RC24714 (W0812-120) December 23, 2008
Computer Science

IBM Research Report

Throughput Maximization of Real-time Scheduling with Batching

Amotz Bar-Noy[1], Sudipto Guha[2], Yoav Katz[3], Joseph Naor[4], Baruch Schieber[5], Hadas Shachnai[4]

[1]Computer and Information Science Department
Brooklyn College
2900 Bedford Avenue
Brooklyn, NY 11210

[2]Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

[3]IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

[4]Computer Science Department
Technion
Haifa 3200
Israel

[5]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Throughput Maximization of Real-Time Scheduling with Batching

AMOTZ BARNOY
Computer and Information Science Department
Brooklyn College
and
SUDIPTO GUHA
Department of Computer and Information Science
University of Pennsylvania
and
YOAV KATZ
IBM Haifa Research Lab
and
JOSEPH (SEFFI) NAOR
Computer Science Department
Technion, Israel
and
BARUCH SCHIEBER
IBM T.J. Watson Research Center
and
HADAS SHACHNAI
Computer Science Department
Technion, Israel

Authors' addresses: A. Bar-Noy, Computer and Information Science Department, Brooklyn College, 2900 Bedford Avenue Brooklyn, NY 11210. Email: amotz@sci.brooklyn.cuny.eduu. Part of this work was done while the author was at the Electrical Engineering Department, Tel Aviv University, Tel Aviv, Israel.

S. Guha, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. Email: sudipto@cis.upenn.edu.

Y. Katz, IBM Haifa Research Lab, University of Haifa, Haifa 31905, Israel. Email: katz@il.ibm.com.

J. Naor, Computer Science Department, Technion, Haifa 32000, Israel. Email: naor@cs.technion.ac.il. Research supported in part by EU contract IST-1999-14084.

B. Schieber, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. Email: sbar@watson.ibm.com.

H. Shachnai, Computer Science Department, Technion, Haifa 32000, Israel. Email: hadas@cs.technion.ac.il. Part of this work was done while the author was on leave at Bell Labs, Lucent Technologies, Murray Hill, NJ.

We consider the following scheduling with batching problem that has many applications, e.g., in multimedia-on-demand and manufacturing of integrated circuits. The input to the problem consists of n jobs and k parallel machines. Each job is associated with a set of time intervals in which it can be scheduled (given either explicitly or non-explicitly), a weight, and a family. Each family is associated with a processing time. Jobs that belong to the same family can be batched and executed together on the same machine. The processing time of each batch is the processing time of the family of jobs it contains. The goal is to find a non-preemptive schedule with batching that maximizes the weight of the scheduled jobs. We give constant factor (4 or $4+\varepsilon$) approximation algorithms for two variants of the problem, depending on the precise representation of the input. When the batch size is unbounded and each job is associated with a time window in which it can be processed, these approximation ratios reduce to 2 and $2+\varepsilon$, respectively. We also give approximation algorithms for two special cases when all release times are the same.

Categories and Subject Descriptors: G.2.3 [**Discrete Mathematics**]: Applications; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms: Theory, Algorithms

Additional Key Words and Phrases: Scheduling, batching, local ratio technique

1. INTRODUCTION

In scheduling problems, exclusiveness is usually one of the basic constraints; that is, two jobs cannot be scheduled on the same machine at the same time. In this paper we explore models that need not obey this constraint and even benefit from batching several jobs of the same type together. Such models have many applications as detailed below. One example is scheduling clients in a multimedia-on-demand (MOD) system. Each client requests a specific video program at several possible times. When several clients are willing to view the same program at the same time, their requests can be batched and satisfied by a single transmission.

The above scenario can be formulated as follows. The input to the problem consists of n jobs (clients) and k parallel machines (channels). For each job (client), a *weight* (revenue) and a *processing time* is given. Also, each job is associated with either a release time and a due date that define a window of time in which it can be processed, or with an (explicit) set of possible time intervals in which it can be processed, or with a combination of both. The jobs are partitioned into F *families* (all the clients requesting the same program) for some $F \geq 1$, and all jobs belonging to a particular family have the same processing time. Jobs that belong to the same family can be *batched* and executed together in the same time that it takes to execute a single job from that family. The number of jobs (from the same family) that can be batched together can be either bounded or unbounded. The goal is to find a feasible non-preemptive schedule with batching that maximizes the weight (revenue) of the scheduled jobs. Such scheduling problems are frequently referred to as *real-time* scheduling problems with *batching of incompatible families* (*f-batch* for short), and the objective of maximizing the value of completed jobs is frequently referred to as *throughput*. In the standard $\alpha|\beta|\gamma$ notation for scheduling problems, the problem we consider here is either $P|f-batch, b, r_j|\sum w_j(1-U_j)$, or

$P|f-batch, r_j|\sum w_j(1-U_j)$, depending on whether the batch size is bounded by a parameter b or unbounded. Both versions of our problem are strongly NP-hard. Indeed, if every family consists of a single job, then our problem is equivalent to maximizing throughput in real-time scheduling when no batching is allowed (see e.g., [Bar-Noy et al. 2001]).

We note that when all jobs have the same length and the same release times with bounded batch size, our problem reduces to a special case of the *class-constrained multiple knapsack* [Shachnai and Tamir 2001] (see below). When the batch size is unbounded, we get an instance of *maximum weighted matching* in bipartite graphs (which is solvable in polynomial time).

1.1 Applications

Our problem has numerous applications in activity selection and in resource sharing among jobs with conflicting requirements. We describe applications related to multimedia-on-demand (MOD) services and production planning.

In a MOD system (see, e.g., [Dan et al. 1996], and recent performance studies in [Boggia et al. 2005; Lee et al. 2005]), clients send requests to view video programs to a centralized video server. The system has a fixed number of channels, through which the programs are transmitted to the clients. Using a multicast facility, the server can batch several clients (who wish to view the same program) to use the same communication channel. The server needs to decide which of the requests will be serviced, and in which order, such that revenue is maximized. It is assumed that clients must have some degree of "patience", in order to allow requests to be batched together. In this application, if the network has unlimited multicast capabilities, the batch size will be unbounded. However, with Internet multicasting, it is reasonable to assume bounded multicasting to assure reliability.

The production of very large-scale integrated circuits (VLSI) involves complex processes. Since orders for the products come with strict delivery times, it is important to optimize the execution of these processes. One of the processes in the wafer fabrication stage is diffusion. This process is long and is often the bottleneck in wafer fabrication. The diffusion is done in a reactor that has the capacity to process several jobs simultaneously. However, due to differences in the chemical nature of the products, jobs of different families cannot be batched together in the diffusion reactor. Suppose that a set of products from different families is given, each with an arrival time, a due date, and a revenue. The goal is to schedule the reactor to maximize the total revenue of jobs that meet their deadlines. In this application, the batch size is bounded by the size of the diffusion reactor.

