Scheduling with Batching: Two Job Types

Dorit S. Hochbaum *
IEOR Department and School of Business Administration
University of California
Berkeley, California 94720
email: dorit@hochbaum.berkeley.edu

Dan Landy †
IEOR Department
University of California
Berkeley, California 94720
email: landy@cimsim.berkeley.edu

Abstract

In this paper we present an algorithm for the 2-weight batching problem: given n_1 jobs with weight w_1, and n_2 jobs with weight w_2, all having processing time p, and given a batch setup time Δ, find a sequence of batches of jobs such that the weighted sum of the $n = n_1 + n_2$ job completion times is minimized. The algorithm has running time $O(\sqrt{n} \log n)$ when p and Δ are fixed; previously, the fastest known algorithm for this problem had running time $O(n)$. Various properties of optimal solutions are exploited to reduce the problem to one of finding the minimum entry in a certain matrix. Since this matrix has some strong structural properties, its minimum entry can be found in time that is a sublinear function of the matrix size.

1 Introduction

In this paper we present an algorithm for the 2-weight batching problem: given n_1 jobs with weight w_1, and n_2 jobs with weight w_2, all having processing time p, and given a batch setup time Δ, find a sequence of batches of jobs such that the weighted sum of the $n = n_1 + n_2$ job completion times is minimized. Our algorithm has running time $O(\sqrt{n} \log n)$ when p and Δ are fixed; previously, the fastest known algorithm for this problem had running time $O(n)$.

The problem analyzed here originates in the semiconductor manufacturing process. (For a detailed explanation of semiconductor manufacturing, see [Sze83] or [GB86].) In the production of IC chips, a wafer is etched with a series of layers by lithography machines (litho for short); these machines are characterized by a prohibitively high cost. In order to produce the photographic imprint of each

*Supported in part by the Competitive Semiconductor Manufacturing project by the Sloan Foundation and by the Office of Naval Research under grant N00014-88-K-0377

†supported in part by the Competitive Semiconductor Manufacturing project by the Sloan Foundation

layer, a *mask* is placed in the litho machine. Typically, each layer requires the placement of a different mask. Replacing a mask involves a certain amount of set-up time that may be small (on the order of magnitude of one minute) if the mask is immediately accessible, or may be considerable (typically about 15 minutes) if the mask is to be retrieved from a storage area. The most recent models of litho machines include a cartridge that allows for the storage of a number of masks, which reduces the setup time involved. In either case, due to the high capital cost of litho machines, their number is frequently limited. The result is a bottleneck in the semiconductor manufacturing process. In current practice, the utilization of litho machines is commonly less than 70%, due to time lost in setups and breakdowns. It is therefore desirable to plan the production schedule so as to minimize the number of set-ups required.

The manufacturing scenario addressed here involves the production of a single type of semiconductor chip. The production process is triggered by customer orders, that are known at the beginning of the planning process. The quality of the service depends on the time of delivery of each unit of the order (a unit will be a lot of wafers). The revenue from the sale is therefore assumed to be decreasing as the delivery date gets later. To indicate different priority levels associated with different customers, penalty weights may be assigned to each order. A single customer order may also have more than one penalty rate on various portions of the total order. The objective of maximizing revenues is then equivalent to an objective of minimizing the weighted sum of completion times in the presence of set-up costs.

In order to minimize the setup costs, the best production plan is to combine all orders in a single batch. This plan however results in a large completion time sum, as all orders will be completed at the same time, which is the time when the last order is completed. If, on the other hand, each unit is processed separately and then delivered, each unit will require its own setup time. Hence the total sum of completion times (or their weighted sum) may be excessive. Our problem is therefore to determine the number of batches in which to process the orders, and the size of each batch.

There are several important aspects of this production process that are not explicitly addressed in our model. One is the fact that the manufacturing process involves processing of wafers prior to the litho operation and following the litho operations. Due to recent advances in technology, the litho machines are no longer the only capital intensive capacity required. Etching and testing machines are now priced in a similar range. Another issue is the assumption that the orders are all given initially - that is, the model is static rather than dynamic. A relatively minor issue is that of yield. Namely, the actual number of wafers produced is less than the number being processed. In our context this is a minor issue since, particularly for a facility producing a single type chip, the yields are typically known and therefore the amount processed will be determined from the amount required multiplied by the inverse of the yield factor. In spite of these limitations, the procedures developed for our simplified scenario shed light on the real problems considered in their full complexity, in that they serve as guidelines to the planning of very good schedules.

Our algorithm for the 2-weight batching problem takes advantage of a method for finding the row minima of a specially structured matrix in sublinear time. Aggarwal et al. [AKM$^+$87] devised a method for finding the row minima of a totally monotone $n \times n$ matrix in $O(n)$ time. Although the matrix arising in the 2-weight batching problem is totally monotone, there is a cost incurred each time a matrix entry is evaluated. Thus the method of [AKM$^+$87] for finding row minima results in a slower algorithm than the one presented here. We reduce the cost of evaluating each entry by taking advantage of certain properties relating adjacent entries in the matrix.

