On Scheduling Problems with Applications to Packet-Switched Optical WDM Networks*

Evripidis Bampis
LaMI, CNRS EP738
Université d'Evry Val d'Essonne
Boulevard F. Mitterrand
91025 Evry Cedex
France

George N. Rouskas
Department of Computer Science
Box 7534
North Carolina State University
Raleigh, NC 27695-7534
USA

ABSTRACT

We consider a scheduling problem, which we call the *Scheduling and Wavelength Assignment* (SWA) problem, arising in optical networks that are based on the Wavelength Division Multiplexing (WDM) technology. We prove that the SWA problem is $\mathcal{NP}$-hard for both the preemptive and the non-preemptive cases. Furthermore, we propose two efficient approximation algorithms. The first is for the preemptive case and it is based on a natural decomposition of the problem to the classical multiprocessor scheduling and open-shop problems. For the non-preemptive case, we prove that a naive implementation of list scheduling produces a schedule that can be m times far from the optimum, where m is the number of processors (equivalently, WDM channels). Finally, we give a more refined version of list scheduling and we prove it to be a 2-approximation algorithm for both the off-line and the on-line contexts.

1. INTRODUCTION

The spectacular growth in data traffic and the surging demand for diverse services has led to a dramatic increase in demand for data transmission capacity. Recent advances in wavelength division multiplexing (WDM) technology[4,8] are expected to provide solutions to this challenge. WDM supports multiple simultaneous channels, each on a different wavelength, on a single fiber. WDM systems operating at aggregate rates exceeding one Terabit per second have been demonstrated, while systems supporting rates of tens of Gigabits per second are becoming commercially available.

As future networks based on WDM technology are developed to support data traffic and the Internet, they must be designed and optimized for that purpose. In particular, a number of new and challenging problems arise in the area of scheduling data packets over multiple wavelengths, both in a local area environment[1,18] and in a backbone network consisting of IP routers.[16] In this paper we consider a scheduling problem with applications to packet-switched optical WDM networks, and we prove that it is $\mathcal{NP}$-hard for both the preemptive and the non-preemptive cases. We then present two efficient approximation algorithms for this problem. For the preemptive case, the approximation algorithm is based on a natural decomposition of the problem into the classical multiprocessor scheduling and open-shop problems. For the non-preemptive case, we develop two list scheduling algorithms, the second of which is a 2-approximation algorithm for both the on-line and off-line contexts.

The paper is organized as follows. In the next section we define the scheduling problem under study, and we motivate it by describing its relationship to packet scheduling in WDM optical networks. In Section 3 we present an off-line approximation algorithm for preemptive scheduling, and in Section 4 we present both off-line and on-line algorithms for the non-preemptive case. We conclude the paper in Section 5.

*This work was performed while G. N. Rouskas was visiting the Université d'Evry Val d'Essone. The work of G. N. Rouskas was supported by the Université d'Evry Val d'Essonne, by the National Science Foundation under grant NCR-9701113 (CAREER), and by the Defense Advanced Research Projects Agency under grant F-30602-00-C-0034.

2. PROBLEM DEFINITION AND APPLICATIONS

2.1. The Scheduling and Wavelength Assignment (SWA) Problem

We consider a set $\mathcal{M}$ of m processors and a set $\mathcal{J}$ of $n, n > m$ jobs. Each job $J_d \in \mathcal{J}$ is defined as a set of n operations, $J_d = \{O_{1d}, O_{2d}, \cdots, O_{nd}\}$, and p_{sd} denotes the processing time of operation O_{sd}. The objective is to schedule the n jobs on the m processors so as to minimize the *makespan*, or maximum finish time of the schedule, subject to the following constraints:

C1. All operations of job J_d are executed on the same processor.

C2. The operations O_{sd} and $O_{sd'}$ cannot be simultaneously executed, for all $s, d \neq d'$.

C3. A processor may execute at most one operation at any time instant.

C1 and **C2** define a set of *compatibility* constraints among the different operations. Specifically, two operations O_{sd} and $O_{s'd'}$ are said to be *incompatible* if either $s = s'$ or $d = d'$; otherwise the operations are *compatible*. Incompatible operations cannot be executed simultaneously. Constraint **C1** also implies that once an operation of job J_d has been executed on a processor x, then all operations $O_{sd}, s = 1, \cdots, n$, must be executed on the same processor x. However, the processor on which the operations of a job J_d are executed is not known in advance, rather, it is determined as part of the solution to the scheduling problem. Furthermore, the operations of a job J_d can be executed on processor x in any order.

