Spectrum Assignment in Optical Networks: A Multiprocessor Scheduling Perspective

Sahar Talebi, Evripidis Bampis, Giorgio Lucarelli, Iyad Katib, and George N. Rouskas

***Abstract*—The routing and spectrum assignment problem has emerged as the key design and control problem in elastic optical networks. In this work, we show that the spectrum assignment (SA) problem in mesh networks transforms to the problem of scheduling multiprocessor tasks on dedicated processors. Based on this new perspective, we show that the SA problem in chain (linear) networks is NP-hard for four or more links, but is solvable in polynomial time for three links. We also develop new constant-ratio approximation algorithms for the SA problem in chains when the number of links is fixed. Finally, we present several list scheduling algorithms that are computationally efficient and simple to implement, yet produce solutions that, on average, are within 1%–5% of the lower bound.**

***Index Terms*—Approximation algorithms; Elastic optical networks; Multiprocessor scheduling; Network design; Routing and spectrum assignment; Spectrum assignment.**

I. Introduction

Orthogonal frequency division multiplexing (OFDM) technology is the foundation of the elastic optical network (EON) concept [1], also referred to as the "spectrum-sliced elastic optical path network" (SLICE) [2]. The main driver of the EON architecture is the ability to allocate bandwidth at the granularity of an OFDM subcarrier rather than at the coarse unit of a wavelength in a fixed-grid network, using bandwidth-variable and format-agile transponders that may be reconfigured dynamically via software [3]. Optical signals are routed along the path to the destination by multigranular optical switches that adapt to the data rate and center frequency of incoming channels via software control [4,5]. Bandwidth-variable transponders and switches make it possible to support efficiently a range of traffic demands, from sub- to super-wavelength, by *slicing off* just a sufficient amount of spectral resources along end-to-end paths to satisfy the client requirements. OFDM-based EONs have several advantages relative to existing WDM networks, including [1,6] resiliency to physical impairments, elastic data rates that can be adjusted to demand granularity, path distance, the time-varying nature of demands, and spectral efficiency.

The routing and spectrum assignment (RSA) problem [7,8] has emerged as the key network design and control problem in EONs. In offline RSA, the input typically consists of a set of forecast traffic demands, and the objective is to assign a physical path and contiguous spectrum to each demand so as to minimize the total amount of allocated spectrum (either over the whole network or on any link). Offline RSA arises whenever the traffic patterns in the network are reasonably well known in advance and any traffic variations take place over long time scales. For instance, offline RSA is an effective technique for provisioning a set of semipermanent connections. Since these connections are assumed to remain in place for relatively long periods of time, it is worthwhile to attempt to optimize the way in which network resources (e.g., physical links and spectrum) are assigned to each connection, even though optimization may require considerable computational effort.

Several variants of the RSA problem have been studied in the literature that take into account various design aspects including the reach versus modulation level (spectral efficiency) trade-off [9], traffic grooming [10], and restoration [11]. These problem variants are NP-hard, as RSA is a generalization of the well-known routing and wavelength assignment (RWA) problem [12]. Therefore, while most studies provide integer linear program (ILP) formulations for the RSA variant they address, they propose heuristic algorithms for solving medium to large problem instances. Such *ad hoc* solution approaches have two drawbacks. First, they do not provide insight into the structure of the optimal solution and hence cannot be easily adapted to other problem variants. Second, it is quite difficult to characterize the performance of heuristic algorithms, and our recent work has demonstrated that heuristics for the related RWA problem produce solutions that are far away from optimal even for problem instances of moderate size [13]. For a survey of spectrum management techniques in EONs, including a review of solution approaches to RSA problem variants, we refer the reader to [14].

A recent study [15] considered the complexity of the RSA problem in chain (linear) networks. In chain networks, the routing aspect of the problem is completely determined,

Manuscript received April 10, 2014; revised June 18, 2014; accepted June 29, 2014; published July 31, 2014 (Doc. ID 209860).