Several authors have analyzed the batching problem when all the jobs are identical, which we will refer to as the *equal-weight problem* given n jobs with processing time p, and given a batch setup time Δ, find a sequence of batches that minimizes the sum of the job completion times. Dobson Kamarkar and Rummel [DKR87] solved a relaxation of the problem by dropping the requirement that batches have integral size. For fixed p and Δ, Naddef and Santos [NS88] presented an $O(n)$ algorithm, and Coffman, Nozari and Yannakakis [CNY89] improved the running time to $O(\sqrt{n})$. Shallcross [Sha92] pointed out that these algorithms are not polynomial in the logarithms of the problem parameters n, Δ, and p. He demonstrated a method for determining the *value* of the optimal schedule in $O(\log np \log p)$ operations, which *is* polynomial. The algorithm yields a formula which can then be used to calculate the actual batch sizes in the optimal solution. Since there are $O(\sqrt{n})$ batches (for fixed p and Δ), determining the optimal schedule requires $O(\sqrt{n})$ operations using Shallcross' method.

Coffman, Yannakakis, Magazine, and Santos [CYMS90] generalized the batching problem to the case where jobs have arbitrary processing times (but equal weights), and presented an $O(n \log n)$ algorithm to solve it. Albers and Brucker [AB93] further generalized the problem to the case where jobs have arbitrary processing times and arbitrary weights. They showed that the problem of scheduling to minimize the total weighted completion time is NP-hard, and presented an $O(n)$ algorithm for determining the optimal batch sizes when the job sequence is fixed. This algorithm solves the general weighted batching problem (when jobs have equal processing times but arbitrary weights) in $O(n \log n)$ time, because it can be shown that in this case jobs are always ordered by decreasing weight, so the algorithm for a fixed sequence can be applied after the jobs are sorted. When there are only two types of jobs, the sorting can be done in linear time; hence the total running time is $O(n)$. Hochbaum and Landy [HL94] analyzed the batching problem with the objective funtion of minimizing the weighted number of tardy jobs. They showed that the problem is NP-complete but solvable in pseudo-polynomial time, and that certain special cases are solvable in polynomial time.

The algorithms presented in [AB93] and [CYMS90] are both based upon a modified dynamic programming procedure, in which an optimal schedule for n jobs is built up one job at a time. Our algorithm for the 2-weight problem is substantially different. We show that an optimal schedule is composed of three parts: a sequence of batches containing the first type of job, a single *mixed* batch containing both types of jobs, and a sequence of batches containing the second type of job. We show that if the composition of the mixed batch is fixed, we can quickly determine the optimal batchings for the other two parts of the schedule. The problem then reduces to one of finding the optimal number of each type of job in the mixed batch. We demonstrate that this involves finding the minimum value in a matrix with special structural properties, and present a sublinear method for accomplishing this task.

The paper is organized as follows. We first review some results pertaining to the equal-weight batching problem (in which all jobs are identical). We then introduce the concept of k-batchings and prove some useful facts about them. After defining the 2-weight batching problem, we prove some structural properties of optimal solutions which can be used in an enumerative algorithm. By proving some additional properties, we are able to substantially reduce the number of schedules which must be evaluated; the resulting algorithm is presented in the final section.

2 The Equal-Weight Batching Problem

Given n jobs with processing time p, and given a batch setup time Δ, we wish to find a schedule that minimizes the sum of the n job completion times. A schedule S is a sequence of batches,

$$S = (b_1, b_2 \ldots b_k),$$

where b_i is the number of jobs in the ith batch. The completion time of every job in batch i is equal to the completion time of the batch, which is given by $i\Delta + \sum_{j=1}^{i} b_j p$. We define the cost of a schedule to be the sum of the job completion times, and we denote it by $C(S)$, where

$$C(S) = \sum_{i=1}^{k} b_i (i\Delta + \sum_{j=1}^{i} b_j p).$$

An *optimal schedule* is one that minimizes $C(S)$ over all possible schedules S. If there is more than one optimal schedule, a *shortest* optimal schedule is one with the least number of batches.

For $S = (b_1, b_2, \ldots b_k)$, a schedule for n jobs, and $i \in \{1 \ldots k\}$, the *ith increment of S* is given by

$$S^i = (b_1, \ldots, b_{i-1}, b_i + 1, b_{i+1}, \ldots, b_k).$$

It is the schedule for $n+1$ jobs obtained by adding 1 job to the ith batch of S. The $k+1$st increment of S is given by $(b_1, \ldots b_k, 1)$; it is the schedule for $n+1$ jobs obtained by adding a new batch with 1 job to the end of S. The following theorem is the basis for the algorithm in [CNY89] for the equal-weight batching problem; it was originally proved in [NS88] in a more general form (c.f. Lemma 5).

Theorem 1 (Naddef and Santos) *An optimal schedule for $n+1$ jobs is an increment of some optimal schedule for n jobs. Conversely, any optimal schedule for n jobs has an increment that is an optimal schedule for $n+1$ jobs.*

Using this result it is possible to construct an optimal schedule for n jobs one job at a time by repeatedly choosing the least-cost increment of the current schedule. Later we will make use of the following consequence of the theorem.

