generalization is used to achieve maximum ratio guarantee of $1 + O(R_J^{1/3})$, in the algorithm presented in Section 7. This algorithm easily generalizes to the multiple server case.

The formal model is presented in Section 2. Many proofs are omitted from this extended abstract.

2 Problem Statement and Notation

Instances. An instance of the perfectly-periodic scheduling problem is a set of n *jobs* $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$, where b_i is the *size* or *length* of j_i, and τ_i is the *requested period* of j_i. We sometimes refer to jobs also as *clients*. The maximal length of a job in an instance J is defined by $B_J \stackrel{\text{def}}{=} \max\{b_i \mid i \in J\}$. The maximal and minimal values of the requested periods in instance J are defined by $T_J \stackrel{\text{def}}{=} \max\{\tau_i \mid i \in J\}$, and $t_J \stackrel{\text{def}}{=} \min\{\tau_i \mid i \in J\}$. The ratio between B_J and t_J is called the *extent* of J, formally defined by $R_J \stackrel{\text{def}}{=} \frac{B_J}{t_J}$. For the single server model, we assume that $R_J \leq 1$ (otherwise, no schedule can satisfy the requests of J). We omit subscripts when they are clear from the context.

Schedules. A *schedule* S for an instance J is a set of infinite sequences of *start times* $S = \{I_1, \ldots, I_n\}$, such that $I_i = \langle A_{i_1}, A_{i_2}, A_{i_3}, \ldots\rangle$ where A_{i_k} is a nonnegative integer for each i, k. We say that job i is *scheduled* in *time slot* t if for some k, $A_{i_k} \leq t < A_{i_k} + b_i$. We assume that $A_{i_{k+1}} \geq A_{i_k} + b_i$ for all i, k. A schedule is m*-feasible* if for all t, at most m jobs are scheduled at t. The parameter m is called the *number of servers*. For the most part of this paper, we will be interested in the single server case, i.e., $m = 1$. A schedule S is said to be *perfectly periodic* (or just *periodic* for short) if for each job i there exists a *granted period* τ_i^S such that for all j, $A_{i_{j+1}} - A_{i_j} = \tau_i^S$. Note that the granted periods may be different from the requested periods, but the job lengths cannot be truncated by the schedule. Periodic schedules can be represented by their *cycles*, i.e, a sequence of time slots whose length is the least common multiple of job periods.

The *requested bandwidth* of job j_i is defined by $\beta_i \stackrel{\text{def}}{=} \frac{b_i}{\tau_i}$. The total bandwidth of an instance J is defined by $\beta_J \stackrel{\text{def}}{=} \sum_{i=1}^n \beta_i$. Note that if $\beta_J > m$ where m is the number of servers, there is no feasible schedule for the instance. The *free bandwidth* of an instance J is defined by $\Delta_J \stackrel{\text{def}}{=} m - \beta_J$.

Quality measures. Let J be an instance for the perfectly periodic scheduling problem and let S be a schedule for J. The *individual ratio* of a job i in S is defined by $\rho_i \stackrel{\text{def}}{=} \frac{\tau_i^S}{\tau_i}$. The MAX measure is simply the maximum over the individual ratios, defined formally by $C_{\text{MAX}}(J, S) \stackrel{\text{def}}{=} \max\{\rho_i \mid i \in J\}$. The AVE measure is the weighted average of the individual ratios, where the weight of i is its requested bandwidth.

$$C_{\text{AVE}}(J, S) \stackrel{\text{def}}{=} \frac{1}{\beta_J}\sum_{i=1}^n \beta_i\rho_i = \frac{1}{\beta_J}\sum_{i=1}^n b_i\frac{\tau_i^S}{\tau_i^2} .$$

Slotted vs. Non-slotted. So far we have presented the model for *slotted* schedules. In such schedules, all jobs have integer lengths and in addition, in the resulting schedule, all jobs should start (and therefore end) at the beginning of a time slot (that is at an integer time). However, sometimes we would like to refer to a *non-slotted* model in which the start time of a job can be any real number and so can the periods and the lengths of the jobs. All algorithms presented have both slotted and non-slotted versions.

3 A Lower Bound on the Approximation Factor

In this section we show that in general, it is impossible to achieve better results than $1+R-O(\frac{1}{t})$ for either the AVE or MAX measures, where R is the extent of the instance. This result is interesting because it holds for *any* given values of B (the maximal client size) and t (the shortest requested period). In particular, it holds even in the case where all job sizes and requested periods are powers of 2, which is a trivial case to schedule in the unit length model. In other words, the bad example below exposes an inherent discrepancy between the unit length and the multiple length models. On the other hand, it extends known lower bounds for the unit-length model [10].

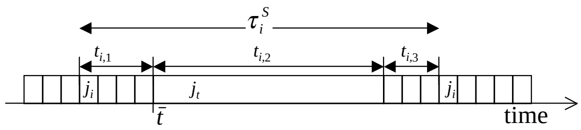

Figure 1: Illustration of the construction used for the lower bound proof.