The scheduling problem defined above can be logically decomposed into two sub-problems. Because of constraint **C1**, the first sub-problem is to assign each job J_d to a processor x, meaning that all operations of J_d will be executed on x. Given this assignment of jobs to processors, the second sub-problem is to schedule the operations on their assigned processors so as to minimize the makespan, while also satisfying the compatibility constraints **C2** and the processor constraint **C3**. This decomposition leads to a natural way of solving the scheduling problem by applying existing algorithms to each sub-problem in isolation, as discussed later. However, we will also show that it is possible to design algorithms to simultaneously solve both sub-problems, and these algorithms are more efficient than the two-step approach described above.

We will refer to this type of scheduling problem as the *Scheduling and Wavelength Assignment (SWA)* problem, since it arises naturally in packet-switched optical WDM networks, as we explain in Section 2.3. We have also chosen to use s and d as subscripts for the operations to reflect the fact that, in the network settings described in Section 2.3, operation O_{sd} corresponds to the amount of traffic to be transmitted from a source s to a destination d in the network.

We now introduce notation that will be useful in later sections. Let T_{seq} denote the sum of the computation times of all operations, let D_{max} be the maximum amount of computation time required by any job, and let S_{max} denote the maximum amount of computation time associated with any source s:

$$T_{seq} = \sum_{s=1}^{n} \sum_{d=1}^{n} p_{sd} \tag{1}$$

$$D_{max} = \max_{d=1,\cdots,n} \left\{ \sum_{s=1}^{n} p_{sd} \right\} \tag{2}$$

$$S_{max} = \max_{s=1,\cdots,n} \left\{ \sum_{d=1}^{n} p_{sd} \right\} \tag{3}$$

Let C_{max}^{opt} denote the optimal makespan. We obviously have that:

$$C_{max}^{opt} \geq \frac{T_{seq}}{m} \tag{4}$$

$$C_{max}^{opt} \geq S_{max} \tag{5}$$

$$C_{max}^{opt} \geq D_{max} \tag{6}$$

2.2. Related Classical Scheduling Problems

There are two classical scheduling problems that are closely related to the *SWA* problem we consider in this work. The first one is the *multiprocessor scheduling problem*,[7,12] in which we have a set of n tasks that must be scheduled on m machines in such a way that the makespan is minimized. As in the *SWA* problem, the tasks can be executed on any machine and their processing times are not machine-dependent. It is well-known that the multiprocessor scheduling problem is $\mathcal{NP}$-hard,[7] and that any list-scheduling algorithm is a 2-approximation algorithm.[11] Note that this result is very important, since it remains valid in an *on-line* context. Of course, for the *off-line* case, there exist algorithms which provide significantly better performance guarantees. These include the LPT (Largest Processing Time) algorithm, which is a 4/3-approximation algorithm,[11] and the MULTI-FIT algorithm, which guarantees a relative performance of 1.2.[6] In addition, there exists a polynomial time approximation scheme (PTAS) for the multiprocessor scheduling problem that is due to Hochbaum and Schmoys.[13] The main difference between the multiprocessor scheduling problem and the *SWA* problem is that in the former, the n tasks are assumed to be pair-wise independent, and thus, any two tasks may be scheduled simultaneously on different processors. In the *SWA* problem, on the other hand, there is a set of compatibility constraints among the operations, as defined by constraints **C1** and **C2**, which prevent the simultaneous execution of certain operations.

The second related problem is the *open-shop problem* where we have a set of n jobs with m operations each,[12,10,15,9] such that the l-th operation, $l = 1, \cdots, m$, of a job must be processed on machine l. Similar to the *SWA* problem, under open-shop scheduling, the operations of any given job may be processed in any order. The preemptive version of the open-shop problem is solvable in polynomial time.[10] A polynomial-time algorithm also exists for the non-preemptive open-shop problem with $m = 2$ processors.[10] However, the general non-preemptive open-shop problem is known to be $\mathcal{NP}$-hard, and it has been shown that any list-scheduling algorithm is a 2-approximation algorithm.[14] Here also, it is interesting to point out that this result holds in the on-line context.

2.3. Applications

We now describe several network environments where the *SWA* problem arises.

Broadcast WDM optical networks.[1] Consider a WDM optical network with n nodes interconnected via a broadcast star that supports $m < n$ distinct wavelengths.[17] Nodes communicate by exchanging fixed-length packets, and the time it takes to transmit a packet is taken as the unit of time. Since there are fewer wavelengths than nodes, packet transmissions by several nodes may share a single wavelength, and the problem of scheduling these packet transmissions arises. At the same time, an important objective in such a network is load balancing across the different wavelengths, since it has been shown that network performance deteriorates significantly if the traffic load concentrates on a few wavelengths.[20,19,2,3]

Let us assume that the long-term traffic requirements of the nodes are known, let O_{sd} denote the operation of transmitting packets from node s to node d, and let p_{sd} be the number of packets that need to be transmitted between these two nodes in the network. Let us also assume that the nodes are equipped with fast-tunable transmitters, so that no cost is incurred when a transmitter switches from one wavelength to another, but that receivers are fixed-tuned to a certain wavelength (these are tunability characteristics of nodes in the `Helios` DARPA NGI project[1]). Let us also consider the operations of transmitting packets to a single receiver d as a job J_d. Then, scheduling the packet transmissions over the m wavelengths is equivalent to the *preemptive SWA* problem, since the transmission of the p_{sd} packets of operation O_{sd} can be preempted (at the end of any packet) and continued at a later time. Note that, minimizing the makespan for this problem implies balancing the traffic across the various wavelengths by properly assigning the fixed-tuned receivers to wavelengths.