Corollary 2 *If A and B are optimal schedules for m and n jobs respectively, where $m < n$, then the cost of optimally incrementing A is less than the cost of optimally incrementing B.*

Proof: By the previous theorem we may assume that B is derived from A by a sequence of $n - m$ increments, and the result follows from the observation that every increment has a cost that is at least as large as the preceding increment. ∎

In establishing the correctness of their algorithm, Coffman et. al. demonstrate the following upper bounds on both the number of batches and the maximum batch size in an optimal schedule for the equal-weight batching problem.

Lemma 3 (Coffman et. al.) *When p and Δ are fixed, the number of batches in an optimal schedule is $O(\sqrt{n})$ and the size of every batch is also $O(\sqrt{n})$.*

The algorithm of Coffman et. al. determines the sizes of the batches in an optimal schedule in $O(\sqrt{n})$ operations. Shallcross [Sha92] showed that these batch sizes can also be calculated by finding parameters c and l which are used in the following formula:

$$b_i = \lfloor \frac{c - i\Delta - 1}{p} \rfloor + \begin{cases} 1 & \text{if } (c - i\Delta)/p \in \mathbf{Z},\ 1 \le i \le l \\ 0 & \text{otherwise.} \end{cases}$$

Determining the values of c and l can be done in polynomial time. However, since there are $O(\sqrt{n})$ batches (for fixed p and Δ), evaluating the formula to produce a list of the actual batch sizes requires $O(\sqrt{n})$ operations. If only the *cost* of the optimal schedule is required, we may use the following result, due to Shallcross [Sha92].

Lemma 4 (Shallcross) *Given n jobs with processing time p and batch setup time Δ, the cost of an optimal schedule for the equal-weight batching problem can be computed in $O(\log np \log p)$ operations.*

In the remainder of the paper we shall consider p and Δ to be fixed. Thus the running time of Shallcross' algorithm will be expressed as $O(\log n)$.

3 k-Batchings

A *k-batching* of n jobs of equal weight is a schedule of the n jobs with exactly k batches. An *optimal k-batching* is a k-batching that minimizes the sum of the n completion times. We denote the cost of an optimal k-batching for n jobs by $F_k^*(n)$. Let $k^*(n)$ denote the number of batches in an optimal schedule for n equal weight jobs (that is, when there is no restriction placed upon the number of batches). When there is more than one optimal schedule, $k^*(n)$ is the number of batches in the *shortest* optimal schedule, i.e. the schedule with the least number of batches. Let $F^*(n)$ denote the cost of such a schedule. The following result (from [NS88]) shows that optimal k-batchings share the structural property of optimal schedules described by Theorem 1.

Lemma 5 (Naddef and Santos) *For any $k \leq k^*(n+1)$, an optimal k-batching for $n+1$ jobs is an increment of some optimal k-batching for n jobs. Conversely, any optimal k-batching for n jobs has an increment that is an optimal k-batching for $n+1$ jobs.*

We now state a simple but useful property relating batch sizes in different k-batchings.

Lemma 6 *For any $k < k^*(n)$, let $A = (a_1 \dots a_{k+1})$ be an optimal (k+1)-batching of n equal weight jobs. Then there exists an optimal k-batching $B = (b_1 \dots b_k)$ such that*

$$b_i \geq a_i \text{ for all } i \in \{1 \dots k\}.$$

Proof: Let $A' = (a_1 \dots a_k)$ be the schedule composed of the first k batches of schedule A. A' must be an optimal k-batching for $n - a_{k+1}$ jobs, otherwise the cost of schedule A could be reduced by rebatching the first k batches optimally. By Lemma 5, an optimal k-batching for n jobs can be found by incrementing schedule A' a_{k+1} times, yielding the desired k-batching B. ■

Lemma 7 *Let $A = (a_1, \dots, a_k)$ and $B = (b_1, \dots, b_k)$ be optimal k-batchings of n and $n+1$ equal weight jobs respectively. Let $A' = (a'_1, \dots, a'_{k+1})$ and $B' = (b'_1, \dots, b'_{k+1})$ be optimal (k+1)-batchings of n and $n+1$ jobs. Then*

$$C(B) - C(B') \geq C(A) - C(A').$$

Proof: By Lemma 6 we may assume that $a_i \geq a'_i$ for all $i \in \{1 \ldots k\}$. By Lemma 5 we may assume that B is the ith increment of A for some $i \in \{1 \ldots k\}$. Thus

$$C(B) - C(A) = a_i p + i\Delta + (n+1)p \geq a'_i p + i\Delta + (n+1)p.$$

Since B' is an optimal $(k+1)$-batching for $n+1$ jobs,

$$C(B') - C(A') \leq a'_i p + i\Delta + (n+1)p,$$

for all $i \in \{1 \ldots k\}$. Combining these two inequalities yields $C(B) - C(A) \geq C(B') - C(A')$, or

$$C(B) - C(B') \geq C(A) - C(A').$$

■

Corollary 8 *Let $A = (a_1, \ldots, a_k)$ and $B = (b_1, \ldots, b_k)$ be optimal k-batchings of n and $n+d$ equal weight jobs respectively, where $d \geq 1$. Let $A' = (a'_1, \ldots, a'_{k+r})$ and $B' = (b'_1, \ldots, b'_{k+r})$ be optimal (k+r)-batchings of n and $(n+d)$ jobs, respectively, where $r \geq 0$ satisfies $k + r \leq k^*(n+d)$. Then*