Theorem 3.1 *For any given integers $B < t$, there exists an instance J with maximal job size B and minimal requested period t such that for all schedules S for J, we have $C_{\text{AVE}}(J,S) > 1 + R_J - \frac{2+R_J}{t}$.*

Proof: Define an instance J with $t-1$ "short" jobs and one "long" job as follows: the short jobs have $b_i = 1$ and $\tau_i = t$ for $i = 1, \ldots, t-1$; and the long job j_t has $b_t = B$, $\tau_t = Bt$. Clearly, the minimal period is t and the largest size is B. Consider any schedule S for J, and let τ_i^S denote the granted period of job j_i under S. Then, by definition and the construction above, we have

$$\begin{aligned} C_{\text{AVE}}(J,S) &= \sum_{i=1}^{t} \frac{b_i}{\tau_i^2} \cdot \tau_i^S \\ &= \sum_{i=1}^{t-1} \frac{1}{t^2} \cdot \tau_i^S + \frac{\tau_t^S}{Bt^2} \qquad (1) \\ &> \frac{1}{t^2} \sum_{i=1}^{t-1} \tau_i^S \ . \end{aligned}$$

We now bound the latter sum. Consider a time $\bar{t}$ in which the long job j_t starts, and consider, for any other job j_i, the time interval between two consecutive occurrences of j_i that contains $\bar{t}$. This interval has length τ_i^S by definition; partition it into three parts (see Figure 1): the start of j_i until $\bar{t}$; $\bar{t}$ until the start of the job following j_t; and the start of the job following j_t until the start of j_i. Denote the lengths of these sub-intervals by $t_{i,1}, t_{i,2}$ and $t_{i,3}$, respectively. By definition, we have that $\tau_i^S = t_{i,1} + t_{i,2} + t_{i,3}$, and that $t_{i,2} = B$. Hence

$$\begin{aligned} \sum_{i=1}^{t-1} \tau_i^S &= \sum_{i=1}^{t-1} (t_{i,1} + t_{i,2} + t_{i,3}) \\ &= \sum_{i=1}^{t-1} t_{i,1} + (t-1)B + \sum_{i=1}^{t-1} t_{i,3} \ . \end{aligned} \qquad (2)$$

The crucial observation is that $\sum_{i=1}^{t-1} t_{i,1} \geq \frac{t(t-1)}{2}$: this is true since the short jobs $j_1, \ldots, j_{t-1}$ must occupy $t-1$ distinct slots prior to $\bar{t}$. Similarly, $\sum_{i=1}^{t-1} t_{i,3} \geq \frac{(t-1)(t-2)}{2}$.

Thus we get from Eq. (1) and Eq. (2) that

$$\begin{aligned} C_{\text{AVE}}(J,S) &> \frac{1}{t^2}\left(\sum_{i=1}^{t-1} t_{i,1} + (t-1)B + \sum_{i=1}^{t-1} t_{i,3}\right) \\ &\geq \frac{1}{t^2}\left(\frac{t(t-1)}{2} + (t-1)B + \frac{(t-1)(t-2)}{2}\right) \\ &= \left(1 - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) \ . \quad \blacksquare \end{aligned}$$

A slightly stronger bound can be proved for the MAX measure.

Theorem 3.2 *For any given integers $B < t$, there exists an instance J with maximal job size B and minimal requested period t such that for all schedules S for J, we have $C_{\text{MAX}}(J,S) \geq 1 + R_J - \frac{1}{t}$.*

Proof Sketch: Use the same construction as above; observe that for the *last distinct* short job i_0 scheduled after $\bar{t}$, we have $t_{i_0,3} \geq t-2$. Since $t_{i_0,1} \geq 1$ by definition, the result follows. $\blacksquare$

4 The Scale & Balance Algorithm

In this section we present a basic technique for periodic scheduling with multiple lengths. The algorithm scales up the requested periods so as to have sufficient free bandwidth, and then allocates the time slots in a balanced way. We first present Algorithm bal, that assumes that there is sufficient free bandwidth, and finds the schedule with the exact requested periods. This algorithm works when all requested periods are powers of 2 times a common constant, and it is given a parameter t^* such that $t^* = t/2^e$ for some nonnegative integer e. (The parameter t^* will come into play later; for the time being, it may be convenient to assume that $t^* = t$.) The algorithm constructs a complete binary tree with T/t^* leaves (recall T is the largest requested period). Each leaf represents a run of t^* time slots. The idea in the algorithm is to spread the occurrences of jobs so as not to overload the leaves. To this end, we define a total order on jobs: for jobs j_i, j_k with requested periods τ_i and τ_k, respectively, we say that $j_i \lessdot j_k$ if either $\tau_i < \tau_k$, or if $\tau_i = \tau_k$ and $i < k$. The tree is constructed recursively, and each node will contain "job parts": each job part has its own local associated period. For the "$\lessdot$" relation, each job uses its local associated period, and its inherited job index number.