Another VLSI related application is scheduling the thermal treatment for Multi-Layer Ceramic (MLC) packaging. The last stage of MLC manufacturing is a thermal treatment that is done in an oven. Due to the length of this process (around 24 hours) and the limited size of the oven, this process is the manufacturing bottleneck. Here, several jobs can be batched together for the process as long as their thermal treatment is the same, and the goal is to maximize the (weighted) throughput of jobs completed before their deadlines.

We conclude with the following illustrative "real life" optimization problem. Consider a tour operator that offers tours of different lengths. The operator employs a fixed number of guides that can lead the tours. Each tour can accommodate a limited number of participants. Assume that the operator allows "overbooking" and a reservation for a tour may be declined at any time (usually with some kind of compensation). Given reservations for various tours, each with a different projected revenue, the operator has to decide, given the limited number of tour guides, which of the reservations to honor, so as to maximize its revenue.

1.2 Contribution

Our main results are constant factor (4 or $4+\varepsilon$) approximation algorithms[1] for two variants of the problem, depending on the input instance (discrete vs. continuous time). We show that this approximation ratio reduces to 2 or $2+\varepsilon$ if the batch size is unbounded and each job is associated with a time window in which it can be processed. To the best of our knowledge, this paper gives for the first time approximation algorithms with guaranteed performance bounds for these problems. A preliminary version of our results appeared in [Bar-Noy et al. 2002].

Our approximation algorithms are based on a nontrivial application of the *local ratio* technique and are not hard to implement. Our technique can be extended to a more general problem (that generalizes also the parallel batch problem described below), where jobs in the same family have different processing times and the processing time of a batch is determined by the longest job in the batch. It can also be generalized to the case where the possible time intervals of the same job have different lengths and weights.

We consider two special cases of the problem for which we give an improved approximation ratio. In both cases each job is associated with a time window in which it has to be processed. Specifically, we give a 2-approximation algorithm in case all jobs have the same release time and equal processing time, and a $(2+\varepsilon)$-approximation algorithm in case all jobs have the same release time and equal weight. Note that by inverting the time line, the same algorithms apply also to the corresponding instances when all jobs have the same due date. A detailed description of these algorithms as well as algorithms for other special cases is given in [Katz 2001].

1.3 Related Work

Maximizing the throughput in real-time scheduling without batching was studied extensively in [Spieksma 1999; Bar-Noy et al. 2001; Berman and DasGupta 2000; Bar-Noy et al. 2001; Chuzhoy et al. 2006]. All of these papers focused on the case where jobs specify more than one time interval in which they can be performed (in either a discrete or a continuous fashion). This model captures many applications, e.g., scheduling a space mission, bandwidth allocation, and communication in a linear network. The approximation factor obtained is 2 (and $2+\varepsilon$ for the continuous case). The paper [Chuzhoy et al. 2006] gives an improved bound of $(e/(e-1)+\varepsilon)$ for the equal weight version of the problem, where e is the base of the natural logarithms, and $\varepsilon \in (0,1)$ is some small constant.

Gandhi et al. studied in [Gandhi et al. 2006] the special case of the continuous f-batch problem in which all jobs have the same (unit) processing times, the batch

[1] We define the approximation ratio as the ratio of the optimum solution to the solution given by the algorithm. By this definition the approximation factor of any algorithm is always at least 1.

size is unbounded, and each job is associated with a time window in which it can be processed. They presented an LP-based algorithm which improves the ratio of $2+\epsilon$ for arbitrary processing times (given in Section 4) to 4/3. Bansal et al. [Bansal et al. 2006] improved this bound to 6/5. Gailis and Khuller [Gailis and Khuller 2003] showed that this special case of the f-batch problem is NP-Hard, even if all jobs have equal weights.

In the *parallel batch processing (p-batch)* model, each machine can batch several jobs to run in parallel. In this model, all the jobs are assumed to belong to the same family, and thus any group of jobs can be batched together. However, jobs may have different lengths. A batch is completed when the longest job in the batch is completed. Brucker et al. [Brucker et al. 1998] showed that in this model the problem $1|p-batch|\sum w_j(1-U_j)$ is strongly NP-hard. Baptiste [Baptiste 2000] showed that, when all jobs have the same length, the bounded batch case is solvable in $O(n^8)$ steps, even when the jobs are released at different times.

A special case of the p-batch problem involves instances where jobs belong to m types (or families) and all jobs from the same type have the same length. Note that unlike the f-batch problem, in this case, jobs from different types can be batched together, and the length of the batch is the length of the longest job type in the batch. This model is often referred to as *batching with families*, in contrast to our model, of *batching with incompatible job families*. When the objective is to minimize flow time, the problem is polynomially solvable for a constant number of types (see [Hochbaum and Landy 1997]).

Batching with incompatible job families was studied previously in the Operations Research literature, however the objective functions were different than ours. Specifically, the measures considered in these works were the weighted sum of completion times or the makespan (see e.g. [Uzsoy 1995; Dobson and Nambimadom 2001; Azizoglu and Webster 2001]). The paper [Mehta and Uzsoy 1998] presented exact and heuristic algorithms for the problem of family batching with the objective of minimizing total tardiness. Pereza et al. [Fowler and Carlyle 2005] examined the case where the objective is to minimize total weighted tardiness. This problem is NP-hard even when all jobs have the same release dates; their paper presents various heuristic approaches for tackling this problem.

Uzsoy [Uzsoy 1995] studied the feasibility version of our problem, in which we need to determine whether all jobs can be scheduled and meet their due dates. The paper shows that with identical release times the feasibility problem is solvable in $O(n \log n)$ steps, using a variant of the *earliest due date* (EDD) algorithm.

When all families have the same execution times and release dates, our problem can be reduced to the class-constrained multiple knapsack. Indeed, we can associate each time slot with a knapsack of capacity b, in which we can pack unit size items (jobs) from a single class (family). An item can be packed when the corresponding job is available. A greedy algorithm proposed in [Chekuri and Khanna 2006] can be adapted to yield a 2-approximation ratio for this problem (a similar algorithm is given in Section 5).

1.4 Organization of the Paper

The rest of the paper is organized as follows. Section 2 introduces the local ratio technique and some notation. Section 3 describes the general local ratio algorithm, and Section 4 considers the unbounded case. Section 5 gives improved performance ratios for special cases when all release dates are the same. Finally, Section 6 describes some open problems.

2. PRELIMINARIES

2.1 The Local Ratio Technique

Our algorithms are based on the local ratio technique, developed by Bar-Yehuda and Even [Bar-Yehuda and Even 1985], and later extended by Bafna, Berman and Fujito [Bafna et al. 1999]. We describe below the maximization variant of the local ratio technique required for our algorithm (see, e.g., in [Bar-Noy et al. 2001]).