S. Talebi and G. N. Rouskas (e-mail: rouskas@ncsu.edu) are with Operations Research and Department of Computer Science, North Carolina State University, Raleigh, North Carolina 27695-8206, USA.

E. Bampis and G. Lucarelli are with LIP6, CNRS UMR 7606, Université Pierre et Marie Curie, 4 Place Jussieu, 75252 Paris Cedex, France.

I. Katib and G. N. Rouskas are with King Abdulaziz University, Jeddah, Saudi Arabia.

http://dx.doi.org/10.1364/JOCN.6.000754

and it reduces to a spectrum allocation (SA) problem. Using results from graph coloring theory, it was shown in [15] that the SA problem in chains is NP-hard, and that a $(2+\epsilon)$ approximation algorithm for computing the interval chromatic number of an interval graph may be used for solving the SA problem with the same performance bound. The study also extends this algorithm to solve the SA problem in ring networks with a performance bound of $(4+2\epsilon)$.

Our work makes two main contributions. We start by showing that the spectrum assignment (SA) problem in (mesh) networks of general topology is a special case of a well-known scheduling problem, namely, that of scheduling multiprocessor tasks on dedicated processors [16,17]. This new insight opens the potential to develop new approaches to the SA problem by leveraging results from the vast scheduling literature. We then carry out an in-depth study of the SA problem in chain (linear) networks and develop constant-ratio approximation algorithms and heuristics that are inspired by the new scheduling perspective. In addition to advancing our understanding of SA as a scheduling problem, our new results are important in that 1) they can be readily extended to ring networks that comprise large parts of the existing optical network infrastructure, as we have shown in more recent work [18]; 2) they may be used to analyze approximately general-topology networks, e.g., by extending path-based decomposition techniques that have been developed for the more special case of wavelength assignment [19]; and 3) they may be applied to solve large-scale multiprocessor scheduling problems efficiently and effectively, as our numerical results demonstrate.

Following this introduction, we present in Section II a transformation of the SA problem to a scheduling problem. Based on this new perspective, we show in Section III that the SA problem in chains is NP-hard for four or more links, but is solvable in polynomial time for three links. In Section IV, we develop new constant-ratio approximation algorithms for the SA problem in chains, we introduce a suite of list scheduling algorithms in Section V, and in Section VI we present numerical results to demonstrate the effectiveness of the algorithms. We conclude the paper in Section VII.

II. Spectrum Assignment Problem

We consider the following basic definition of the RSA problem.

Definition 2.1 (RSA): Given a graph $G=(\mathcal{V},\mathcal{A})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{A}$ is the set of arcs (directed links), and a spectrum demand matrix $T=[t_{sd}]$, where t_{sd} is the amount of spectrum required to carry the traffic from source node s to destination node d, assign a physical path and contiguous spectrum to each demand so as to minimize the total amount of spectrum used on any link in the network, under three constraints: 1) each demand is assigned contiguous spectrum (spectrum contiguity constraint), 2) each demand is assigned the same spectrum along all links of its path (spectrum continuity constraint), and 3) demands that share a link are assigned nonoverlapping parts of the available spectrum (nonoverlapping spectrum constraint).

If a single route for each source–destination pair is provided as part of the input, and each traffic demand is constrained to follow the given route, the RSA problem reduces to the SA problem.

Definition 2.2 (SA): The RSA problem under the additional constraint that all traffic from source s to destination d must follow the given physical path r_{sd}.

We now show that the SA problem can be viewed as a problem of scheduling tasks on multiprocessor systems in which tasks may require more than one processor simultaneously. Consider the following scheduling problem that has been studied extensively in the literature [16,17]:

Definition 2.3 ($P|fix_j|C_{max}$): Given a set of n tasks and a set of m identical processors, a processing time p_j, and a prespecified set fix_j of processors for task j, $j=1,\ldots,n$, schedule the tasks so as to minimize the makespan $C_{max}=max_j C_j$, where C_j denotes the completion time of task j, under the following constraints: 1) preemptions are not allowed, 2) each task must be processed simultaneously by all processors in fix_j, and 3) each processor can work on at most one task at a time.