Let us briefly justify Step 3b of the algorithm. We will show that given sufficient free bandwidth in the instance, the total bandwidth associated with each leaf is at most t^*/T; since the associated period of all jobs parts in the leaves is T, it follows that the sum of lengths of all job parts in any leaf is at most t^*, and hence Step 3b cannot fail. We remark that completing the number of slots to t^* is essential to the periodicity of the resulting schedule.

Algorithm bal

Input: Instance J, parameter t^*.
Output: Schedule S.
Code:

(1) Create a complete binary tree of $1 + \log \frac{T}{t^*}$ levels. Associate all jobs of the given instance with the root.

(2) Traverse the tree, starting from the root (either depth-first or breadth-first). In each visited non-leaf node v, scan all job parts associated with v in increasing "$\lessdot$" order. For each scanned job-part j, let τ_j denote the period associated with j:

- If $\tau_j < T$, add j to both children of v, with associated periods $2 \cdot \tau_j$.
- If $\tau_j = T$, add j to the child of v whose total associated bandwidth (i.e., sum of the job lengths divided by the associated periods) is smaller. In case of a tie, add j to the left child.

(3) Scan the leaves left-to-right. For each leaf ℓ:

3a: Output the job parts associated with ℓ in increasing $\lessdot$ order.

3b: Let the number of time slots used by ℓ be denoted by s_ℓ. Add $t^* - s_\ell$ following idle time slots.

The main properties of Algorithm bal are summarized in the following lemma.

Lemma 4.1 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the periodic scheduling problem with free bandwidth $\Delta \geq R\frac{t}{t^*}$, and suppose that there exists a real number $c > 0$ and nonnegative integers $e, e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$, then Algorithm* bal *with parameter t^* finds a periodic schedule for J where the granted period of each job is its requested period.*

Proof: Feasibility follows from Lemma 4.2, and periodicity is proven in Lemma 4.3. ∎

Lemma 4.2 *If $\Delta \geq R\frac{t}{t^*}$, then the bandwidth associated with each leaf by Algorithm* bal *is t^*/T.*

Proof: Given any node x, let h_x denote its level (distance from the root), let β_x denote the total bandwidth associated with x, and define $\Delta_x = 2^{h_x} - \beta_x$. We need to show that $\Delta_\ell \geq 0$ for any leaf ℓ. We first claim that if y_1 and y_2 are children of an internal node x, then

$$\min(\Delta_{y_1}, \Delta_{y_2}) \geq \frac{\Delta_x - \frac{B}{T}}{2} . \quad (3)$$

To see that Eq. (3) is true let us assume, without loss of generality, that $\Delta_{y_1} \geq \Delta_{y_2}$. By the algorithm, $\beta_{y_2} - \beta_{y_1} \leq \frac{B}{T}$, since the children may differ at most by the maximal bandwidth of a single element. Hence

$$\Delta_{y_1} - \Delta_{y_2} \leq \frac{B}{T} . \quad (4)$$

Also, since $\beta_x = \beta_{y_1} + \beta_{y_2}$, we have

$$\Delta_{y_1} + \Delta_{y_2} = \Delta_x . \quad (5)$$

Combining Eq. (4) and Eq. (5) yields Eq. (3). Now let ℓ be any leaf. Let $p(\cdot)$ denote the "parent of" function of the tree, and let r denote the root node. Using Eq. (3), and the assumption that $\Delta_r \geq R\frac{t}{t^*}$, we can conclude that

$$\begin{aligned}
\Delta_\ell &\geq \frac{\Delta_{p(\ell)}}{2} - \frac{1}{2}\frac{B}{T} \\
&\geq \frac{\Delta_{p(p(\ell))}}{4} - \frac{B}{T}\left(\frac{1}{2} + \frac{1}{4}\right) \\
&\geq \cdots \quad > \quad \frac{t^*}{T} \cdot \Delta_r - \frac{B}{T} \\
&\geq \frac{t^*}{T} \cdot \frac{B}{t} \cdot \frac{t}{t^*} - \frac{B}{T} \quad = \quad 0 .
\end{aligned}$$ ∎

Lemma 4.3 *The schedule constructed by Algorithm* bal *is perfectly periodic.*

Proof Sketch: Associate a schedule with each node x in the tree by applying Step 3 of Algorithm bal only to the subtree rooted at x. We prove the lemma by induction on the height of the nodes. The base case is height 0, i.e., a leaf. In this case the lemma holds since in each leaf, each job appears at most once. For the induction step, let x be an interior node. Denote its left and right children by y_1, y_2, respectively, and let S, S_1, S_2 be the schedules of x, y_1, y_2, respectively. Let j_i be any job in S, with requested period τ_i. If $\tau_i = T$, then j_i is scheduled only once in S and therefore its schedule is trivially periodic. Otherwise, $\tau_i < T$, and j_i is scheduled both in S_1 and S_2. By the algorithm, j_i has the same associated period $2\tau_i$ in both y_1, y_2. Observe that by the algorithm, the start times of j_i in S_1 and S_2 depend only on jobs with smaller "$\lessdot$" value. Since each such job must appear in both S_1 and S_2, we are guaranteed that j_i has the same start in S_1 and S_2. It follows that in S, whose cycle is the cycle of S_1 followed by the cycle of S_2, the schedule of j_i is perfectly periodic. ∎

We can now state Algorithm s&b. This algorithm works for jobs whose periods are powers of 2 times a common factor. It gets a parameter t^* for the balancing part.