$$C(B) - C(B') \geq C(A) - C(A').$$

Proof: This result follows by repeatedly applying the preceding lemma. ■

4 The 2-Weight Batching Problem

In the 2-weight problem we are given n_1 jobs of weight w_1, and n_2 jobs of weight w_2, with $w_1 > w_2$. We will call the two types of jobs *heavy* and *light* respectively. All jobs have the same integer processing time p, and there is an integer batch setup time of Δ. A schedule S is a sequence of batches and is denoted by:

$$S = (b_1, b_2, \ldots b_k),$$

where b_i is the number of jobs in the ith batch. The completion time of every job in batch i is equal to the completion time of the batch, which is given by $i\Delta + \sum_{j=1}^{i} b_j p$. We wish to find a schedule that minimizes the sum of the weighted completion times of all $n = n_1 + n_2$ jobs. Let h_i and l_i denote the number of heavy and light jobs in batch i, so $b_i = h_i + l_i$. The cost of schedule $S = (b_1, b_2, \ldots b_k)$, denoted by $C(S)$, is the weighted sum of job completion times:

$$C(S) = \sum_{i=1}^{k} (w_1 h_i + w_2 l_i)(i\Delta + \sum_{j=1}^{i} b_j p).$$

A batch which contains only heavy jobs will be called a *heavy batch*, a batch that contains only light jobs will be called a *light batch*, and a batch that contains both heavy and light jobs will be called a *mixed batch*.

Before presenting our algorithm for the 2-weight problem, we prove several structural properties of optimal schedules.

4.1 Structural Properties of Optimal Schedules

Finding an optimal schedule for a batching problem with different types of jobs can be considered as a two stage process: first a sequence of jobs is determined, and then the sequence is broken into batches. The following result shows that the sequencing problem is easily solved for the 2-weight batching problem.

Lemma 9 *In an optimal schedule for 2 types of jobs, where type i has weight $w_i, i \in \{1, 2\}$, a batch containing a light job never precedes a batch containing a heavy job. Thus there is at most one mixed batch.*

Proof: Suppose that in an optimal schedule there were a batch containing a light job preceding a batch containing a heavy job. Swapping these jobs would reduce the sum of weighted completion times, contradicting optimality. ■

An optimal schedule will therefore have some number, say k, of heavy batches, followed by at most one mixed batch, followed by some number of light batches. It is easy to see that the heavy batches will satisfy the following property.

Lemma 10 *Suppose an optimal schedule has k heavy batches. Then these batches must be an optimal k-batching.*

Proof: Otherwise, the schedule could be improved by rebatching the first k batches. ■

Lemma 11 *Let $S = (b_1, \dots b_k)$ be an optimal n-job schedule for the 2-weight problem. Then for any $j \in \{1 \dots k\}$, the schedule $S' = (b_j, \dots b_k)$ is an optimal schedule for the $\sum_{i=j}^{k} b_i$ jobs in batches $b_j, \dots, b_k$ of S.*

Proof: Otherwise, S could be improved by rebatching the $\sum_{i=j}^{k} b_i$ jobs, contradicting optimality. (The sum of the completion times of jobs in batches $b_j \dots b_k$ would be reduced, while the completion times of the preceding jobs would be unaffected). ■

The following lemma provides an upper bound on the number of heavy batches in an optimal schedule.

Lemma 12 *An optimal schedule for $n = n_1 + n_2$ jobs has at most $k^*(n_1)$ heavy batches, where $k^*(n_1)$ is the number of batches in the shortest optimal schedule for n_1 equal weight jobs.*

Proof: Suppose an optimal schedule had $m \leq n_1$ jobs in $k > k^*(n_1)$ heavy batches. Since $m \leq n_1$, Theorem 1 tells us that $k > k^*(n_1) \geq k^*(m)$. Rebatching the m jobs in $k^*(m)$ batches would reduce the sum of their completion times (by the optimality of $k^*(m)$), and would also reduce the completion times of all subsequent jobs, since the number of preceding batches would be reduced. Thus rebatching with $k^*(m)$ heavy batches would improve the schedule, contradicting its optimality. ■

We have shown that an optimal schedule for the 2-weight problem begins with an optimal k-batching of heavy jobs, for some $k \in \{1 \dots k^*(n_1)\}$. By Lemma 11, we know that removing any number of batches from the beginning of an optimal schedule leaves a subschedule that is optimal for the jobs it contains.

In particular, if there are l jobs in light batches in an optimal schedule, they are batched according to an optimal schedule for the l-job equal-weight problem.