WDM IP routers employing multi-protocol label switching (MPLS). Consider a backbone network of n high-speed IP routers interconnected by fiber lines each supporting m distinct wavelengths. The routers employ MPLS,[5] whereby each IP packet carries a special label used by the routers to forward the packet to its destination backbone router, avoiding computationally expensive IP table lookups. To avoid packet reordering at the destination router, packets for a given destination d (i.e., those carrying the same label) must be sent over the same wavelength in a first-come, first-served order. Since IP packets are of variable length, a *non-preemptive SWA* problem arises in this case.

Grooming packet traffic over WDM SONET/SDH rings. In this scenario, a node on a SONET/SDH ring must transmit ATM fixed-size cells or IP variable-size packets over the m wavelengths in the ring. Assuming that

there are n destination nodes and that traffic to each destination must be carried on the same wavelength (groomed), the traffic scheduling problem is equivalent to the *SWA* problem.

While all the applications described in this section are in packet scheduling in WDM networks, for the rest of the paper we will use terminology from the multiprocessor scheduling literature. The reader should keep in mind, however, that the terms "processor," "computation time," and "job" correspond to "wavelength," "transmission time," and "destination," respectively, in the network environment.

3. PREEMPTIVE SCHEDULING

Let us first assume that the operations O_{sd} of any job J_d are preemptable, i.e., there is no cost in preempting an operation and resuming it later on the *same* processor (refer to constraint **C1**). We have the following result.

LEMMA 3.1. *The preemptive SWA problem is $\mathcal{NP}$-complete.*

Proof. Consider an instance of the *SWA* problem with $m = 2$ processors and n jobs. Each job J_d, $d = 1, \cdots, n$, of the instance consists of a single operation O_{dd} with non-zero processing time p_{dd} (i.e., all other operations $O_{sd}, s \neq d, s = 1, \cdots, n$, of job J_d are such that $p_{sd} = 0$). Since an operation may not change processor after a preemption, this special case of the preemptive *SWA* is equivalent to the *PARTITION* problem,[7] which is $\mathcal{NP}$-complete. ■

An approximation algorithm for the preemptive *SWA* problem may now be obtained by considering the problem decomposition described previously. The approximation algorithm consists of two steps. The first step assigns jobs to processors in a way that attempts to balance the amount of work (i.e., the total processing time) assigned to each processor. The second step is to apply existing approximation algorithms to the resulting open-shop problem.

Algorithm DA (Decomposition algorithm)

S1. Let P_d be the total processing time required by job J_d, $P_d = \sum_{s=1}^{n} p_{sd}$. Considering each job J_d as an independent task with processing time equal to P_d, run an approximation algorithm for the resulting multiprocessor problem to assign each job J_d to a processor.

S2. Let $\mathcal{J}_x$ denote the set of jobs assigned to processor x at the end of Step **S1**. For each processor x, create new operations O'_{sx} with processing time $p'_{sx} = \sum_{d|J_d \in \mathcal{J}_x} p_{sd}$. Now, the original *SWA* problem has been transformed to a preemptive open-shop scheduling problem with m processors and operations O'_{sx}. Run the Gonzalez and Sahni algorithm[10] to obtain an optimal schedule for the new open-shop problem.

LEMMA 3.2. *Algorithm* **DA** *is an approximation algorithm for the preemptive SWA problem.*

Proof. Correctness. Step **S1** assigns each job to a certain processor, thus, all operations of a job will be executed on the same processor, satisfying constraint **C1**. In Step **S2**, a preemptive open-shop problem is constructed, such that the processing time of an operation O_{sx} for a source s on processor x is the sum of the processing times of operations O_{sd} for which job J_d has been assigned to processor x. Consider two jobs J_d and $J_{d'}$, $d \neq d'$. If the jobs have been assigned to two different processors, the Gonzalez and Sahni algorithm ensures that constraint **C2** is satisfied. If the jobs have been assigned to the same processor, constraint **C2** is also satisfied since, whenever operation O_{sx} is being processed, at most one of operations O_{sd} or $O_{sd'}$ (which are part of O_{sx}) is processed, but not both. Finally, the operation of the Gonzalez and Sahni algorithm does not violate constraint **C3**.