Let $\mathbf{w} \in \mathbb{R}^n$ be a weight vector, and let $\mathcal{F}$ be a set of feasibility constraints on vectors $\mathbf{x} \in \mathbb{R}^n$. A vector $\mathbf{x} \in \mathbb{R}^n$ is a *feasible solution* to a given problem $(\mathcal{F}, \mathbf{w})$ if it satisfies all of the constraints in $\mathcal{F}$. The *value* of a feasible solution $\mathbf{x}$ is the inner product $\mathbf{w} \cdot \mathbf{x}$. A feasible solution is *optimal* for a maximization problem if its value is larger than or equal to the value of all feasible solutions. A feasible solution $\mathbf{x}$ is a *ρ-approximate* solution, or simply a *ρ-approximation*, if $\mathbf{w} \cdot \mathbf{x} \geq \frac{1}{\rho} \cdot \mathbf{w} \cdot \mathbf{x}^*$, where $\mathbf{x}^*$ is an optimal solution. An algorithm is said to have a *performance guarantee* of ρ, if it always computes ρ-approximate solutions.

Theorem 2.1 (Local Ratio) *Let $\mathcal{F}$ be a set of constraints and let $\mathbf{w}$, $\mathbf{w_1}$, and $\mathbf{w_2}$ be weight vectors such that $\mathbf{w} = \mathbf{w_1} + \mathbf{w_2}$. Then, if $\mathbf{x}$ is a ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w_1})$ and with respect to $(\mathcal{F}, \mathbf{w_2})$, then $\mathbf{x}$ is a ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w})$.*

Proof. Let $\mathbf{x}^*$, $\mathbf{x_1^*}$, $\mathbf{x_2^*}$ be optimal solutions for $(\mathcal{F}, \mathbf{w})$, $(\mathcal{F}, \mathbf{w_1})$, and $(\mathcal{F}, \mathbf{w_2})$ respectively. Then $\mathbf{w} \cdot \mathbf{x} = \mathbf{w_1} \cdot \mathbf{x} + \mathbf{w_2} \cdot \mathbf{x} \geq \frac{1}{\rho} \cdot \mathbf{w_1} \cdot \mathbf{x_1^*} + \frac{1}{\rho} \cdot \mathbf{w_2} \cdot \mathbf{x_2^*} \geq \frac{1}{\rho} \cdot (\mathbf{w_1} \cdot \mathbf{x}^* + \mathbf{w_2} \cdot \mathbf{x}^*) = \frac{1}{\rho} \cdot \mathbf{w} \cdot \mathbf{x}^*$ □

The Local Ratio Theorem is usually applied in the following way. Given a problem defined in the above formulation, we find a decomposition of $\mathbf{w}$ into $\mathbf{w_1} + \mathbf{w_2}$ with the property that every *maximal* solution is a ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w_1})$. We solve the problem recursively with respect to $(\mathcal{F}, \mathbf{w_2})$. Then, we extend the ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w_2})$ found in the recursion to a maximal solution. The resulting solution is a ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w_1})$ and with respect to $(\mathcal{F}, \mathbf{w_2})$, thus it is a ρ-approximate solution with respect to $(\mathcal{F}, \mathbf{w})$. In most applications of the local ratio technique, the most involved part is finding the decomposition of the weight function.

The Local Ratio Theorem applies to all problems in the above formulation. Note that $\mathcal{F}$ can include arbitrary feasibility constraints and not just linear, or linear integer, constraints. Nevertheless, all successful applications of the local ratio technique to date involve problems in which the constraints are either linear or linear integer, and this is also the case for the problems treated herein.

2.2 Definitions and Notation

Suppose that n jobs $\{J_1, \ldots, J_n\}$ need to be scheduled on a set of k machines. There are F different job families; all the jobs in a family $f \in \{1, \ldots, F\}$ have the same processing time, p_f. Each job J_j belongs to a family f_j has weight w_j, and processing time $p_j = p_{f(j)}$. A problem instance may be either *discrete* or *continuous*. In a discrete instance, for each job we have an explicit list of the time intervals in which it can be scheduled. In a continuous instance, each job comes with a release date, r_j, and a due date, d_j, defining a time window in which the job can be processed (and which is typically larger than the processing time). We note that our algorithm applies also to the more general case where each job is associated with a number of possible time windows. However, to keep the presentation simple we consider the case of a single time window per job.

We distinguish between *bounded* and *unbounded* batching. In the case of unbounded batching, any number of jobs can be batched together on each of the machines, as long as they belong to the same family. In the case of bounded batching, each of the machines can process the jobs in batches of at most b jobs, with the restriction that all jobs in a batch belong to the same family. Our results apply also for the case in which each family f has a different bound b_f. To simplify the presentation we assume a single bound b.

A *job instance* is a job and a feasible time interval in which it can be executed. A *batch instance* is a set of at most b job instances; all belong to the same family and have the same execution time interval.

For any given time t, and for each family f, let $\mathcal{J}_{f,t}$ be the set of jobs from family f that can start at time t. Note that the possible batch instances of family f that can start at t are all subsets of $\mathcal{J}_{f,t}$ of size at most b. For a batch instance B, let $f(B)$ be the family of the jobs in B, $t(B)$ be the starting time of the batch B and $p(B)$ be the processing time of the jobs in batch B ($p(B) = p_{f(B)}$). Denote by $I(B)$ the time interval of this batch instance, i.e., $I(B) = [t(B), t(B)+p(B))$; $J(B)$ is the set of jobs in B, and $w(B)$ is the sum of the weights of the jobs in B, i.e., $w(B) = \sum_{j \in J(B)} w_j$.

We say that two batch instances B and B' are *simultaneous* if $I(B) = I(B')$. Two batch instances B and B' *conflict in time* if $I(B)$ and $I(B')$ intersect. Two batch instances B and B' *conflict in jobs* if $J(B)$ and $J(B')$ intersect. Two batch instances *conflict* if they conflict in either time or jobs. A batch instance B' is *contained* in batch instance B if $J(B') \subseteq J(B)$. A batch instance B is an *extension* of a batch instance B', denoted $B' \preceq B$, if B and B' are simultaneous and B contains B'. Conversely, a batch instance B is a *reduction* of a batch instance B', if B' is an extension of B. Note that a batch instance B is both an extension and a reduction of itself.

A *feasible* schedule consists of a set of batch instances $\mathcal{B}$ such that (i) all the batch instances in $\mathcal{B}$ do not conflict in jobs, and (ii) the batch instances in $\mathcal{B}$ can be partitioned into k subsets of non-conflicting batch instances, where k is the number of available machines. The objective is to find a feasible schedule that maximizes the overall weight of scheduled batches. We call this problem *real-time scheduling with batching.*

3. APPROXIMATION ALGORITHM FOR BOUNDED BATCHING

3.1 The Algorithm

We present the approximation algorithm for a discrete instance and a single machine. Later, we show how to extend it to other cases.

We start with a generic scheme based on the local ratio technique. In the scheme we consider a slightly more general problem, where instead of having a weight per job, there is a weight per batch instance. The goal is to schedule a set of non conflicting batch instances with maximum weight. Clearly, this problem is a generalization of the original problem, in which the weight of a batch instance B is the sum of the weights of the jobs in $J(B)$. Note that the number of batch instances may be super polynomial. To keep the size of the input polynomial, we assume that the weights of the batch instances are given implicitly, as described later in the polynomial time implementation (see Section 3.2).