We denote by $Pm|fix_j|C_{max}$ the special case of $P|fix_j|C_{max}$ in which the number of processors m is considered to be fixed. It has been shown [17] that the three-processor problem $P3|fix_j|C_{max}$ is strongly NP-hard for general processing times, but that if the number of processors m is fixed and all tasks have unit times, i.e., $Pm|fix_j,p_j=1|C_{max}$, then the problem is solvable in polynomial time. Approximation algorithms and/or polynomial time approximation schemes (PTAS) have been developed for several versions of the problem [20].

We have the following result.

Lemma 2.1: SA on general (mesh) networks transforms to $P|fix_j|C_{max}$.

Proof. Consider an instance of the SA problem on a network of general topology represented by graph $G=(\mathcal{V},\mathcal{A})$, demand matrix $T=[t_{sd}]$, and path set $\{r_{sd}\}$. Construct an instance of $P|fix_j|C_{max}$ as follows. For each arc $a_k\in\mathcal{A}$, $k=1,\ldots,|\mathcal{A}|$, there is a processor k. For each spectrum demand t_{sd}, there is a task j with $p_j=t_{sd}$ and $fix_j=\{k:a_k\in r_{sd}\}$. In other words, the amount of spectrum of a demand transforms to the processing time of the corresponding task, and the links of its path to the processors that the task requires. Due to the nonoverlapping spectrum constraint, each processor may work on at most one task at a time; due to the spectrum continuity constraint, each task must be processed simultaneously by all its processors; and due to the spectrum contiguity constraint, preemptions are not allowed. By construction, the amount of spectrum assigned to any arc of G in a solution of the SA instance is equal to the completion time of the last task scheduled on the corresponding processor; hence minimizing the spectrum on any link in the SA problem is equivalent to minimizing the makespan of the schedule in the corresponding problem $P|fix_j|C_{max}$. ∎

The above lemma shows that any instance of the SA problem can be transformed into an instance of the $P|fix_j|C_{max}$ problem, and, hence, an algorithm for solving the latter problem may be used for solving the former one. However, the reverse of Lemma 2.1 is not true. In other words, there exist instances of $P|fix_j|C_{max}$ for which there is no corresponding instance of the SA problem, as we now show.

Lemma 2.2: There exist instances of $P|fix_j|C_{max}$ for which there is no corresponding instance of the SA problem.

Proof. By counterexample. Consider an instance of $P4|fix_j|C_{max}$ with these five tasks whose processing times can be arbitrary:

Task	fix_j
τ_1	{1, 2}
τ_2	{2, 3}
τ_3	{3, 4}
τ_4	{4, 1}
τ_5	{2, 4}

Because of the first four tasks, the graph of the corresponding SA instance would have to be the four-link unidirectional ring network such that link 1 is adjacent to 2, 2 is adjacent to 3, 3 to 4, and 4 to 1. But then, there is no path r_{sd} for the spectrum demand corresponding to the last task; hence an instance of SA does not exist. ∎

The following lemma shows that the SA problem in unidirectional rings with as few as three links is NP-hard.

Lemma 2.3: The SA problem in unidirectional rings is NP-hard.

Proof. The $P3|fix_j|C_{max}$ problem can be transformed to the SA problem on a unidirectional three-link ring, where each processor corresponds to a link, and each task j corresponds to the traffic demand on the segment of the ring defined by the links in fix_j. Since $P3|fix_j|C_{max}$ is NP-complete [17], the same is true for the SA problem on the three-link ring. ∎

A. SA Problem in Chains

In chain (linear) networks, the route r_{sd} of each traffic demand is uniquely determined by its source and destination nodes. Let us define the following special case of problem $P|fix_j|C_{max}$:

Definition 2.4 ($P|line_j|C_{max}$): The $P|fix_j|C_{max}$ problem under the additional constraint that the identical processors are labeled 1, 2, 3,..., and the prespecified set $line_j$ of processors for each task j consists of processors with consecutive labels.