Suppose it is known that an optimal schedule for the 2-weight problem has k heavy batches, and a mixed batch with i heavy jobs and j light jobs. Let $S_k(i,j)$ represent the optimal schedule that satisfies these requirements. By the preceding lemmas, we can calculate the cost of $S_k(i,j)$, denoted by $C_k(i,j)$, as follows:

$$\begin{aligned} C_k(i,j) &= w_1 F_k^*(n_1 - i) + (iw_1 + jw_2)(\Delta(k+1) + p(n_1 + j)) + \\ &\quad w_2 F^*(n_2 - j) + w_2(n_2 - j)(\Delta(k+1) + p(n_1 + j)). \end{aligned} \tag{1}$$

The first term is the cost of the optimal k-batching of the $n_1 - i$ jobs in heavy batches. The second term is the cost of the mixed batch. The third term is the cost of an optimal schedule for the $n_2 - j$ jobs in light batches, and the final term is the weighted delay to these light jobs caused by the preceding heavy and mixed batches. By the result of Shallcross (Lemma 4), we can calculate the third term in $O(\log n)$ time. The second and fourth terms can be evaluated in a constant number of operations. The following lemma establishes the fact that we can calculate the first term, and hence the entire expression, in $O(\log n)$ time.

Lemma 13 *The cost of an optimal k-batching of n equal-weight jobs (and hence the value of $C_k(i,j)$) can be calculated in $O(\log n)$ operations.*

Proof: The calculation is done using a modification of Shallcross' method [Sha92] for finding the cost of an optimal schedule for n jobs when there is no restriction on the number of batches. Shallcross' method involves finding parameters c and l such that n, the number of jobs, is equal to the number of integer points in the set

$$T(c-1) = \{(x,y) : x \geq 1, y \geq 1, py \leq c - \Delta x - 1\}$$

plus the number of integer points on the line segment $\{(x,y) : 1 \leq x \leq l, py = c - \Delta x\}$. Finding the number of integer points in $T(c-1)$ is done with an algorithm due to Zamanskii and Cherkasskii [ZC84].

We modify this method by finding c and l such that n is equal to the number of integer points in the trapezoidal set

$$Z(c-1) = \{(x,y) : 1 \leq x \leq k, y \geq 1, py \leq c - \Delta x - 1\}$$

plus the number of integer points on the line segment described above.

Finding c and l is exactly as described by Shallcross: by doing binary search on c in the range $\Delta \leq c \leq np + \Delta$ and then using the Euclidean g.c.d. algorithm to find l. Thus the running time of the entire calculation has the same upper bound as Shallcross' method, which is $O(\log n)$ for fixed p and Δ.

■

Let $S(i,j)$ denote the schedule with the lowest cost among those that have a mixed batch with i heavy jobs and j light jobs, and let $C(i,j)$ denote its cost. $C(i,j)$ is determined by minimizing over all possible values of k, the number of heavy batches:

$$C(i,j) = \min_{k \in \{1 \ldots k^*(n_1)\}} C_k(i,j). \tag{2}$$

It is possible to show that $C(i,j)$ can be calculated in $O((\log n)^2)$ operations; using the fact that $F_k(n) - F_{k+1}(n)$ decreases as k increases, binary search is used to find the value of k that minimizes $C_k(i,j)$. For the purposes of the algorithm presented here however, it suffices to find the minimum by testing all values of k in the range $1 \ldots k^*(n_1)$. By Lemma 3, $k^*(n_1)$ is $O(\sqrt{n})$; hence $C_(i,j)$ can be calculated in $O(\sqrt{n}\log n)$ operations.

It is clear that C^*, the cost of the optimal schedule for the 2-weight batching problem, is simply:

$$C^* = \min_{i \le n_1, j \le n_2} C(i,j).$$

A simple enumerative scheme will require the evaluation of $O(n_1 n_2)$ different schedules. The remainder of the paper is devoted to establishing properties which will significantly reduce the number of such evaluations.

5 Further Properties

In order to reduce the number of schedules which must be evaluated by an enumerative algorithm, we demonstrate the following upper bound on the size of the mixed batch in an optimal schedule.

Lemma 14 *Let $b_1^*(n)$ be the size of the first batch in an optimal schedule for $n = n_1 + n_2$ equal weight jobs. Then there exists an optimal schedule for the 2-weight problem in which the size of the mixed batch, m, satisfies $m \le b_1^*(n)$.*

Proof: Suppose every optimal schedule has a mixed batch of size $m > b_1^*(n)$. Let T be such a schedule, with a mixed batch that has h heavy jobs and l light jobs, where $h, l > 0$, and $h + l > b^*(n)$. Create schedule $A = (a_1 \ldots a_p)$ by removing all the heavy batches from T and renumbering, so that the first batch of A is mixed, with $a_1 = m = h + l$. By Lemma 11, A is an optimal schedule for h heavy jobs and n_2 light jobs. We will show that A can be improved, contradicting its optimality. Let schedule $B = (b_1, \ldots b_q)$ be an optimal schedule for $h + n_2$ *equal weight* jobs. Since $h + n_2 \le n_1 + n_2$, we have $a_1 > b_1^*(n) \ge b_1$, where the second inequality follows from Theorem 1. Since A and B have the same total number of jobs, and $a_1 > b_1$, there must exist an index $j \in \{2 \ldots p+1\}$ such that $a_j < b_j$. (This is insured by defining $a_{p+1} = 0$). Consider the schedule $B' = (b_1 + 1, b_2, \ldots b_{j-1}, b_j - 1, b_{j+1}, \ldots b_q)$, derived from B by moving a job from batch j to batch 1. By the optimality of B, moving this job from batch 1 back to batch j in B' cannot increase the cost of the schedule. The change in the sum of job completion times is given by:

$$C(B) - C(B') = (j-1)\Delta + p(b_j - b_1 - 1) \le 0$$

Compare this move with a similar one in schedule A; consider moving a light job in schedule A from batch 1 to batch j, yielding schedule A'. The change in the (unweighted) sum of completion times is given by:

$$C(A') - C(A) = (j-1)\Delta + p(a_j - a_1 + 1)$$

Since $a_1 > b_1$ and $a_j \le b_j - 1$, we have:

$$C(A') - C(A) \le C(B) - C(B') \le 0$$

In addition, the decrease of the completion times of the heavy jobs in the first batch of A is multiplied by weight $w_1 > w_2$. Thus moving a light job from batch 1 to batch j in A strictly reduces the cost of the schedule, contradicting its optimality. ∎

By Lemma 3, every batch in an optimal schedule for n equal weight jobs has a size that is $O(\sqrt{n})$. This fact, together with the preceding lemma, guarantees that the mixed batch in an optimal schedule for the 2-weight problem has a size that is $O(\sqrt{n})$.

Let $M = b_1^*(n)$ be the maximum size of the mixed batch, and let C be an $(M-1) \times (M-1)$ matrix with entries $C(i,j)$ which represent the minimum cost of a schedule for the 2-weight problem when the mixed batch is required to have i heavy jobs and j light jobs, where $i \in \{1 \dots M-1\}$, and $j \in \{1 \dots M-1\}$.

We now prove some useful properties of the entries in C. The following lemma shows that as we move across a row of the matrix the number of heavy batches does not change. Thus by Lemma 13, evaluating each new entry in a row requires only $O(\log n)$ operations.

Lemma 15 *For any fixed $i \in \{1 \dots M-1\}$, the number of heavy batches in schedule $S(i,j)$ is the same for all values of j.*

Proof: Combining terms in equation 1, we have

$$C_k(i,j) = w_1 F_k^*(n_1 - i) + (iw_1 + n_2 w_2)(\Delta(k+1) + p(n_1 + j)) + w_2 F^*(n_2 - j)$$

It is clear that the value of k that minimizes $C_k(i,j)$ is independent of j when i is fixed. ∎

The following result establishes that C is totally monotone, a structural property which makes it possible to quickly find the row minima of the matrix.

Lemma 16 *The matrix C is totally monotone, that is: For any $i < i', j < j'$, if $C(i,j) < C(i,j')$, then $C(i',j) < C(i',j')$.*

Proof: First note that for fixed j (the number of light jobs in the mixed batch), the remaining schedule is determined; it is simply an optimal schedule for $n_2 - j$ equal-weight jobs. Suppose $C(i,j) < C(i,j')$, for $j < j'$. Let $t = j' - j$. Then moving t jobs out of the mixed batch (and into light batches) in schedule $S(i,j')$ results in a decrease in cost. Consider moving t jobs out of the mixed batch in schedule $S(i',j')$, yielding schedule $S(i',j)$, where $i' > i$. This move results in the same change in completion times of light jobs, and a greater decrease for the heavy jobs in the mixed batch, since there are more heavy jobs in the mixed batch in schedule $S(i',j')$. Thus this move must also result in a net decrease in cost, and we have $C(i',j) < C(i',j')$. ∎

Having established that the matrix C is totally monotone, we may apply the algorithm of [AKM+87] for finding row minima in $O(M) = O(\sqrt{n})$ time. Since determining the value of each $C(i,j)$ can be done in $O((\log n)^2)$ operations, we can find the matrix minimum (the optimal schedule) in a total of $O(\sqrt{n}(\log n)^2)$ operations. By making use of additional structure in the matrix however, it is possible to further reduce this running time. In particular, if we know that schedule $S(i,j)$ has k batches, determining its cost only requires $O(\log n)$ operations (see Lemma 13). Before taking advantage of this insight, we prove another useful result: that the rows of C are unimodal. In other words, we may examine the entries of a row from left to right until there is no decrease, at which point we are at the row minimum. The following lemma makes this fact explicit.

Lemma 17 *For each $i \in \{1 \ldots M-1\}$, if $C(i,j) < C(i,j+1)$, then $C(i,k) < C(i,k+1)$ for all $k \geq j+1$.*

Proof: Consider the action of removing a light job from the mixed batch in schedule $S(i,j+1)$ and adding it to the optimally chosen light batch, yielding schedule $S(i,j)$. Let $\delta_1 = C(i,j+1) - C(i,j)$. By assumption, $\delta_1 > 0$. Consider a similar move of a light job out of the mixed batch in schedule $S(i,k+1)$, where $k \geq j+1$, and let $\delta_2 = C(i,k+1) - C(i,k)$. In each case, we may break the change in cost into two pieces:

$$\delta_i = R_i - A_i, \text{ for } i = 1, 2$$

where R_i is the decrease resulting from removing the job from the mixed batch, and A_i is the increase resulting from adding the job back into an optimally chosen light batch. Letting t be the index of the mixed batch in the two schedules, we have