In the scheme we consider batch instances with negative weights. Although the initial weights can be assumed to be positive, weights may become negative during the recursive calls. Also, we note that during the recursive calls, some batch instances are deleted. We refer to batch instances that were not deleted as *available.*

3.1.1 *B-Maximal Schedules.* Following the local ratio technique, our scheme generates recursively a schedule which is ρ-approximations with respect to w_2, for some $\rho \geq 1$; then, this ρ-approximate solution is extended to a *maximal* solution. We call the resulting schedule B-maximal meaning that, given a batch instance B (as specified by the algorithm), we add to the schedule the maximal possible subset of unscheduled jobs in B, while keeping the schedule feasible. Formally,

***Definition* 3.1** A schedule $\mathcal{S}$ is B-maximal if it contains a (possibly empty) batch $B' \preceq B$ such that B' cannot be replaced in $\mathcal{S}$ by any other batch B'', where $B' \prec B'' \preceq B$, without violating feasibility.

Observe that if a B-*maximal* schedule $\mathcal{S}$ does not contain all the jobs in $J(B)$, then it must contain a batch instance that conflicts with B in time and yet it is not a reduction of B.

3.1.2 *Decomposition into* $\mathbf{w_1}$ *and* $\mathbf{w_2}$*.* A key component in our scheme is the decomposition of the weight w into w_1 and w_2. We now elaborate on this step. Recall that, initially, for each batch instance B, $w(B) = \sum_{j \in J(B)} w_j$. During the recursive calls, the weight of each batch instance is changed and will be given by

$$w(B) = \sum_{j \in J(B)} u_{j,B} - \Delta_B \ . \tag{1}$$

Initially, $u_{j,B} = w_j$ and $\Delta_B = 0$, for all batch instances B. Thus, the initial weight of a batch instance B is the sum of the original weights of the jobs in $J(B)$.

Consider now a recursive call defined by Δ_B and $u_{j,B}$, for all batch instances B and jobs $j \in J(B)$. These quantities determine the weight $w(B)$ of each batch instance B. We show the decomposition of the weight $w(B)$ into $w_1(B) + w_2(B)$. It suffices to define $w_2(B)$ which is given by (1), using the updated values of $u_{j,B}$ and Δ_B. The value of $w_1(B)$ is therefore the amount of weight added or subtracted from $w(B)$ to define $w_2(B)$.

The following quantities are defined based on the values of $u_{j,B}$ *before* the update. For a batch instance B,

—let $u(B)$ be $\sum_{j \in J(B)} u_{j,B}$, and

—let $m(B)$ be $\max\{u(B')\}$ over all the batch instances B' that are extensions of B.

Later, we show how to maintain these quantities implicitly in polynomial time.

At any stage, given a batch instance $\tilde{B}$ selected by the scheme, The quantities Δ_B and $u_{j,B}$ are updated only for batch instances that conflict with $\tilde{B}$. We distinguish between three types of such conflicting batch instances:

(1) For each batch instance B that conflicts with $\tilde{B}$ only in jobs (i.e., a batch instance B for which $I(\tilde{B})$ and $I(B)$ do not intersect, but $J(\tilde{B})$ and $J(B)$ intersect), Δ_B remains unchanged. For each $j \in J(B) \cap J(\tilde{B})$, $u_{j,B}$ is decremented by its "relative share" of $w(\tilde{B})$; that is,

$$u_{j,B} = u_{j,B} - \frac{u_{j,\tilde{B}}}{u(\tilde{B})} \cdot w(\tilde{B}) \ . \tag{2}$$

As a result, in this case,

$$w_1(B) = \frac{\sum_{j \in J(B) \cap J(\tilde{B})} u_{j,\tilde{B}}}{u(\tilde{B})} \cdot w(\tilde{B}) \ . \tag{3}$$

(2) For each batch instance B that is a reduction of $\tilde{B}$, set

$$\Delta_B = \sum_{j \in J(B)} u_{j,B} \tag{4}$$

and keep $u_{j,B}$ unchanged. Note that this implies that $w_2(B) = 0$ and $w_1(B) = w(B)$.

(3) For each batch instance B that is not a reduction of $\tilde{B}$ and which conflicts with $\tilde{B}$ in time, increment Δ_B by

$$\max\left\{\frac{1}{2}, \frac{u(B)}{m(B)}\right\} \cdot w(\tilde{B}) \ . \tag{5}$$

In words, consider the ratio of $u(B)$ to $m(B)$: if it is at most half, increment Δ_B by $\frac{1}{2}w(\tilde{B})$, otherwise increment it by $w(\tilde{B})$ multiplied by this ratio. The quantities $u_{j,B}$ are unchanged. Note that the value of $w_1(B)$ is exactly this increment in Δ_B.

The following are the steps of the algorithm.

The general Scheme:

(1) Delete all batch instances with non-positive weight.
(2) If no batch instance remains, return the empty schedule. Otherwise, proceed to the next step.
(3) Let $\mathcal{B}$ be the set of batch instances with the minimum end-time. Select a batch instance $\tilde{B} \in \mathcal{B}$ of maximum weight such that $\tilde{B}$ is maximal with respect to extension (i.e., there are no other simultaneous batch instances B' such that $J(\tilde{B}) \subset J(B')$); break ties arbitrarily.
(4) Decompose w by $w = w_1 + w_2$ with respect to $\tilde{B}$, as described above.
(5) Solve the problem recursively using w_2 as the weight function. Let $\mathcal{S}'$ be the schedule returned.
(6) If no batches in $\mathcal{S}'$ conflict with $\tilde{B}$ in time, then turn $\mathcal{S}'$ into a $\tilde{B}$-maximal schedule $\mathcal{S}$ by adding the batch instance that is a reduction of $\tilde{B}$ and which consists of all the jobs in $J(\tilde{B})$ that were not scheduled in $\mathcal{S}'$.

3.1.3 *Analysis.* In analyzing the algorithm we show that the way we choose $\tilde{B}$ and the decomposition of w satisfies the following two conditions.

The $\mathbf{w_1}$ *condition:. Every* $\tilde{B}$-maximal schedule is a 4-approximation with respect to w_1.

The $\mathbf{w_2}$ *condition:.* For every available batch instance B that is a reduction of batch instance $\tilde{B}$, $w_2(B) = 0$.