Theorem 4.4 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the periodic scheduling problem, and assume that there exists a real number $c > 0$ and some nonnegative integers $e, e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If*

Algorithm s&b

Input: Instance J, parameter t^*.
Output: Schedule S.
Code:

(1) Let $f = \beta + R\frac{t}{t^*}$, and let $\tau_i' = f \cdot \tau_i$ for each requested period τ_i.

(2) Run Algorithm bal with requested periods τ_i' and parameter $f \cdot t^*$.

Algorithm A

Input: Instance J.
Output: Schedule S.
Code:

(1) Round the requested periods τ_i up to the nearest powers of 2: let $\tau_i' = 2^{\lceil \log \tau_i \rceil}$.

(2) Apply Algorithm s&b to the jobs with requested periods τ_i' and parameter $t^* = \min\{\tau_i'\}$.

$t^* = t/2^e$ *then Algorithm* s&b *with parameter* t^* *finds periodic schedule* S *for* J *such that* $\frac{\tau_i^S}{\tau_i} \leq \beta + R\frac{t}{t^*}$ *for all* $i = 1, \ldots, n$.

Proof: Let $R^* = R\frac{t}{t^*}$. The bandwidth of the instance submitted to bal is $\beta' = \frac{\beta}{f} = \frac{\beta}{\beta+R^*}$ and the new extent is $R' = \frac{R}{f} = \frac{R}{\beta+R^*}$. Therefore, the free bandwidth of the instance submitted to bal is $1 - \beta' = \frac{R^*}{\beta+R^*} = R'\frac{t}{t^*}$ and the bandwidth requirement holds. The scaling keeps all periods in the form of a constant factor times a power of 2, and so the result follows from Lemma 4.1. ∎

Note that algorithms bal and s&b are presented here in the non-slotted model. To derive slotted versions of these algorithms, all we have to do is change step 3b of algorithm bal and add $(\lfloor t^* \rfloor - s_\ell)$ following idle time slots (an integral number of slots) instead of $(t^* - s_\ell)$. It follows trivially that using such a version of bal gives a granted period of at most τ_i for job j_i. Using this version of bal in s&b gives a granted period of at most $f \cdot \tau_i$ for job j_i. The correctness of this slotted version of bal is due to the integrality of the job lengths and hence the integrality of s_ℓ. If $s_\ell < t^*$ then $s_\ell < \lfloor t^* \rfloor$ as well.

5 A $\frac{9}{8} + O(R)$ Approximation Algorithm for AVE

In this section we present an algorithm for the AVE measure, without any preconditions. This algorithm improves on the known upper bound for the unit-length model presented in [10], and it demonstrates the applicability of Algorithm s&b.

The algorithm relies on Algorithm s&b and a powerful lemma that bounds the approximation factor for AVE in terms of the free bandwidth and bounds on the individual ratios. We start by stating the lemma, which is a variant of the lemma first proven in [10] for the unit length case.

Lemma 5.1 (Leftover Lemma for multiple lengths) *Let* $\{b_1, \ldots, b_n\}$, $\{\tau_1, \ldots, \tau_n\}$ *and* $\{\tau_1', \ldots, \tau_n'\}$ *be sets of positive numbers such that* $\sum \frac{b_i}{\tau_i} = 1$ *and* $\sum \frac{b_i}{\tau_i'} \leq 1$. *Let* $\Delta = 1 - \sum \frac{b_i}{\tau_i'}$. *If for some* ρ_l, ρ_h *it holds that* $\rho_l \leq \frac{\tau_i'}{\tau_i} \leq \rho_h$ *for all* i, *then* $\sum \frac{b_i \tau_i'}{\tau_i^2} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$.

The proof is a straightforward adaptation of the proof of the Leftover Lemma for unit-size jobs which is Lemma 2.5 in [10].

Lemma 5.2 (Lemma 2.5 in [10]) *Let* $\{a_1, \ldots, a_n\}$ *and* $\{f_1, \ldots, f_n\}$ *be such that for all* i, $a_i, f_i > 0$ *and* $\sum a_i = 1$, $\sum f_i \leq 1$. *Let* $\Delta = 1 - \sum f_i$. *If for all* i, $\rho_l \leq \frac{a_i}{f_i} \leq \rho_h$, *for some* $\rho_l \leq \rho_h$, *then* $\sum \frac{a_i^2}{f_i} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$.