Approximation claim. Consider the preemptive open-shop problem in Step **S2**, and let D'_{max} (respectively, S'_{max}) denote the maximum computation time associated with any processor (resp., source). By construction of the open-shop problem we have that:

$$S'_{max} = S_{max} \tag{7}$$

where the quantity S_{max} for the original *SWA* problem is defined in (3). Note, now, that an α-approximation algorithm is used in Step **S1** for the multiprocessor scheduling problem. This approximation algorithm respects constraint **C1** only, therefore the optimal for the multiprocessor scheduling problem is at most equal to the optimal for the *SWA* problem. Thus,

$$D'_{max} = \alpha \, C^{opt}_{max} \tag{8}$$

where C^{opt}_{max} refers to the *SWA* problem. Since the Gonzalez and Sahni algorithm for the preemptive open-shop scheduling problem is optimal,[10] the schedule produced by the decomposition algorithm has length

$$C^{DA}_{max} \;=\; \max\{D'_{max}, S'_{max}\} \;\leq\; \alpha\, C^{opt}_{max} \tag{9}$$

where the last inequality follows from (6). ■

The above lemma implies that, if we use LPT (respectively, MULTI-FIT) in Step **S1**, then the decomposition algorithm is a 4/3-approximation (resp., 1.2-approximation) algorithm. On the other hand, the decomposition algorithm becomes a PTAS if the PTAS developed by Hochbaum and Schmoys for the multiprocessor scheduling problem is used in Step **S1** to obtain the assignment of jobs to processors.

4. NON-PREEMPTIVE SCHEDULING

Let us now consider the non-preemptive *SWA* problem, whereby, once an operation O_{sd} has started processing on a certain processor x, it must be completed before the processor can start executing another operation. The following lemma proves that the non-preemptive version of the problem is also $\mathcal{NP}$-complete.

LEMMA 4.1. *The non-preemptive SWA problem is NP-complete.*

Proof. Consider an instance of the non-preemptive *SWA* problem with m processors and n jobs. Each job J_d, $d = 1, \cdots, n$, of the instance consists of a single operation O_{dd} with non-zero processing time p_{dd} (i.e., all other operations $O_{sd}, s \neq d, s = 1, \cdots, n$, of job J_d are such that $p_{sd} = 0$). This instance consists of n independent tasks (the n operations O_{dd}) and $m < n$ processors. Thus, the well-known $\mathcal{NP}$-complete multiprocessor scheduling problem[7,12] is a special case of the non-preemptive *SWA* problem. ■

4.1. Algorithm Based on Problem Decomposition

An algorithm based on a problem decomposition, similar to the one developed for the preemptive case in Section 3, can also be applied to this problem. However, non-preemptive open-shop scheduling is an $\mathcal{NP}$-complete problem,[10] but it is known that any list scheduling algorithm is a 2-approximation algorithm for this problem.[14] Therefore, instead of the Gonzalez and Sahni optimal algorithm for preemptive open-shop scheduling, a 2-approximation list scheduling algorithm for the non-preemptive open-shop scheduling problem is applied in Step **S2** of the decomposition algorithm **DA** in Section 3. We now have the following lemma.

LEMMA 4.2. *If an α-approximation algorithm is applied to the corresponding multiprocessor scheduling problem in Step* **S1** *of the decomposition, then the decomposition algorithm* **DA** *is a 2α-approximation algorithm for the non-preemptive SWA problem.*

Proof. Correctness. The proof of correctness is similar to the one given for the preemptive *SWA* problem in Lemma 3.2.

Approximation Claim. If an α-approximation algorithm is used in Step **S1** for the multiprocessor scheduling problem, by construction of the open-shop problem we have that (refer also to the proof of Lemma 3.2):

$$D'_{max} \;=\; \alpha\, C^{opt}_{max} \tag{10}$$

$$S'_{max} \;=\; S_{max} \tag{11}$$

where the quantity S_{max} for the original *SWA* problem is defined in (3). Since list scheduling is a 2-approximation algorithm for the non-preemptive open-shop scheduling problem,[10] the schedule produced by the decomposition algorithm has length

$$C^{DA}_{max} \;\leq\; 2\ \max\{D'_{max}, S'_{max}\} \;\leq\; 2\alpha\, C^{opt}_{max} \tag{12}$$

where the last inequality follows from (5). ■

We now note that, while the decomposition algorithm **DA** will correctly schedule the operations of the non-preemptive *SWA* problem (i.e., the resulting schedule will satisfy constraints **C1**, **C2**, and **C3**), it may actually do more than is needed to satisfy the non-preemption requirement. Consider jobs J_d and $J_{d'}$, $d \neq d'$, if they exist, that

have been assigned to the same processor x. In Step **S2** of the algorithm, a new operation $O_{sx}, s = 1, \cdots, n$, is created for the new open-shop problem, with a processing time equal to the sum of the processing times of operations O_{sd} and $O_{sd'}$ (and, possibly, the operations of other jobs assigned to the same processor) of the original *SWA* problem. The list scheduling algorithm in Step **S2** applied to the open-shop scheduling problem will ensure that the operation O_{sx} will not be preempted, while the requirement for the original *SWA* problem is simply that operations O_{sd} and $O_{sd'}$ not be preempted.