Proposition 3.2 *Suppose that the method for choosing* $\tilde{B}$ *and decomposing the weight function satisfies both the* $\mathbf{w_1}$ *and the* $\mathbf{w_2}$ *conditions. Then, the schedule* $\mathcal{S}$ *returned by the algorithm is a 4-approximation.*

Proof. Clearly, the first step in which instances of non-positive weight are deleted does not change the optimal value. Thus, it is sufficient to show that $\mathcal{S}$ is a 4-approximation with respect to the remaining instances. The proof is by induction on the number of recursive calls. At the basis of the recursion, the schedule returned is optimal (and hence a 4-approximation), since no instances remain. For the induction step, assume that $\mathcal{S}'$ is a 4-approximation solution with respect to w_2. Note that $\mathcal{S}$ is either the same as $\mathcal{S}'$ or is given by adding an available batch instance that is a reduction of $\tilde{B}$ to $\mathcal{S}'$. It follows from the $\mathbf{w_2}$ condition that $\mathcal{S}$ is a 4-approximation with respect to w_2. Since $\mathcal{S}$ is $\tilde{B}$-maximal, it follows from the $\mathbf{w_1}$ condition that it is also a 4-approximation with respect to w_1. Thus, by the Local Ratio Theorem, it is a 4-approximation with respect to w. □

Next, we need to show that our choice of $\tilde{B}$ and the decomposition of w into w_1 and w_2 satisfies the $\mathbf{w_1}$ and $\mathbf{w_2}$ conditions. By the definition of $w_2(B)$, for any batch instance that is a reduction of $\tilde{B}$ we have that the $\mathbf{w_2}$ condition is satisfied. It remains to show that the $\mathbf{w_1}$ condition is satisfied, namely, every $\tilde{B}$-maximal schedule is a 4-approximation with respect to w_1. For this we show that the optimal weight is at most $2w_1(\tilde{B})$, and that the weight of every $\tilde{B}$-maximal solution is at least $\frac{1}{2}w_1(\tilde{B})$.

Lemma 3.3 *The optimal solution with respect to* w_1 *is at most* $2w_1(\tilde{B})$.

Proof. Consider an optimal solution that consists of a set of batch instances $\mathcal{B}$. We handle separately two cases.

(i) If the optimal solution contains $\tilde{B}$, then it cannot contain any other batch instance that conflicts with $\tilde{B}$. In addition, by the definition of w_1, the weight of all the batch instances in $\mathcal{B}$ that do not conflict with $\tilde{B}$ is zero. It follows that $\sum_{B\in\mathcal{B}\setminus\{\tilde{B}\}} w_1(B) = 0$, and the weight of the optimal solution is $w_1(\tilde{B})$.

(ii) Suppose that the optimal solution does not contain $\tilde{B}$. In this case it may contain batch instances that conflict with $\tilde{B}$ only in jobs, and at most one batch instance, $\hat{B}$, that conflicts with $\tilde{B}$ in time. By (3), for any batch instance B that conflict with $\tilde{B}$ only in jobs $w_1(B) = \frac{\sum_{j\in J(B)\cap J(\tilde{B})} u_{j,\tilde{B}}}{u(\tilde{B})} \cdot w(\tilde{B})$. Since the sets $J(B)$ for all batch instances $B \in \mathcal{B}$ that conflict with $\tilde{B}$ only in jobs are mutually disjoint, the union of these sets may include at most one copy of each job $j \in \tilde{B}$. Let $H \subseteq J(\tilde{B})$ be the subset of the jobs in $J(\tilde{B})$ included in the union of these sets. The sum of the weights of all these batch instances is bounded by

$$\frac{\sum_{j\in H} u_{j,\tilde{B}}}{u(\tilde{B})} \cdot w(\tilde{B}) \le w(\tilde{B}) = w_1(\tilde{B}) \ .$$

Now, consider the batch instance $\hat{B}$ that conflicts with $\tilde{B}$ in time.

(a) If $\hat{B}$ is not a reduction of $\tilde{B}$ then $w_1(\hat{B})$ equals to the increment in $\Delta_{\hat{B}}$ which is bounded (using (5)) by $w_1(\tilde{B})$.

(b) Suppose that $\hat{B}$ is a reduction of $\tilde{B}$. Since $I(\hat{B}) = I(\tilde{B})$, both $\hat{B}$ and $\tilde{B}$ start to conflict in time with the batch instances chosen in the recursive calls at the same point of time. Also, following the first such call, both conflict in time with all the batch instances chosen in subsequent recursive calls, until the recursive call in which $\tilde{B}$ is chosen. Note that this implies that $u_{j,\hat{B}} = u_{j,\tilde{B}}$, for all $j \in J(\hat{B}) \subset J(\tilde{B})$. This is because $u_{j,\hat{B}}$ and $u_{j,\tilde{B}}$ are updated in the same way in all recursive calls before $\hat{B}$ and $\tilde{B}$ start to conflict in time with the batch instances chosen in the recursive calls, and are kept unchanged from that point on. It follows that $u(\hat{B}) \le u(\tilde{B})$. By our choice of $\tilde{B}$, $m(\hat{B}) = u(\hat{B})$ and $m(\tilde{B}) = u(\tilde{B})$. Hence, from (5), $\Delta_{\hat{B}} \ge \frac{u(\hat{B})}{u(\tilde{B})}\Delta_{\tilde{B}}$. It follows that

$$u(\hat{B})-\Delta_{\hat{B}} \le u(\hat{B})-\frac{u(\hat{B})}{u(\tilde{B})}\Delta_{\tilde{B}} = \frac{u(\hat{B})}{u(\tilde{B})}(u(\tilde{B})-\Delta_{\tilde{B}}) \le u(\tilde{B})-\Delta_{\tilde{B}} = w_1(\tilde{B}) \ .$$

□

Lemma 3.4 *The weight of every $\tilde{B}$-maximal solution is at least $\frac{1}{2}w_1(\tilde{B})$.*

Proof. To prove this claim we distinguish between three cases.

Case 1: If $\tilde{B}$ belongs to the solution, the claim is clearly true.

Case 2: The solution contains a batch instance B that is not a reduction of $\tilde{B}$ and conflicts with $\tilde{B}$ in time. By our construction, $w_1(B)$ is equal at least to the increment in Δ_B, which is at least $\frac{1}{2}w_1(\tilde{B})$.

Case 3: Suppose that the solution does not contain any batch instance that is not a reduction of $\tilde{B}$ and conflicts with $\tilde{B}$ in time. The solution may schedule some of the jobs in $J(\tilde{B})$ in batch instances that do not conflict with $\tilde{B}$ in time. Let $H \subseteq J(\tilde{B})$ be the set of these jobs.