Proof of Lemma 5.1: For all i, define a_i to be $a_i = \frac{b_i}{\tau_i^r}$ and f_i to be $f_i = \frac{b_i}{\beta^g \tau_i^g}$. Our new sets of $\{a_i\}$ and $\{f_i\}$ with ρ_l, ρ_h agree with the terms of Lemma 5.2 and therefore $\sum \frac{a_i^2}{f_i} = \sum \frac{b_i \tau_i^g}{\tau_i^{r\,2}} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$. ∎

The Leftover Lemma enables us to analyze the approximation factor of Algorithm A, which is our first result for *any* instance.

Theorem 5.3 *Let* $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ *be an instance for the periodic scheduling problem with requested bandwidth* 1, *and let* S *be the schedule produced for* J *by Algorithm* A. *Then* $C_{\text{AVE}}(J, S) \leq \frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2$.

Proof Sketch: Let β' and β_f denote the total bandwidths after Steps 1 and 2, respectively. Let $R' = B/\min\{\tau_i'\}$, i.e., R' is the extent of the instance submitted to s&b in Step 2. Let t_i denote the granted period of job j_i. Clearly, $1 \leq \frac{\tau_i'}{\tau_i} < 2$. Step 2 just multiplies the τ_i's by $f = \beta' + R'$ and hence $\rho_l \leq \frac{t_i}{\tau_i} \leq \rho_h$ for $\rho_l = f$ and $\rho_h = 2f$. Also note that $\beta_f = \beta'/f$. From the Leftover Lemma we get

$$\begin{aligned} C_{\text{AVE}}(J, S) &\leq f + 2f - 2f^2 + 2f^2\Delta_f \\ &= -2\beta'^2 + (3 - 2R')\beta' + 3R' \end{aligned}$$

Using elementary calculus we find that the latter expression is maximized when $\beta' = \frac{3-2R'}{4}$, and since $R' \leq R$, we get that the worst-case performance is therefore

$$\begin{aligned} C_{\text{AVE}}(J, S) &\leq (\beta' + R')(3 - 2\beta') \\ &\leq \frac{1}{8}(3 + 2R')^2 \\ &\leq \frac{1}{8}(3 + 2R)^2 \\ &= \frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2\ . \quad ∎ \end{aligned}$$

Algorithm P

Input: Schedule S, parameter p.
Output: Schedule S'.
Code:

(1) Partition each bin into p parts of size w/p each.

(2) Scan the bin parts in order. For each bin part z in turn:

2a: If the last job in z is not completely contained in z, add it to z and remove it from $z+1$.

2b: The schedule associated with z is all jobs in order, followed by idle time slots as to get total length of $w/p + B$ slots.

6 Separable Schedules

In this section we introduce an additional technique for periodic schedules, based on the concept of *Separable Schedules*. Separable schedules are an abstraction of the schedules produced by the s&b algorithm. We show how to perform certain operations on them while bounding the approximation factor. We show how to split and merge separable schedules with low cost and then show how cutting to the right amount and merging gives a good approximation factor.

Definition 6.1 *A schedule S is called* separable *if S can be partitioned into equal-size runs of time slots called* bins *such that the following conditions hold true.*

(1) Each occurrence of a job that starts in some bin z ends in bin z.

(2) Each job appears at most once in each bin.

(3) If a job j starts in some bin z, there are no idle time slots before j in z. Furthermore, all jobs that start in z before j, occur in each bin j occurs in, and they all start before j in each of these bins.

(4) The occurrence of jobs in bins is periodic. That is, if a job appears in bin k and in bin $k+l$, then it also appears in bin $k+il$ for all integers i.

The following property is a direct consequence of Properties 4 and 3.

Lemma 6.1 *A separable schedule is periodic.*

We now describe the operation of splitting a separable schedule, creating another separable schedule with larger periods. The algorithm is given, as input, a separable schedule S with bins of size w, and a natural number p.

The following theorem summarizes the properties of Algorithm P.

Lemma 6.2 *Let S be a separable schedule with bin size w. Then for any given p, Algorithm* P *outputs a separable schedule S' with bin size $\frac{w}{p} + B$ such that $\tau_i^{S'} \leq (1 + \frac{pB}{w})\tau_i^S$.*

Algorithm M

Input: Separable schedules $S_1, \ldots, S_k$.
Output: Merged schedule S.
Code:

(1) Output the round robin schedule of bins: the first bin in S_1, followed by the first bin in S_2, and so on, until the first bin in S_k, followed by the second bin in S_1 etc.

Note that algorithm P as described here works at a non-slotted model. It is very easy to create a slotted version of algorithm P by truncating the bin sizes of the new schedule to $\left\lfloor \frac{w}{p} + B \right\rfloor$. In such case we get $\tau_i^{S'} = \frac{p}{w} \left\lfloor \frac{w}{p} + B \right\rfloor \tau_i^S \leq (1 + \frac{pB}{w})\tau_i^S$. We prefer to use a non-slotted model here in order to simplify the mathematics of the algorithms below.