4.2. Algorithms Based on List Scheduling

We now present list scheduling algorithms for the non-preemptive *SWA* problem. We first show that a naive implementation of list scheduling can produce schedules that are a factor of m from the optimal schedule. We then describe a more refined version of list schedule that yields a 2-approximation algorithm for the non-preemptive *SWA* problem.

We will need the following definitions in our discussion. A job J_d is said to be *assigned* to processor x at time t if:

1. no operation $O_{sd}, s = 1, \cdots, n$, has been executed (fully or partially) on any processor before time t, and
2. processor x is idle at time t and starts processing some operation $O_{sd}, s = 1, \cdots, n$, at that time.

Because of the problem constraints, once a job J_d is assigned to processor x, all operations $O_{sd}, s = 1, \cdots, n$ must be executed on x. An operation O_{sd} is called *schedulable* on processor x at time t if:

1. job J_d has not been assigned to a different processor y at some time $t' < t$, and
2. an incompatible operation $O_{s'd'}$(i.e., such that $s = s'$ or $d = d'$) is not being executed by another processor y at time t.

When a processor x becomes idle at some time t, it can only start processing an operation that is schedulable at time t.

4.2.1. List Scheduling, version 1

Algorithm LS1 (List scheduling, version 1)
All the tasks O_{sd}, $s, d = 1, \cdots, n$, are initially arranged in an arbitrary list L. Consider a processor x which becomes idle at time t. If list L is empty, the algorithm terminates. Otherwise, processor x starts processing the first schedulable operation O_{sd} in L, and the operation is removed from the list. If no schedulable operation is found in L, processor x remains idle until time $t' > t$ at which another processor y that was busy at time t completes its operation. At time t', processors x and y each scan list L to select a new schedulable operation to process. (Ties are broken arbitrarily).

LEMMA 4.3. *Algorithm* **LS1** *is an m-approximation algorithm for the non-preemptive SWA problem.*

Proof. Correctness. By construction, the algorithm ensures that a processor may execute at most one operation at any time instant, thus satisfying constraint **C3**. The definition of a "schedulable" operation above, and the requirement that a processor, upon becoming idle, selects a schedulable operation for execution, also ensure that the algorithm will never violate constraints **C1** or **C2**.

Approximation Claim. Let C_{max}^{LS1} denote the makespan of a schedule produced by algorithm **LS1**, and let T_{idle} denote the total idle time (over all m processors) in this schedule. We first observe that at no point in time can all m processors be idle in a schedule produced by using algorithm **LS1**, thus, $T_{idle} \leq (m-1)C_{max}^{LS1}$. Because of (4), we obtain:

$$\begin{aligned} C_{max}^{LS1} &\leq \frac{T_{idle} + T_{seq}}{m} \leq \frac{m-1}{m} C_{max}^{LS1} + C_{max}^{opt} \\ &\Rightarrow C_{max}^{LS1} \leq m C_{max}^{opt}. \end{aligned} \tag{13}$$

■

	J_1	J_2	J_3	$\cdots$	J_m	J_{m+1}	J_{m+2}	$\cdots$	J_{2m}
1	$M - \frac{m+1}{m}$	0	0	$\cdots$	0	0	0	$\cdots$	0
2	$\frac{1}{m}$	$M - \frac{m+1}{m}$	0	$\cdots$	0	0	0	$\cdots$	0
3	0	$\frac{1}{m}$	$M - \frac{m+1}{m}$	$\cdots$	0	0	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	0	0	0	$\cdots$	$M - \frac{m+1}{m}$	0	0	$\cdots$	0
$m+1$	0	0	0	$\cdots$	$\frac{1}{m}$	$\frac{1}{m}$	$\frac{1}{m}$	$\cdots$	$\frac{1}{m}$
$m+2$	0	0	0	$\cdots$	0	$\frac{1}{m}$	$\frac{1}{m}$	$\cdots$	$\frac{1}{m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$2m$	0	0	0	$\cdots$	0	$\frac{1}{m}$	$\frac{1}{m}$	$\cdots$	$\frac{1}{m}$

Figure 1. Problem instance that asymptotically achieves the upper bound of Lemma 4.3