The following result states that the SA problem on a chain with m links is equivalent to $P|line_j|C_{max}$; hence an algorithm for solving $P|line_j|C_{max}$ may be used to solve SA, and *vice versa*.

Lemma 2.4: The SA problem on a graph G that is a chain with m links is equivalent to $P|line_j|C_{max}$.

Proof. First consider an instance of the SA problem on a chain G with $m+1$ nodes labeled $1, 2, \ldots, m+1$, and m arcs $a_1 = \langle 1, 2\rangle, a_2 = \langle 2, 3\rangle, \ldots, a_m \langle m, m+1\rangle$, and spectrum demand matrix $T = [t_{sd}]$ such that $t_{sd} = 0$ if $s \geq d$. Given this SA instance, the steps of the proof of Lemma 2.1 will construct a valid instance of $P|line_j|C_{max}$ since the sets fix_j consist of processors with consecutive labels.

Given an instance of $P|line_j|C_{max}$, we construct an instance of the SA problem on a chain graph G as follows. The graph has $m+1$ nodes labeled $1, 2, \ldots, m+1$, and m arcs, such that for each processor k, $k = 1, \ldots, m$, there is an arc $a_k = \langle k, k+1\rangle$. For each task j with $line_j = \{s, \ldots, d-1\}$, there is a traffic demand with $t_{sd} = p_j$ and route $r_{sd} = \{\langle s, s+1\rangle, \ldots, \langle d-1, d\rangle\}$. It is not difficult to verify that the three constraints of $P|line_j|C_{max}$ ensure that the three constraints of SA are satisfied, and that minimizing the makespan C_{max} minimizes the maximum amount of spectrum on any arc of G. ∎

In theory, algorithms developed for $P|fix_j|C_{max}$ may be applied to schedule instances of $P|line_j|C_{max}$. Due to its difficulty, however, there are limited results for the former problem, and even algorithms for a small number of processors (e.g., $m = 3, 4, 5$) can be quite complex and not practical for a network operator.

In the following section, we determine the complexity of $Pm|line_j|C_{max}$. Following that, we present approximation algorithms for the most general version of the problem, i.e., for any number of processors and any task sizes.

III. Complexity Results for $Pm|line_j|C_{max}$

We first show that there is a polynomial time algorithm for $P3|line_j|C_{max}$. Recall that problem $P3|fix_j|C_{max}$ is strongly NP-hard [17]; hence the additional constraint that the set of processors in $line_j$ have consecutive labels makes the problem tractable for three processors. Furthermore, note that whereas it was stated in [15] that the SA problem in chains is NP-hard, this result implies that the problem is in fact polynomial on three-link chains.

Lemma 3.1: $P3|line_j|C_{max}$ may be solved in polynomial time.

Proof. The proof is by construction of the optimal schedule. Tasks that require all three processors cannot be executed in parallel with any other task, and hence they may be simply added at the beginning or end of the schedule without affecting optimality. Therefore, we focus our attention on tasks requiring either one or two processors, i.e., tasks with $line_j = \{1\}, \{2\}, \{3\}, \{1, 2\}$, or $\{2, 3\}$. Without loss of generality, let processor 1 be the dominant processor, i.e., the one that requires the most processing time; the case in which either processor 2 or processor 3 is dominant can be handled in a very similar manner. Construct the following schedule. Tasks with $line_j = \{1, 2\}$ are executed back to back without any idle time, followed by tasks with $line_j = \{1\}$. Let t be the time when the last task with $line_j = \{1\}$ completes. Schedule tasks with $line_j = \{2, 3\}$, without any idle time between them, at the end of the schedule and in parallel with tasks with $\{line_j = 1\}$, so that the last