$$R_1 = w_2(t\Delta + n_1 + j + 1) + w_1 i + w_2 n_2 \text{ and,}$$

$$R_2 = w_2(t\Delta + n_1 + k + 1) + w_1 i + w_2 n_2.$$

Thus

$$R_2 - R_1 = w_2(k - j). \tag{3}$$

A_1 is simply the cost of incrementing a schedule for $n_2-(j+1)$ light jobs, plus the weighted delay caused by the preceding heavy batches and the mixed batch. Similarly, A_2 is the cost of incrementing a schedule for $n_2 - (k+1)$ light jobs, plus a weighted delay. By Lemma 5, the cost of optimally incrementing a schedule for $n - (k+1)$ jobs is less than the cost of incrementing a schedule for $n - (j+1)$ jobs, since $n-(k+1) < n-(j+1)$. Since the cost of the heavy and mixed batches is greater in the schedule with the larger mixed batch by $k - j$ jobs, we have $A_2 < A_1 + w_2(k-j)$. Combining this with 3 gives:

$$c(i,k+1) - c(i,k) = \delta_2 > \delta_1 = c(i,j+1) - c(i,j) > 0, \text{ so}$$

$$c(i,k) < c(i,k+1).$$

■

Corollary 18 *For each $i \in \{1 \ldots M-1\}$, if $c(i,j) < c(i,j+1)$, then $c(i,j) < c(i,k)$ for all $k \geq j+1$.*

Proof: By the preceding lemma, $c(i,j) < c(i,j+1)$ implies

$$c(i,j) < c(i,j+1) \ldots < c(i,k-1) < c(i,k).$$

■

We now show that if we move vertically upwards in the matrix by one entry, (i.e. move one heavy job from the mixed batch to a heavy batch), the number of heavy batches either remains the same or increases by one.

Lemma 19 *If schedule $S(i,j)$ has k heavy batches, then schedule $S(i-1,j)$ has either k or $k+1$ heavy batches.*

Proof: Suppose that schedule $S(i,j)$ has k batches, and schedule $S(i-1,j)$ has t batches. We will show that $k \leq t \leq k+1$.

First suppose that $t < k$. By the optimality of $S(i,j)$, batching the first $n_1 - i$ heavy jobs in k batches costs less than batching them in t batches. Thus by Corollary 8, batching the first $n_1 - (i-1)$ heavy jobs of schedule $S(i-1,j)$ in k batches costs less than batching them in t batches. Furthermore, the delay to subsequent jobs (in the mixed and light batches) is greater in the first case, since there is one more job delayed. Thus since using k instead of t heavy batches is an improvement in schedule $S(i,j)$, it must also be an improvement in schedule $S(i-1,j)$, so $t < k$ yields a contradiction.

Now suppose $t > k+1$. Consider the first $k+1$ heavy batches of $S(i-1,j)$, and suppose they contain a total of m jobs. By the optimality of the schedule, using $k+1$ batches for these m jobs costs less than using k batches. Thus using $k+1$ rather than k batches for the first $n_1 - i$ jobs in schedule $S(i,j)$ would result in a decrease in cost by Corollary 8, since $m < n_1 - i$. Furthermore, using an extra batch creates a greater delay in the first case, since $n_1 + n_2 - m > n_2 + i$ jobs are delayed. Thus using $k+1$ rather than k heavy batches in schedule $S(i,j)$ would decrease its cost, contradicting optimality. ■

A useful consequence of this result is that if we have already calculated $C(i,j)$, and the corresponding schedule has k batches, then we may calculate $C(i-1,j)$ in $O(\log n)$ operations by choosing the smaller of $C_k(i-1,j)$ and $C_{k+1}(i-1,j)$. We have now established all of the structural properties required to prove the correctness and the running time of our algorithm, which is described below.

6 The 2-Weight Algorithm

The algorithm starts at the lower left corner of the matrix C and moves to the right (increasing the number of light jobs in the mixed batch) as long as such a move yields a decrease in cost. The entry where it stops is the row minimum. The next entry evaluated is the one immediately above this row minimum. The algorithm proceeds in this fashion, moving to the right and then up, until it has found the minima of all $M-1$ rows. (The correctness of this method follows from the total monotonicity of the matrix). The smallest row minimum is then the minimum for the entire matrix. The procedure is illustrated in Figure 1. The final step of the algorithm is to calculate the cost of the optimal schedule with no mixed batch and to compare it to the best schedule found in the matrix, choosing the one with smaller cost.

The 2-Weight Batching Algorithm

Step 1: {Calculate maximum size of mixed batch} $M = b_1^*(n)$.

Step 2: Calculate $k^*(n_1 - (M-1))$, the maximum number of heavy batches for $n_1 - (M-1)$ jobs.

Step 3: {Find the optimal number of heavy batches in $S(M-1,1)$}
$k := \text{argmin}_{\bar{k} \in \{1 \ldots k^*(n_1)\}} C_{\bar{k}}(M-1,1)$.

Step 4: $C^* := C_k(M-1,1)$, $i := M-2$, $j := 2$.