Case 3.1: $\sum_{j \in H} u_{j,\tilde{B}} \geq \frac{1}{2} u(\tilde{B})$. Note that, by (2), the sum of the w_1-weights taken over all batch instances that contain a job from H is

$$\frac{\sum_{j \in H} u_{j,\tilde{B}}}{u(\tilde{B})} w_1(\tilde{B}) \geq \frac{1}{2} w_1(\tilde{B}) \ .$$

Case 3.2: $\sum_{j \in H} u_{j,\tilde{B}} < \frac{1}{2} u(\tilde{B})$. Consider the batch instance B which is a reduction of $\tilde{B}$ and consists of the jobs in $J(\tilde{B}) \setminus H$. To make the solution $\tilde{B}$-maximal, the batch instance B has to be added to the solution. As in the proof of Lemma 3.3 $u_{j,B} = u_{j,\tilde{B}}$, for all $j \in J(B) \subset J(\tilde{B})$. It follows that $u(B) > \frac{1}{2} u(\tilde{B}) = \frac{1}{2} m(B)$, and thus, from (5), $\Delta_B = \frac{u(B)}{m(B)} \Delta_{\tilde{B}}$. We get that

$$\begin{aligned} w_1(B) &= u(B) - \Delta_B = u(B) - \frac{u(B)}{m(B)} \Delta_{\tilde{B}} \\ &= \frac{u(B)}{u(\tilde{B})} \left(u(\tilde{B}) - \Delta_{\tilde{B}} \right) \quad \text{since } m(B) = u(\tilde{B}) \\ &= \frac{u(B)}{u(\tilde{B})} w_1(\tilde{B}) \\ &> \frac{1}{2} w_1(\tilde{B}) \ . \end{aligned}$$

Finally, the batch B is guaranteed to be available since $\frac{1}{2} w_1(\tilde{B}) > 0$. □

3.2 Polynomial-time implementation

We now show that the algorithm can be implemented in polynomial time. To this end, we need to show that the number of recursive calls is polynomial and that the bookkeeping can be done in polynomial time. During the execution of the algorithm we say that the algorithm *processed* time t, if the minimum end-time of the batch instances currently considered by the algorithm is later than t. Each recursive call of the algorithm is associated with a *call time* which is the end-time of the batch instance $\tilde{B}$ in that call.

We first show how to maintain $u_{j,B}$ and Δ_B. Consider a batch instance B, the following observations hold by definition.

(1) $\Delta_B = 0$ as long as the algorithm has not processed time $t(B)$ (the start point of B).

(2) Δ_B is the same for all simultaneous batch instances B from the same family for which $u(B) = m(B)$.

(3) For any $j \in J(B)$, $u_{j,B}$ is identical for all batch instances B such that $t(B)$ is later than the current call time $\tilde{t}$.

(4) For any $j \in J(B)$, $u_{j,B}$ remains unchanged after the recursive call with the latest call time that is earlier than $t(B)$.

For a job $j \in J$, to maintain $u_{j,B}$ we maintain a single set of values for all batch instances B such that $t(B)$ is later than the current call time $\tilde{t}$. We maintain $u_{j,B}$ for all batch instances B such that $t(B) \leq \tilde{t} \leq t(B) + p(B)$ implicitly. This can be done since $u_{j,B}$ can be computed using $u_{j,B'}$ where $t(B') > \tilde{t}$, which is maintained: $u_{j,B}$ is $u_{j,B'}$ plus the total reductions of $u_{j,B'}$ in all the recursive calls with call times between $t(B)$ and $\tilde{t}$. Since the number of these calls is polynomial, this can be computed in polynomial time.

To maintain (implicitly) Δ_B for all batch instances B such that $t(B) \leq \tilde{t} \leq t(B) + p(B)$, we note that, from (5), Δ_B is $\max\left\{\frac{1}{2}, \frac{u(B)}{m(B)}\right\}$ times the sum of $w(\tilde{B})$ in all recursive calls with call times in the interval $[t(B), \tilde{t})$. Again, since the number of these calls is polynomial, this can be computed in polynomial time. We remark that $m(B)$ can be computed by checking if $|J(B)| < b$, and in case this is true, $m(B)$ is given by adding to $u(B)$ the weight of (at most) $b - |J(B)|$ jobs not in $J(B)$ with maximum values of $u_{j,B}$, among the jobs in $f(B)$ that can start at $t(B)$.

Consider one of the recursive calls. Recall that $\tilde{B}$ is the batch instance having the maximum weight among all batch instances having minimum end-point (denoted by $\tilde{t}$). The time $\tilde{t}$ is easy to compute since the input is discrete. To find $\tilde{B}$, we consider all the jobs that can be completed at time $\tilde{t}$. We have one candidate for $\tilde{B}$ from each family. Consider a job family f. We first find the maximum $m(B)$ for all batch instances from family f. This is the sum of (up to) b jobs from family f that can end at $\tilde{t}$ with the maximum values of $u_{j,B}$ (which is the same for all batches B from f with this end-time). The candidate is a batch instance B with $u(B)$ equals this maximum.

Finally, we remark that since in each recursive call we delete at least one batch instance that is maximal with respect to extension (and all its reductions), the number of recursive calls is polynomial.

3.3 Continuous Inputs

We now explain how to handle continuous input, i.e., the case where batches can start at any point on the time line. Our exposition follows [Bar-Noy et al. 2001]. The idea is to operate on whole windows at a time, rather than modify the parameters of individual batch instances. At each iteration we delete all windows whose batch instances have non-positive profit, and find a batch instance $\tilde{B}$ with earliest end-time among the remaining batch instances. The invariant that we maintain is that the parameters of all job instances belonging to the same window are the same. This requires splitting windows; it is easy to see that the points at which the time windows may be split are all of the form: start-time of some window plus a finite sum of lengths of batch instances (not necessarily of the same family). Since any such point can be no greater than the maximum end-time of an instance, there are only finitely many such points. Thus, this implementation always halts in finite, if super-polynomial, time. In order to attain polynomial running time we trade accuracy for speed. For any fixed ε, $0 < \varepsilon < 1$, we modify the algorithm as follows. Whenever $\tilde{B}$ is chosen such that $w(\tilde{B})$ has dropped below $\varepsilon \cdot \sum_{j \in J(\tilde{B})} w_j$ we simply delete the window containing $\tilde{B}$ and do not alter any other parameters. It can be shown that the running time of the algorithm with this change is $O(1/\varepsilon)$ times the running time of the discrete input algorithm. The approximation factor obtained degrades by an additive factor of ε, to $4+\varepsilon$. The reader is referred to [Bar-Noy et al. 2001] for more details.

3.4 Multiple Machines

The way to handle multiple machines also follows [Bar-Noy et al. 2001]. The recursive calls are the same as for the single machine case, the only change is in the value of $w_1(B)$ for every batch instance B that is not a reduction of $\tilde{B}$ and which conflicts with $\tilde{B}$ in time. The increment of Δ_B for these batch instances is $\frac{1}{k}\max\{\frac{1}{2}, \frac{U(B)}{M(B)}\}w(\tilde{B})$; that is, $\frac{1}{k}$ of the increment in the single machine case, where k is the number of machines. Similar to the technique described in [Bar-Noy et al. 2001], and similar to the proofs of Lemmas 3.3 and 3.4, it can be shown that after this modification the optimal weight is at most $2w_1(\tilde{B})$, and that the weight of every $\tilde{B}$-maximal solution is at least $\frac{1}{2}w_1(\tilde{B})$.

3.5 Hardness of Approximation

Since the batching problem is more general than the corresponding scheduling problem, where the size of a batch is exactly 1, all the hardness results for scheduling apply. In particular, for the case where discrete intervals are provided, the multiple machines case is identical to the single machine problem [Spieksma 1999]. Thus, using the Max SNP hardness shown for multiple machines and batch size one in [Bar-Noy et al. 2001], the Max SNP hardness of maximizing throughput of batch scheduling follows immediately.