The following problem instance shows that, asymptotically, the bound of the lemma is a *tight one*. The instance consists of a number m of processors and a number $2m$ of jobs whose operations are shown in Figure 1. The first m jobs consist of two operations, one long operation of length $M - \frac{m+1}{m}$ and one short operation of length $\frac{1}{m}$. The last m jobs consist of exactly m short operations each, of length $\frac{1}{m}$. Let $M = m(m-1)$. The optimal schedule, shown in Figure 2(a), is such that processor $x, x = 1, \cdots, m$, executes exactly two jobs, say jobs x and $x+m$. This schedule has length $C_{max}^{opt} = M = m(m-1)$. On the other hand, it is easy to see that under algorithm **LS1**, it is possible for one processor, say processor 1, to be assigned jobs 1 through $m+1$, while processor $x, x = 2, \cdots, m$, is assigned only one of the last $m-1$ jobs. Such a schedule is illustrated in Figure 2(b), and has length

$$C_{max}^{LS1} \;=\; 2 \;+\; m\left(M - \frac{m+1}{m}\right) \;=\; mM - (m-1) = \;\; m^2(m-1) - (m-1) \tag{14}$$

Then,

$$\frac{C_{max}^{LS1}}{C_{max}^{opt}} \;=\; \frac{m^2(m-1) - (m-1)}{m(m-1)} \;=\; m \;-\; \frac{1}{m}. \tag{15}$$

4.2.2. List Scheduling, version 2

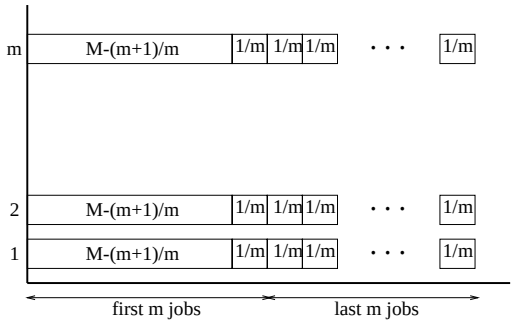
We now present a different version of list scheduling which yields a 2-approximation algorithm for the non-preemptive problem.

Algorithm LS2 (List scheduling, version 2)
The n jobs $J_d, d = 1, \cdots, n$, are initially arranged in a list L. The operations of each job J_d are also arranged in a list L_d. A list l_x is also associated with each processor x, and it is initialized to the empty list. Consider a processor x that becomes idle at time t, and let O_{sd} be the operation that was just completed by the processor. Processor x selects an operation to process next by taking the following three steps in the order presented:

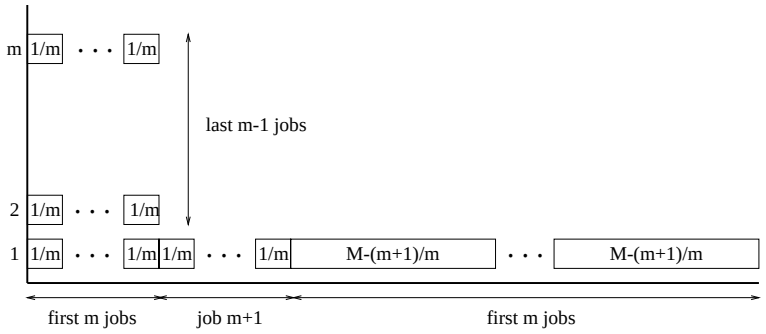
1. if there exists an operation $O_{sd'}, d' \neq d$, in the processor's list l_x, it is removed from the list and x starts processing this operation at time t (note that operation $O_{sd'}$ is schedulable on x at time t, since scheduling it does not violate constraint **C2**).

2. if no operation $O_{sd'}$ is found in Step 1, x starts processing any other schedulable operation in its list l_x, and the operation is removed from the list.

3. If no operation is found in the first two steps, list L is scanned. If L is empty, the algorithm terminates. Otherwise, let $J_{d'}$ be the first job in L with an operation that is schedulable on processor x at time t, if one exists. Job $J_{d'}$ is removed from L, and its list of operations $L_{d'}$ is appended to processor x's list l_x. Processor x starts processing the first schedulable operation in its new list l_x, and the operation is removed from l_x.



(a) Optimal schedule



(b) Schedule under algorithm LS1

Figure 2. (a)Optimal schedule and (b) worst case schedule produced by algorithm **LS1** for the instance shown in Figure 1

If no schedulable operation is found at the end of the third step, processor x remains idle until time $t' > t$ at which another processor y that was busy at time t completes its operation. At time t', processors x and y each repeat the above procedure to select a new schedulable operation to process. (Ties are broken arbitrarily.)