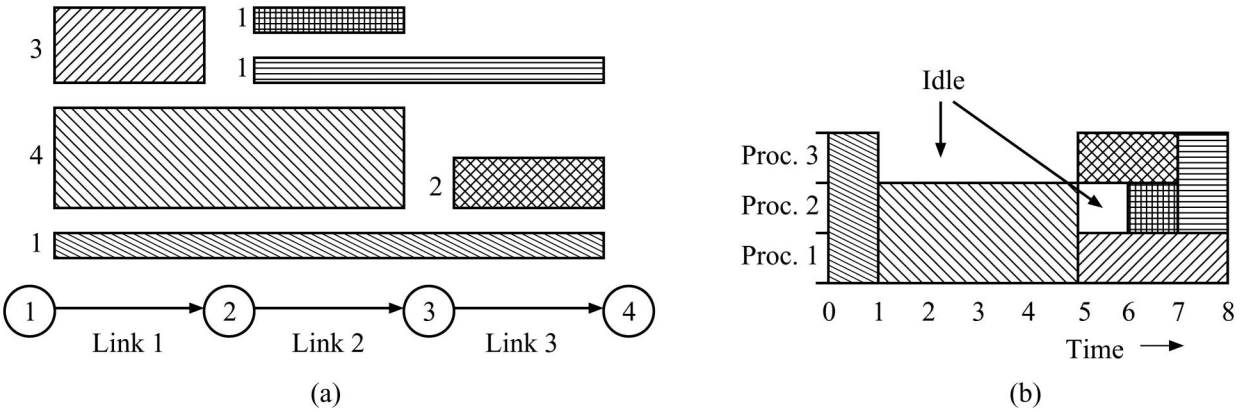

Fig. 1. (a) Instance of the offline RSA problem on a chain. (b) Optimal schedule for the corresponding $P3|line_j|C_{max}$ problem.

one finishes at time t. Then, schedule tasks with $line_j = \{2\}$ and $line_j = \{3\}$ before tasks with $line_j = \{2, 3\}$. Clearly, these tasks can fit in the schedule since processors 2 and 3 are not dominant. The schedule is optimal since the dominant processor is never idle, and hence the makespan $C_{max} = t$, the time required to execute the tasks on the dominant processor. ∎

As an example, consider the RSA problem instance on a chain network with four nodes and three links, as shown in Fig. 1(a). For this instance, the spectrum demand matrix is

$$T = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The corresponding $P3|line_j|C_{max}$ instance has six tasks:

Task	p_j	$line_j$
τ_1	1	{1, 2, 3}
τ_2	4	{1, 2}
τ_3	3	{1}
τ_4	1	{2, 3}
τ_5	1	{2}
τ_6	2	{3}

The optimal schedule constructed as described in the proof of Lemma 3.1 is shown in Fig. 1(b). In this schedule, tasks requiring all three processors are executed first.

The following theorem shows that the problem $Pm|line_j|C_{max}$ is NP-complete for four or more processors. The proof is based on a reduction from the PARTITION problem [21], which is defined as follows.

Definition 3.1 (PARTITION): Given a set of k integers $A = \{a_1, a_2, \ldots, a_k\}$ such that $B = \sum_{j=1}^{k} a_j$, does there exist a partition of A into two sets, A_1 and A_2, such that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = B/2$?

Theorem 3.1: $P4|line_j|C_{max}$ is NP-complete.

Proof. Given an instance of PARTITION, we create an instance of $P4|line_j|C_{max}$ as follows. For each $a_j \in A$ we create a task τ_j with processing time $p_j = a_j$ and $line_j = \{3\}$. Moreover, we create the following gadget tasks:

Task	p_j	$line_j$
T_a	$B/2$	{1, 2, 3}
T_b	$B/2$	{1, 2}
T_c	$B/2$	{2, 3}
T_d	$3B/2$	{3, 4}
T_e	$B/2$	{2, 3, 4}
T_1	$3B$	{1}
T_2	$2B$	{2}
T_3	$2B$	{4}

If there is a partition of A into A_1 and A_2 such that $\sum_{a_j \in A_1} = \sum_{a_j \in A_2} = B/2$, then we can schedule the jobs as shown in Fig. 2 and get a feasible schedule with $C_{max} = 4B$.