4. UNBOUNDED BATCHING

We describe here how to obtain improved approximation factors for unbounded batching. For continuous inputs, the improved factor is $2+\varepsilon$. For discrete inputs, the improved factor is 2, however, we need to make the following assumption. For every job J_j and $t_1 < t_3$, suppose that $[t_1, t_2]$ and $[t_3, t_4]$ are feasible time intervals for J_j ($t_4 - t_3 = t_2 - t_1 = p_j$), then if for some $t_1 < t' < t_3$, $[t', t'']$ is feasible for any job in the family f_j, it is also feasible for J_j. We present the algorithm for the case of a single machine and discrete input. The extensions to continuous input, multiple machines, and all implementation issues are similar to the general algorithm. The algorithm follows the framework of the generic scheme. The only difference is in the way the weight function w_1 is computed and in the transformation of $\mathcal{S}'$ to $\mathcal{S}$.

We consider the jobs in $J(\tilde{B})$ in descending order of the end-time of their latest time interval: $j_1, j_2, \ldots$. Let z be the maximum index such that $\sum_{i=1}^{z} u_{j_i,\tilde{B}} \le w(\tilde{B})$. Since Δ_B can be greater than zero, it follows that z is not necessarily the size of $J(\tilde{B})$. Let

$$\delta = w(\tilde{B}) - \sum_{i=1}^{z} u_{j_i,\tilde{B}} \ .$$

Note that $\delta \neq 0$ only if $\sum_{i=1}^{z} u_{j_i,\tilde{B}} < w(\tilde{B})$. We change the definition of $\tilde{B}$-maximal solution to be a solution that schedules $j_1, \ldots, j_{z+1}$. To define the decomposition we define w_2 by showing how $u_{j,B}$ and Δ_B are updated. As in the general scheme, these quantities will be updated only for batch instances that conflict with $\tilde{B}$.

(1) Suppose that batch instance B conflicts with $\tilde{B}$ only in jobs. For each job

$j \in \{j_1, \ldots, j_z\} \cap J(B)$, set $u_{j,B}$ to zero. If $j_{z+1} \in J(B)$ then decrement $u_{j_{z+1},B}$ by δ. The value of Δ_B remains unchanged.

(2) For each batch instance B that is not a reduction of $\tilde{B}$ that conflicts with $\tilde{B}$ in time increment Δ_B by $w(\tilde{B})$.

(3) For each batch instance B that is a reduction of $\tilde{B}$, set $w_1(B) = w(B)$.

Note that in the recursive call the weight of jobs $\{j_1, \ldots, j_z\}$ is zero and thus we may assume that they are not present. If $\mathcal{S}'$ schedules a batch instance that conflicts with $\tilde{B}$ in time, then it is $\tilde{B}$-maximal. Otherwise, to turn the schedule $\mathcal{S}'$ into a $\tilde{B}$-maximal solution, we do the following. If $\mathcal{S}'$ does not schedule any jobs in $J(\tilde{B})$, we add $\tilde{B}$ to $\mathcal{S}'$ to form $\mathcal{S}$. Otherwise, $\mathcal{S}'$ scheduled some jobs in $J(\tilde{B})$. Suppose that $\mathcal{S}'$ contains a batch B such that $J(B) \cap J(\tilde{B}) \neq \emptyset$. Since b is unbounded and by our assumption $[t(B), t(B) + p(B)]$ is a feasible time intreval for all jobs $\{j_1, \ldots, j_z\}$, we may extend B to include these jobs as well. For the same reason we may extend B to include j_{z+1} as well, in case it was not scheduled already. This may only increase the value of the solution relative to w_2 and make it $\tilde{B}$-maximal.

It is easy to see that, in the problem with respect to w_1, the optimal weight is at most $2w_1(\tilde{B})$.[2] We prove that the weight of every $\tilde{B}$-maximal solution is at least $w_1(\tilde{B})$. The 2-approximation follows.

Lemma 4.1 *The weight of every $\tilde{B}$-maximal solution is at least $w_1(\tilde{B})$.*

Proof. The claim is clearly true in case batch instance $\tilde{B}$ is in the solution. Suppose that this is not the case. Note that the $\tilde{B}$-maximal solution would never schedule a batch instance that is a strict reduction of $\tilde{B}$. We distinguish between two cases. (i) There is a batch $\hat{B}$ that conflicts in time with $\tilde{B}$, then from (2) above, $w_1(\hat{B}) = w(\tilde{B})$, and the claim holds. (ii) All jobs $\{j_1, \ldots, j_{z+1}\}$ are scheduled in batch instances that do not conflict $\tilde{B}$ in time. Let $\mathcal{B}$ be the set of batch instances in which these jobs are scheduled. By our construction $\sum_{B \in \mathcal{B}} w_1(B) = w_1(\tilde{B})$. □

5. IDENTICAL RELEASE TIMES

In this section we consider the case when all jobs have the same release time. We improve the approximation ratio to 2 and $(2+\varepsilon)$, respectively, when all job families have the same processing times, and when all jobs have the same weight (with arbitrary processing times). Both algorithms apply the local ratio technique.

5.1 Identical Processing Times

Note that since all release times are the same, there exists a optimal schedule with no idle times. Set the release time to 0. It follows that all scheduled batches start at times which are multiples of their processing time p.

We modify the local ratio algorithm described in Section 3 as follows. First, we consider only batch instances B such that $t(B)$ is a multiple of p. This implies that if two batches B and B' conflict in time it must be that $I(B) = I(B')$. Batch instance $\tilde{B}$ is selected as in the general scheme. To define the decomposition we

[2] The proof is similar to the proof of Lemma 3.3.

specify how $u_{j,B}$ and Δ_B are updated. The only difference from the the general scheme is in the way these quantities are updated for each batch instance B that is not a reduction of $\tilde{B}$ that conflicts with $\tilde{B}$ in time. For each such batch instance B set Δ_B to be $w(\tilde{B})$.

Since batch instance $\tilde{B}$ has the maximum weight among all batch instances B with $I(B) = I(\tilde{B})$, in the recursive call the weight of all batch instances B that conflict with $\tilde{B}$ in time is at most zero and thus we may assume that they are not present. To turn the schedule $\mathcal{S}'$ into a $\tilde{B}$-maximal solution, we do the following. If $\mathcal{S}'$ does not schedule any jobs in $J(\tilde{B})$, we add $\tilde{B}$ to $\mathcal{S}'$ to form $\mathcal{S}$. Otherwise, we add the reduction of $\tilde{B}$ consisting only of the jobs not scheduled by $\mathcal{S}'$.

Using an argument similar to the one used for the general case it can be shown that, in the problem with respect to w_1, the optimal weight is at most $2w_1(\tilde{B})$. The modified way we set Δ_B implies that the weight of every $\tilde{B}$-maximal solution is at least $w_1(\tilde{B})$. The 2-approximation follows.