LEMMA 4.4. *Algorithm* **LS2** *is a 2-approximation algorithm for instances of the non-preemptive* SWA *problem for which no operation has zero processing time (i.e.,* $p_{sd} > 0 \; \forall \; s, d$*).*

Proof. Correctness. By construction, the algorithm satisfies constraint **C3**. The fact that when the first operation of a job J_d is schedulable on a processor x, the job is removed from list L and is appended to processor x's list, ensures that constraint **C1** is not violated. Finally, the requirement that a processor, upon becoming idle, selects a schedulable operation for execution, guarantees that the resulting schedule will satisfy constraint **C2**.

Approximation Claim. Consider a schedule $\mathcal{S}$ built using algorithm **LS2**, and let J_d be the job that is the last to be assigned to a processor (i.e., all other jobs have been assigned to processors before J_d). Let t be the time when job J_d is assigned to a processor. We claim that no processor is idle before time t. To see that the claim is true, assume that a processor y became idle at time $t' < t$. Note now that at most $m-1$ different operations were being processed at time t', one at each of the other processors. By assumption, J_d includes $n > m-1$ non-zero operations, and $n - m + 1$ of these operations are schedulable on y at time t'. Thus, under algorithm **LS2**, job J_d should have been assigned to y at that point, contradiction.

At time t, all jobs have been assigned to processors. Let $\mathcal{P} = \{\mathcal{J}_1, \cdots, \mathcal{J}_m\}$ be the partition of the job set $\mathcal{J}$ such that $\mathcal{J}_x$ is the subset of jobs that have been assigned by the algorithm to processor $x, x = 1, \cdots, m$. Let $p_{sd}(t)$ be the amount of processing that operation O_{sd} has received up to time t. Consider a new scheduling problem with m processors and n jobs J'_s, where each job is a set of operations, $J'_s = \{O'_{s1}, \cdots, O'_{sm}\}$, and the processing time of operation O'_{sx} is given by:

$$p'_{sx} \quad = \quad \sum_{d \in \mathcal{J}_x} (p_{sd} - p_{sd}(t)) \, . \tag{16}$$

This new problem is an instance of open-shop scheduling[10] with m processors and n jobs J'_s, where each job has a single operation O'_{sx} to be executed on each processor x. We now observe that algorithm **LS2** is also a list scheduling algorithm for this open-shop problem. To see this, consider a processor x that becomes idle at time $t' > t$. Since the last job was assigned to some processor at time t, no new jobs will be assigned to x at or after time t'. Let O_{sd} be the operation that was just completed by x at time t'. Because of the first step in selecting a new job under algorithm **LS2**, if there is another job $O_{sd'}$ in list l_x, x will start processing $O_{sd'}$ at time t'. When x becomes idle again, the same selection process will be repeated until all operations $O_{sd'}$ with the same source s in list l_x are completed (note that each of these operations are schedulable on x at the instant the previous one is completed). Thus, these operations $O_{sd'}$ will be executed back-to-back, in some order, without interruption, and no operations with the same source will be added to list l_x after time t'. But all these operations $O_{sd'}$ are part of the operation O'_{sx} of job J'_s in the open shop problem. Consequently, operation O'_{sx} will be executed without preemption on processor x by algorithm **LS2**. Therefore, algorithm **LS2** is a list scheduling algorithm for the open-shop problem.

Let D'_{max} (respectively, S'_{max}) denote the maximum computation time associated with any processor (resp., source) in the open-shop problem. Since there is no idle time in the schedule constructed by algorithm **LS2** until time t, we have that (note that we give the problem to which the various parameters are related in parentheses, *OS* for open-shop, and *SWA* for the original *SWA* problem):

$$C^{opt}_{max}(SWA) \;\geq\; t \;+\; \max\{D'_{max}, S'_{max}\} \tag{17}$$

Since **LS2** is a list scheduling algorithm for the open-shop problem, we also have that:

$$C^{LS2}_{max}(OS) \;\leq\; 2\ \max\{D'_{max}, S'_{max}\} \;\leq\; 2\ C^{opt}_{max}(OS) \tag{18}$$

Finally, we get the desired result as follows:

$$\begin{aligned} C^{LS2}_{max}(SWA) &= t \;+\; C^{LS2}_{max}(OS) \;\leq\; t \;+\; 2\ \max\{D'_{max}, S'_{max}\} \\ &\leq t \;+\; C^{opt}_{max}(SWA) \;+\; \max\{D'_{max}, S'_{max}\} \;\leq\; 2\ C^{opt}_{max}(SWA) \end{aligned} \tag{19}$$

■

Now assume that some of the operations in the original non-preemptive *SWA* problem are zero. Without loss of generality, assume that each job J_d consists of at least one operation of non-zero length (otherwise, we can remove this job reducing the problem into an equivalent one with m processors and $n-1$ jobs). We can obtain a schedule that is at most twice the optimum one for this problem by taking the following three steps. First, replace all zero-length operations with ones of length equal to $\epsilon > 0$. Then, run algorithm **LS2** on this new instance of the *SWA* problem for which no operation has zero processing time. Finally, remove from the schedule obtained by algorithm **LS2** all operations that are of zero processing time in the original problem.