Let us now assume that there exists a feasible schedule $\mathcal{S}$ with $C_{max} \leq 4B$. Without loss of generality, assume that T_a is executed before T_e in $\mathcal{S}$; otherwise, we can use similar arguments and reach the same conclusion. Then, it must be that T_3 is executed before both T_d and T_e. Moreover, T_d must be executed before T_e; otherwise it would not be possible to schedule task T_2 for the schedule to have length at most $4B$. Hence, tasks T_a, T_3, T_d, and T_e must be scheduled in the order shown in Fig. 2. Moreover, tasks T_a and T_b must be scheduled before T_1, while task T_c must be scheduled before T_2 for the schedule length to be at most $4B$. If T_a is scheduled before T_b (as shown in Fig. 2), then on processor 3, only the intervals $[B/2, B]$ and $[3B/2, 2B]$ are available for the execution of the PARTITION jobs. If T_b is scheduled before T_a, then on processor 3, only the intervals

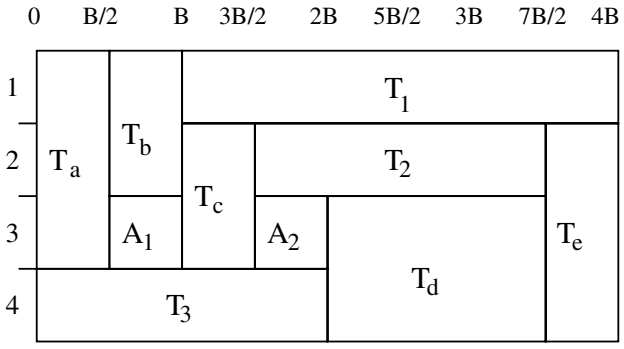

Fig. 2. Feasible schedule with $C_{max} = 4B$.

$[0, B/2]$ and $[3B/2, 2B]$ are available for the PARTITION jobs. In both cases, a partition exists. ∎

As stated in [15], the above result confirms that the SA problem on chains with four or more links is harder than the wavelength assignment problem, which can be solved in polynomial time on chains.

IV. Approximation Algorithms for $P|line_j|C_{max}$

We first show that there exist 1.5-approximation algorithms for four and five processors.

Lemma 4.1: There exists a 1.5-approximation algorithm for $P4|line_j|C_{max}$.

Proof. The 1.5-approximation algorithm for the $P4|fix_j|C_{max}$ problem [22] can be used to solve $P4|line_j|C_{max}$ with the same performance bound. ∎

Lemma 4.2: There exists a 1.5-approximation algorithm for $P5|line_j|C_{max}$.

The proof is by construction. Due to its length, the proof is omitted but is available in the first author's dissertation.

A. Two-Stage Approximation Algorithms

We now introduce a two-stage algorithm for $P|line_j|C_{max}$, and show that it yields a constant approximation ratio for any number of processors $m \geq 6$. The algorithm, described in Algorithm 1, considers three sets of processors based on their labels: a set consisting of the $k < m$ processors with low index labels $1, \ldots, k$, a set of l processors, $l + k < m$, with labels $k + 1, \ldots, k + l$, and a set of $m - k - l$ processors with the high index labels $k + l + 1, \ldots, m$. We partition the set of tasks into three sets:

- set $\mathcal{T}_{mid}$ consists of tasks that require at least one of the l middle processors (and may also require processors from one or both of the other sets);
- set $\mathcal{T}_{lo}$ contains tasks requiring only processors in the set of k low index processors (and no other processor); and
- set $\mathcal{T}_{hi}$ is composed of tasks that require only processors in the set of $m - k - l$ high index processors (and no other processor).