It is easy to construct and example where this algorithm indeed achieves a 2 approximation ratio. Just consider two job families each consisting of a single job. Let $p = 1$ and $d_1 = 1$ and $d_2 = 2$. Suppose that both jobs have the same weight. The algorithm may schedule only job J_2 at time 0 while the optimal solution schedules both jobs.

The implementation of the algorithm is similar to the discrete case and since we may assume without loss of generality that the due dates take one of the n values $\{p, 2p, \ldots, np\}$, this implementation is strongly polynomial.

5.2 Identical Weights

We consider the case when all jobs have equal weights (with arbitrary processing time for each job family). We use several properties of optimal schedules for jobs with identical release times. (See [Uzsoy 1995].)

Proposition 5.1 *There exists an optimal solution for* $1|f-batch|\sum w_j(1-U_j)$, *such that*

(*1*) *All jobs of the same family are scheduled in the order of their due dates.*

(*2*) *All batches, except possibly for the last batch from each family, are full. If a batch* B *is not full, then it consists of the* $|J(B)|$ *jobs with the latest due dates in family* $f(B)$.

(*3*) *For each batch* B, $J(B)$ *consists of a consecutive block of jobs from the sequence of jobs in* $f(B)$ *ordered by their due dates.*

We modify the local ratio algorithm described in Section 3 as follows. First, we consider only batch instances that satisfy the conditions of Proposition 5.1. The batch instance $\tilde{B}$ is chosen to be a batch instance with the minimum end-time among all batch instances. Among such batch instances batch instance $\tilde{B}$ is chosen to be a batch instance that contains the job with the earliest due date. (Note that only one such instance exists.) For each batch instance B that conflicts with $\tilde{B}$ in either time or jobs, define $w_1(B) = w(\tilde{B})$. Set $w_1(B) = 0$ for the rest of the batch instances.

Clearly, every schedule contains at most one batch instance that conflicts with $\tilde{B}$ in time. Since we consider only batch instances that satisfy the conditions of

Proposition 5.1, it is not difficult to see that every schedule contains at most one batch instance that conflicts with $\tilde{B}$ in jobs. It follows that, in the problem with respect to w_1, the optimal weight is at most $2w_1(\tilde{B})$.

If $\mathcal{S}'$ schedules a batch instance that conflicts with $\tilde{B}$, then it is $\tilde{B}$-maximal. Otherwise, we add $\tilde{B}$ to schedule $\mathcal{S}'$ to turn it into a $\tilde{B}$-maximal solution. Clearly, the weight of every $\tilde{B}$-maximal solution with respect to w_1 is $w_1(\tilde{B})$.

The 2-approximation follows. However, the polynomial implementation of the algorithm degrades the approximation ratio to $(2+\varepsilon)$.

6. OPEN PROBLEMS

We have presented 2 and $(2+\varepsilon)$-ratio approximation algorithms for several classes of instances of our problem, including the case of unbounded batch size. These bounds coincide with the best known bounds for the real-time scheduling problem. The question whether our bounds (of 4 and $(4+\varepsilon)$) for general instances can be matched with the bounds for real-time scheduling remains open.

We have not been able to find exact polynomial time algorithms, or to establish hardness results for some classes of instances that are polynomially solvable in the corresponding real-time scheduling problem, including, for example, the problem $1|f-batch|\sum(1-U_j)$.

The existence of PTAS for subclasses of instances, and in particular, for the special case where $b=1$ (the real-time scheduling problem), is still unresolved.

REFERENCES

Azizoglu, M. and Webster, S. 2001. Scheduling a batch processing machine with incompatible job families. *Computer and Industrial Engineering 39*, 325–335.

Bafna, V., Berman, P., and Fujito, T. 1999. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics 12*, 289–297.

Bansal, N., Coppersmith, D., and Sviridenko, M. 2006. Improved approximation algorithms for broadcast scheduling. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm*. 344–353.

Baptiste, P. 2000. Batching identical jobs. *Mathematical Methods of Operations Research 53*, 355–367.

Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., and Schieber, B. 2001. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM 48*, 1069–1090.

Bar-Noy, A., Guha, S., Katz, Y., Naor, J., Schieber, B., and Shachnai, H. 2002. Throughput maximization of real-time scheduling with batching. In *Proceedings of the 13th annual ACM-SIAM symposium on Discrete algorithm*. 742–751.

Bar-Noy, A., Guha, S., Naor, J., and Schieber, B. 2001. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing 31*, 331–352.

Bar-Yehuda, R. and Even, S. 1985. Local ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathemathics 25*, 27–46.

Berman, P. and DasGupta, B. 2000. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization 4*, 307–323.

Boggia, G., Camarda, P., Mazzeo, L., and Mongiello, M. 2005. Performance of batching schemes for multimedia-on-demand services. *IEEE Transactions on Multimedia 7*, 920–931.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C., Tautenhahn, T., and de Velde, S. L. V. 1998. Scheduling a batching machine. *Journal of Scheduling 1*, 31–54.

Chekuri, C. and Khanna, S. 2006. A ptas for the multiple knapsack problem. *SIAM Journal on Computing 35*, 713–728.

Chuzhoy, J., Ostrovsky, R., and Rabani, Y. 2006. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research 31*, 730–738.

Dan, A., Sitaram, D., and Shahabuddin, P. 1996. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems Journal 4*, 112–121.

Dobson, G. and Nambimadom, R. S. 2001. The batch loading and scheduling problem. *Operations Reserach 49*, 52–65.

Fowler, I. C. P. J. W. and Carlyle, W. M. 2005. Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research 32*, 327–341.

Gailis, R. and Khuller, S. 2003. Broadcast scheduling with deadlines. Unpublished mansucript.

Gandhi, R., Khuller, S., Parthasarathy, S., and Srinivasan, A. 2006. Dependent rounding on bipartite graphs. *Journal of the ACM 53*, 324–360.

Hochbaum, D. S. and Landy, D. 1997. Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations Reserach 45*, 874–885.

Katz, Y. 2001. Scheduling with batching and incompatible job families. M.S. thesis, Computer Science Dept., Technion, Haifa, Israel.

Lee, V. W. H., Wong, E. W. M., Ko, K.-T., and Tang, K.-S. 2005. Multimedia-on-demand systems with broadcast, batch and interactive services. *IEICE Transactions on Communication E88-B*, 3097–3100.

Mehta, S. V. and Uzsoy, R. 1998. Minimizing total tardiness on a batch processing machine with incompatible jobs types. *IIE Transactions on Scheduling and Logistics 30*, 165–175.

Shachnai, H. and Tamir, T. 2001. On two class-constrained versions of the multiple knapsack problem. *Algorithmica 29*, 442–467.

Spieksma, F. C. R. 1999. On the approximability of an interval scheduling problem. *Journal of Scheduling 29*, 442–467.

Uzsoy, R. 1995. Scheduling batch processing machines with incompatible job families. *International Journal of Production Research 33*, 2685–2708.

Received Month Year; revised Month Year; accepted Month Year