To see that this schedule is at most twice the optimum for the original *SWA* problem (the one with some operation of zero length), we observe that at most $n(n-1)$ operations are added to the problem instance in the first step. Now note that a schedule for the new problem can be obtained by sequentially processing the $n(n-1)$ operations on one processor after the end of the optimal schedule for the original problem, thus:

$$C^{opt}_{max}(new) \;\leq\; C^{opt}_{max}(original) \;+\; n(n-1)\ \epsilon \tag{20}$$

The optimum schedule for the new problem is at most equal to this schedule. Since at the third step of the algorithm we remove some operations from the schedule produced by **LS2**, we have that:

$$\begin{aligned} C^{LS2}_{max}(original) &\leq C^{LS2}_{max}(new) \;\leq\; 2\ C^{opt}_{max}(new) \\ &\leq 2\ C^{opt}_{max}(original) \;+\; 2\ n(n-1)\ \epsilon \end{aligned} \tag{21}$$

By selecting an appropriate value for ϵ, we see that the schedule obtained in this way for the original problem is at most twice the optimal schedule plus a constant.

Finally, we note that algorithm **LS2** is valid in an on-line context, where the jobs are not known in advance but appear one after the other.

5. CONCLUDING REMARKS

We considered the *Scheduling and Wavelength Assignment* (SWA) problem which has applications in packet-switched optical WDM networks. We proved that the SWA problem is $\mathcal{NP}$-hard for both the preemptive and the non-preemptive cases. Furthermore, we propose two efficient approximation algorithms. For the preemptive case, we described an efficient approximation algorithm based on a natural decomposition of the problem. For the non-preemptive case, we presented two list scheduling algorithms, the second of which is a 2-approximation algorithm for both the off-line and the on-line contexts.

REFERENCES

1. The NGI Helios project. http://www.anr.mcnc.org/projects/Helios/Helios.html.
2. I. Baldine and G. N. Rouskas. Dynamic load balancing in broadcast WDM networks with tuning latencies. In *Proceedings of INFOCOM '98*, pages 78–85. IEEE, March 1998.
3. I. Baldine and G. N. Rouskas. Traffic adaptive wdm networks: A study of reconfiguration issues. *IEEE/OSA Journal of Lightwave Technology*, 19(4):433–455, April 2001.
4. G.-K. Chung, K.-I. Sato, and D. K. Hunter (Eds.). Special issue on optical networks. *Journal of Lightwave Technology*, 18(12), December 2000.
5. G-S. Kuo (Ed.). Special issue on multiprotocol label switching. *IEEE Communications Magazine*, 37(12), December 1999.
6. D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computation*, 13(1):170–181, 1984.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., New York, 1979.
8. O. Gerstel, B. Li, A. McGuire, G. N. Rouskas, K. Sivalingam, and Z. Zhang (Eds.). Special issue on protocols and architectures for next generation optical WDM networks. *IEEE Journal Selected Areas in Communications*, 18(10), October 2000.
9. T. Gonzalez. A note on open-shop preepmptive schedules. *IEEE Transactions on Computers*, C-28:782–786, 1979.
10. T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4):665–679, Oct 1976.
11. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, Mar 1969.
12. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnoy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, Jan 1979.
13. D. S. Hochbaum and D. B. Schmoys. Using dual approximation algorithms for scheduling problems. *Journal of ACM*, 34(1):144–162, Jan 1987.
14. T. Fiala I. Bárány. Tobbgepes, utemezesi problemak kozel optimalis megoldasa. *Szigma-Mat.-Kozgazdasagi Folyoirat*, 15:177–191, 1982.
15. E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of ACM*, 25:612–619, 1978.
16. E. Modiano. WDM-based packet networks networks. *IEEE Communications*, 37(3):130–135, March 1999.
17. R. Ramaswami and K. N. Sivarajan. *Optical Networks*. Morgan Kaufmann Publishers, San Francisco, California, 1998.
18. G. N. Rouskas and V. Sivaraman. Packet scheduling in broadcast WDM networks with arbitrary transceiver tuning latencies. *IEEE/ACM Transactions on Networking*, 5(3):359–370, June 1997.
19. V. Sivaraman and G. N. Rouskas. HiPeR-ℓ: A High Performance Reservation protocol with ℓook-ahead for broadcast WDM networks. In *Proceedings of INFOCOM '97*, pages 1272–1279. IEEE, April 1997.
20. V. Sivaraman and G. N. Rouskas. A reservation protocol for broadcast WDM networks and stability analysis. *Computer Networks*, 32(2):211–227, February 2000.