Next, we define the operation of merging separable schedules. This time, the resulting schedule is not necessarily separable. The input is k separable schedules $S_1, \ldots, S_k$ with bin sizes $w_1, \ldots, w_k$, respectively. The job sets of the schedules are disjoint.

We have the following result.

Lemma 6.3 *Let $S_1, \ldots, S_k$ be separable schedules with bin sizes $w_1, \ldots, w_k$, respectively. Then Algorithm* M *outputs a periodic schedule S such that for all $j \in S_i$ we have $\tau_j^S = \frac{W}{w_i}\tau_j^{S_i}$, where $W = \sum_{i=1}^k w_i$.*

Finally, we prove that the schedules produced by Algorithm s&b are separable. This property will be used to apply the spilt and merge operations on schedules produced by Algorithm s&b.

Lemma 6.4 *Let J be an instance of the periodic scheduling problem. Let $t^* = t_J/2^e$ for some nonnegative integer e. Then Algorithm* s&b *with parameter t^* produces a separable schedule with bin size $t^*(\beta_J + R_J\frac{t_J}{t^*}) = t^*\beta_J + B_J$.*

Proof Sketch: Consider the schedule generated by s&b, and consider each leaf as a bin. By construction, each bin has size $t^*(\beta_J + R_J\frac{t_J}{t^*}) = t^*\beta_J + B_J$, and hence Property 1 of Definition 6.1 holds. Property 2 follows from the fact that t/t^* is a power of 2. Property 3 holds from the fact that a job j appears before job j' in a leaf only if $j \lessdot j'$. Property 4 follows from Theorem 4.4. ▮

7 A General Algorithm for MAX

In this section we present our general algorithm for the MAX measure called Algorithm C. The algorithm is presented using parameters k and L that are determined later.

Algorithm C

Input: Instance J, parameters k, L.
Output: Schedule S.
Code:

(1) Round the requested periods up to the next powers of $2^{\frac{1}{k}}$. Formally, let $\tau_i' = 2^{\frac{1}{k}\lceil k \log \tau_i \rceil}$.

(2) Partition the jobs into k classes $G_0, \ldots, G_{k-1}$ according to their τ' values: Job j is in G_l if $\tau_j' = 2^{e+\frac{l}{k}}$ for some integer e.

(3) Let $t' = \min\left\{\tau_j' \mid j \in J\right\}$. Let l^* be such that $G_{l^*} \ni j$ for some job j with $\tau_j' = t'$. Define $t_0 = \frac{t'}{2^{l^*/k}}$, and for $l = 1, \ldots, k-1$, define $t_l = t_0 \cdot 2^{l/k}$.

(4) Apply Algorithm s&b to each class G_l with parameter t_l. Let S_l denote the resulting schedule for class G_l.

(5) Apply Algorithm P to each schedule S_l, with parameter $p_l = \lceil L \cdot 2^{l/k} \rceil$.

(6) Apply Algorithm M to the k schedules produced, and output the resulting schedule.

Since the analysis is a bit complicated, we present the single server case in more detail, and then explain how to generalize it to multiple servers.

Theorem 7.1 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling, and let S be the schedule produced by Algorithm* C *for J with parameters k, L. Then $C_{\mathrm{MAX}}(J,S) \le (1+\frac{1}{k}) \cdot (1+\frac{1}{L}) \cdot (1+2k(L+1)R_J)$.*

Proof: First, observe that Algorithm s&b is applicable in Step 4: This is true since the periods of all jobs in the same G_l class are powers of 2 up to a common factor of $2^{\frac{l}{k}}$. In addition, the minimal period of jobs in G_l is the minimal period in G_{l^*} times $2^{e+\frac{l-l^*}{k}}$ for some integer e, and hence t_l is a power of $\frac{1}{2}$ times the minimal period in G_l.

Next, we analyze the approximation factor. Step 1 contributes a factor of at most $2^{\frac{1}{k}}$. For Step 4, let β_l denote the total bandwidth of jobs in G_l. By Theorem 4.4, Step 4 increases the periods of jobs in G_l by a factor $f_l = \beta_l + \frac{B}{t_l}$. Define $r_l = B/t_l$. Note that r_l is an upper bound on the extent of the jobs in G_l. Let $R_l = r_l/f_l$: R_l is an upper bound on the extent of the jobs in S_l. To analyze Step 5, note that by Lemma 6.2 and Lemma 6.4, the periods are increased by a factor of $1 + \frac{p_l B}{t_l f_l} = 1 + p_l R_l$. The latter expression can be bounded as follows.

$$1 + p_l R_l \quad \le \quad 1 + (L2^{l/k}+1)\frac{r_l}{f_l} \quad = \quad \frac{\beta_l + 2r_l + Lr_0}{\beta_l + r_l}.$$