Step 5: {Find heavy batch number} $k := \text{argmin}\{C_k(i,j), C_{k+1}(i,j)\}$.

i \ j	1	2	3	.	.	.	.	.	.	M-1
1					→	→	→	→	*	
					↑					
2					*					
					↑					
.			→	→	*					
			↑							
.			*							
			↑							
.		→	*							
		↑								
.	→	*								
	↑									
M-1	*									

Figure 1: A Sample Search Path ($*$ indicates row minimum)

Step 6: {Find row minimum} Increment j until $C_k(i,j) \leq C_k(i,j+1)$, or $i+j=M$.

Step 7: If $C_k(i,j) < C^*$ then: $C^* := C_k(i,j)$, $i^* := i$, $j^* := j$, $k^* := k$.

Step 8: $i := i-1$. If $i \geq 1$ GOTO step 5.

Step 9: Calculate $C(0,0)$ (the cost of the optimal schedule with no mixed batch), by finding $k^o := \text{argmin}_{\bar{k}\in\{1\ldots k^*(n_1)\}} C_{\bar{k}}(0,0)$.

Step 10: If $C(0,0) < C^*$, return $C(0,0), k^o, 0, 0$. Otherwise return C^*, k^*, i^*, j^*.

Theorem 20 *The 2-Weight batching algorithm determines the cost of the optimal schedule for the 2-weight batching problem in $O(\sqrt{n}\log n)$ time.*

Proof: Step 1 can be done in $O(\log n)$ operations by using Shallcross' method to find the size of the first batch in an optimal schedule for n_1 equal weight jobs. Step 2 can also be done with Shallcross' method, which determines the number of batches in an optimal schedule for n equal-weight jobs in $O(\log n)$ operations. In step 3, $C_{\bar{k}}(M,0)$ can be calculated in $O(\log n)$ steps (by Lemma 13) for each value of $\bar{k}$. Since there are $k^*(n_1) = O(\sqrt{n})$ possible values for $\bar{k}$, the entire calculation is done in $O(\sqrt{n}\log n)$ operations. Step 5 involves choosing the smaller of two values, each of which is calculated in $O(\log n)$ operations (Lemma 13). In step 6 the algorithm moves from left to right across the current row. Since the total maximum movement in this direction is $M = O(\sqrt{n})$ (summing the movement over all rows), and since each evaluation of $C_k(i,j)$ requires $O(\log n)$ operations, the number of operations devoted to this step throughout the algorithm is $O(\sqrt{n}\log n)$. Step 9 requires $O(\sqrt{n})$ calculations of $C_k(0,0)$, each taking $O(\log n)$ operations, for a total of $O(\sqrt{n}\log n)$ operations. All the other steps require a constant number of operations. Thus the running time of the entire procedure is $O(\sqrt{n}\log n)$. ■

The algorithm determines the cost of the optimal schedule (C^*), the number of heavy batches (k^*), and the number of heavy and light jobs in the mixed batch (i^* and j^*, respectively). To determine the actual batch sizes, we can use the algorithm in [CNY89] to find an optimal schedule for the $n_2 - j$ light jobs in light batches, and a modification of this algorithm to find an optimal k-batching for the heavy jobs in heavy batches. Both of these steps can be done in $O(\sqrt{n})$ time, so the running time of our algorithm is not affected.

7 Conclusion

We have shown that solving the 2-weight batching problem can be done in time that is sublinear in the number of jobs. The obvious question posed by this result is: can it be extended for any fixed number of job types? The algorithm of [AB93] solves the more general problem in $O(n)$ time. Perhaps this bound can be improved upon when the number of distinct job types is fixed.

References

[AB93] S. Albers and P. Brucker. The complexity of one-machine batching problems. *Discrete Applied Mathematics*, 47:87–107, 1993.

[AKM+87] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[CNY89] E. Coffman Jr., A. Nozari, and M. Yannakakis. Optimal scheduling of products with two subassemblies on a single machine. *Operations Research*, 37:426–436, 1989.

[CYMS90] E. Coffman Jr., M. Yannakakis, M. Magazine, and C. Santos. Batch sizing and sequencing on a single machine. *Annals of Operations Research*, 26:135–147, 1990.

[DKR87] G. Dobson, U.S. Karmarkar, and J.L. Rummel. Batching to minimize flow times on one machine. *Management Science*, 33:784–799, 1987.

[GB86] P. Gise and R. Blanchard. *Modern Semiconductor Fabrication Technology.* Prentice-Hall, Englewood Cliffs, N.J., 1986.

[HL94] D. Hochbaum and D. Landy. Scheduling with batching: Minimizing the weighted number of tardy jobs. *Operations Research Letters*, 16:79–86, 1994.

[NS88] D. Naddef and C. Santos. One-pass batching algorithms for the one machine problem. *Discrete Appl. Math.*, 21:133–145, 1988.

[Sha92] D. Shallcross. A polynomial algorithm for a one machine batching problem. *Operations Research Letters*, 11:213–218, 1992.

[Sze83] S. Sze. *VLSI Technology.* McGraw-Hill, New York, 1983.

[ZC84] L. Ya. Zamanskii and V.L. Cherkasskii. A formula for finding the number of integer points under a straight line and its application. *(in Russian) Ekonom. i Mat. Metody*, 20:1132–1138, 1984.