To analyze Step 6, we compute the bin size w_l for each S_l produced by Step 5. On the one hand, since $\lceil L2^{l/k} \rceil \ge L2^{l/k}$, we get

$$w_l \quad \le \quad \frac{f_l t_l}{L \cdot 2^{l/k}} + B \quad = \quad \frac{t_0}{L}(\beta_l + r_l + Lr_0)\ .$$

On the other hand, since $\lceil L2^{l/k} \rceil < L2^{l/k} + 1$, we get, after some algebraic manipulation,

$$\begin{aligned} w_l &> \frac{f_l t_l}{L \cdot 2^{l/k} + 1} + B \\ &= \frac{t_0(\beta_l + r_l + Lr_0(1 + 2^{-l/k}L^{-1}))}{L(1 + 2^{-l/k}L^{-1})} \\ &= \frac{t_0(\beta_l + 2r_l + Lr_0)}{L(1 + 2^{-l/k}L^{-1})}\ . \end{aligned}$$

Hence we have that the sum of the bin sizes W is at most

$$\begin{aligned} W &= \sum_{l=0}^{k-1} w_l \\ &\le \sum_{l=0}^{k-1} \frac{t_0}{L}(\beta_l + r_l + Lr_0) \\ &\le \frac{t_0(1 + kr_0(1+L))}{L}\ . \end{aligned}$$

Now we can apply Lemma 6.3 to get that Step 6 increases the periods by at most

$$\begin{aligned} \frac{W}{w_l} &\le \frac{\frac{t_0}{L}(1 + kr_0(1+L))}{\frac{t_0(\beta_l + 2r_l + Lr_0)}{L(1+2^{-l/k}L^{-1})}} \\ &= \frac{(1+2^{-l/k}L^{-1})(1 + kr_0(1+L))}{\beta_l + 2r_l + Lr_0}\ . \end{aligned}$$

To conclude, we multiply together all factors affecting the periods, and find that

$$\begin{aligned} C_{\mathrm{MAX}}(J,S) &\le 2^{1/k} \cdot f_l \cdot (1 + p_l R_l)\frac{W}{w_l} \\ &\le 2^{1/k}(1+2^{-l/k}L^{-1})(1 + k(r_0 + Lr_0)) \\ &= 2^{1/k}(1+2^{-l/k}L^{-1})(1 + k(1+L)r_0) \\ &\le \frac{L+1}{L} \cdot \frac{k+1}{k} \cdot (1 + 2kR_J(1+L))\ . \end{aligned}$$

The last inequality follows from the fact that $2^{\frac{1}{k}} \le 1 + \frac{1}{k}$ for $k \ge 1$, and since $r_0 < 2R_J$ by the fact that $t_l > t_J/2$ for all l. ∎

Corollary 7.2 *For any instance J of the periodic scheduling problem, there exists a schedule S such that*

$$\begin{aligned} C_{\mathrm{MAX}}(J,S) &\le 1 + 3(2R_J)^{\frac{1}{3}} + O(R^{\frac{2}{3}}) \\ &< 1 + 3.78{R_J}^{\frac{1}{3}} + O(R^{\frac{2}{3}})\ . \end{aligned}$$

Proof: Apply Algorithm C with parameters $k = L = (2R_J)^{-\frac{1}{3}}$. ∎

Algorithm genP

Input: Schedule S, parameters m, p s.t. $m|p$.
Output: Schedules $S_0, \ldots, S_{m-1}$.
Code:

(1) Apply algorithm P with parameter p on S and enumerate the resulting bins in order.

(2) For all l, the output schedule S_l is a concatenation of all bins with $x \equiv l \mod m$ where x is the index of the bin. That is, the first bin is scheduled in S_1, the second in S_2, the mth in S_m, the $(m+1)$th in S_1 and so on.

7.1 A Slotted Version of Algorithm C

We now explain how to adapt this algorithm to the slotted case. In the slotted case each job has integral length and a job must start at some integral time slot. Observe the schedule we get from algorithm C, it contains blocks of size W that are composed of blocks of size w_l. The blocks of size w_l contain consequent initiations of jobs of integral lengths followed by possibly non-integral idle time. If we truncate the blocks to size $\lfloor w_l \rfloor$, we are guaranteed that no job is being cut. By doing this, the block of the merged schedule cannot get longer and therefore the periods of the jobs do not increase. Periodicity is preserved since the offset of the lth bin within the merged block is the same in all blocks of the new schedule. We receive a slotted schedule (since all jobs start at integral time points) whose performance is at least as good as the one of the non-slotted version of algorithm C.

7.2 Multiple Servers

It is almost straightforward to generalize Algorithm C above to the case of m servers. The differences are in Steps 5 and 6. In Step 5, the algorithm splits the schedule using parameter $p_l = m\left\lceil L2^{\frac{l}{k}}\right\rceil$. The result is that for each class G_l, we get a number of schedules which is a multiple of m. Now we can take each "block" of m consecutive bins and multiplex it among our m machines. This is possible since all m bins were split from the same bin and therefore share no common jobs. We formalize the new Step 5 using a generalization of algorithm P. The new algorithm, Algorithm genP, takes two arguments m, p such that $m|p$ and an input separable schedule S and creates m separable schedules $S_0, \ldots, S_{m-1}$.

The following lemma summarizes the properties of Algorithm genP.

Lemma 7.3 *Let S be a separable schedule with bin size w. Then for any given m, p such that $m|p$, Algorithm* genP *outputs separable schedules $S_0, \ldots, S_{m-1}$ with bin size $\frac{w}{p} + B$ such that $\tau_i^{S_l} = \frac{1}{m}(1 + \frac{pB}{w})\tau_i^S$ for all $j_i \in S_l$.*

Proof Sketch: We need to show is that if j_i appears in bins numbered y and z in S', then $m|(z-y)$. This follows from Step 2 of algorithm P. Assume bin z is originated from bin z^* of S and bin y was originated from bin y^* of S (obviously, $z^* \neq y^*$ since j_i cannot appear in a bin of S more than once). Since j_i appears only once in any bin of S, and since Step 2 of algorithm P assigns a bin to j_i based only on its predecessors (which are identical in z^* and y^*), if follows that for some $u < m$: $z = mz^* + u$ and $y = my^* + u$. Therefore $z - y = m(z^* - y^*)$ and $m|(z-y)$. The periods in S_l are the periods in S', reduced by a factor of m as for all m equal sized bins in S', there is only one such bin in S_l. ∎

In Step 5 we therefore apply algorithm genP for each class l with parameters $m, p_l = m\left\lceil L2^{\frac{l}{k}}\right\rceil$ to receive m schedules for each class. In Step 6, the algorithm creates m schedules by applying Algorithm M m times, each one on one schedule per class that was created by algorithm genP. Note that this algorithm is a generalization of algorithm C: for $m = 1$ we get algorithm C precisely.

The generalization of the approximation factor analysis is very similar to the one shown in Theorem 7.1.

Theorem 7.4 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling for m machines, and let S be the schedule produced by Algorithm* C *for J with parameters k, L. Then $C_{\mathrm{MAX}}(J, S)$ is no greater than*

$$\left(1 + \frac{1}{k}\right) \cdot \left(1 + \frac{1}{L}\right) \cdot \left(1 + 2k\left(L + \frac{1}{m}\right)R_J\right).$$

Corollary 7.5 *For any instance J of the periodic scheduling problem on m servers, there exists a schedule S such that*

$$\begin{aligned} C_{\mathrm{MAX}}(J, S) &\leq 1 + 3(2R_J)^{\frac{1}{3}} + O(R^{\frac{2}{3}}) \\ &< 1 + 3.78R_J^{\frac{1}{3}} + O(R^{\frac{2}{3}}). \end{aligned}$$

Proof: Apply the multiple servers version of Algorithm C with parameters $k = L = (2R_J)^{-\frac{1}{3}}$. ∎

References

[1] *Bluetooth technical specifications, version 1.1*. Available from http://www.bluetooth.com/, Feb. 2001.

[2] S. Acharya, R. Alonso, M. J. Franklin, and S. B. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proc. 1995 ACM SIGMOD*, pages 199–210, 1995.

[3] M. H. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, Dec 1985.

[4] S. Anily, C. A. Glass, and R. Hassin. Scheduling of maintenance services to three machines. *Annals of Operations Research*, 86:375–391, 1999.

[5] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation cost of periodic scheduling. In *Proc. of the 9th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 11–20, 1998.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[7] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir. Efficient Periodic Scheduling by Trees. To appear in *INFOCOM 2002*, June 2002.

[8] Z. Brakeski, V. Dreizin, and B. Patt-Shamir. Dispatching in Perfectly-Periodic Schedules. Unpublished manuscript, 2001.

[9] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queueing theory. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 376–385, 1996.

[10] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly optimal perfectly-periodic schedules. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 107–116, 2001.

[11] Y. D. Chung and M.-H. Kim. QEM: A scheduling method for wireless broadcast data. In *Proc. Sixth International Conf. on Database Systems for Advanced Applications*, pages 135–142. IEEE Computer Society, 1999.

[12] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In R. T. Snodgrass and M. Winslett, editors, *Proc. 1994 ACM SIGMOD*, pages 25–36. ACM Press, 1994.

[13] C. Kenyon, N. Schabanel, and N. Young. Polynomial-time approximation scheme for data broadcast. In *Proc. 32nd STOC*, pages 659–666, May 2000.

[14] S. Khanna and S. Zhou. On indexed data broadcast. In *Proc. 30th ACM STOC*, pages 463–472, 1998.

[15] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[16] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32:323–330, 1980.

[17] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proc. First Symposium on Operating Systems Design and Implementation*, November 1994.

[18] W. Wei and C. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2:90–93, 1983.