The key idea is based on the observation that the set of tasks $\mathcal{T}_{lo}$ may be scheduled in parallel with the set of tasks $\mathcal{T}_{hi}$. Therefore, we use an optimal or approximation algorithm to schedule the tasks in each set separately, creating three schedules, $\mathcal{S}_{mid}$, $\mathcal{S}_{lo}$, and $\mathcal{S}_{hi}$, respectively. The final schedule for the original problem consists of two stages: in the first stage, schedule $\mathcal{S}_{mid}$ is executed individually, while in the second stage, schedules $\mathcal{S}_{lo}$ and $\mathcal{S}_{hi}$ are executed in parallel.

We have the following result.

Lemma 4.3: Let α_{mid}, α_{lo}, and α_{hi} be the approximation ratios of the algorithms used to schedule tasks in sets $\mathcal{T}_{mid}$, $\mathcal{T}_{lo}$, and $\mathcal{T}_{hi}$, respectively (the approximation ratio is 1 if an optimal algorithm exists). Then the two-stage algorithm of Algorithm 1 is an approximation algorithm for the original problem with ratio

$$\alpha = \alpha_{mid} + \max\{\alpha_{lo}, \alpha_{hi}\}. \tag{1}$$

Proof: The proof follows from the fact that the makespan of an optimal schedule for each of the three sets of tasks is no longer than the makespan of an optimal schedule for the original set of tasks. ∎

Algorithm 1 Two-Stage Approximation Algorithm for $P|line_j|C_{max}$, $m \geq 6$

Input: A set $\mathcal{T}$ of n tasks on m processors, each task j having a processing time p_j and a set $line_j \subseteq \{1, 2, \ldots, m\}$ of required processors
Output: A two-stage schedule of tasks

begin

//Sets of low, mid, and high index processors
1. $lo \leftarrow \{1, \ldots, k\}$ //$k < m$
2. $mid \leftarrow \{k + 1, \ldots, k + l\}$ //$k + l < m$
3. $hi \leftarrow \{k + l + 1, \ldots, m\}$

//Set of tasks $\mathcal{T}$ for each subproblem
4. $\mathcal{T}_{mid} \leftarrow \{j \in \mathcal{T} : \{k + 1, \ldots, k + l\} \cap line_j \neq \emptyset\}$
5. $\mathcal{T}_{lo} \leftarrow \{j \in \mathcal{T} \setminus \mathcal{T}_{mid} : \{1, \ldots, k\} \cap line_j \neq \emptyset\}$
6. $\mathcal{T}_{hi} \leftarrow \{j \in \mathcal{T} \setminus \mathcal{T}_{lo} \setminus \mathcal{T}_{mid}\}$
7. Schedule tasks in $\mathcal{T}_{mid}$ using an optimal or approximation algorithm for the corresponding $Pl|line_j|C_{max}$ problem
8. Schedule tasks in $\mathcal{T}_{lo}$ using an optimal or approximation algorithm for the corresponding $Pk|line_j|C_{max}$ problem
9. Schedule tasks in $\mathcal{T}_{hi}$ using an optimal or approximation algorithm for the corresponding $P(m - k - l)|line_j|C_{max}$ problem
10. **return** a schedule of two stages: the first stage consists of the schedule for tasks $\mathcal{T}_{mid}$; the second stage consists of the schedule for tasks $\mathcal{T}_{lo}$ and $\mathcal{T}_{hi}$ executed in parallel

end

Figure 3 shows a two-stage schedule for $m = 9$ processors in which $k = l = m - k - l = 3$. Due to Lemma 3.1, the $P3|line_j|C_{max}$ problem can be solved optimally in polynomial time; hence $\alpha_{mid} = \alpha_{lo} = \alpha_{hi} = 1$. Therefore, the two-stage algorithm is a two-approximation algorithm for $m = 9$ processors (and also for problems with $m = 6, 7, 8$ processors).

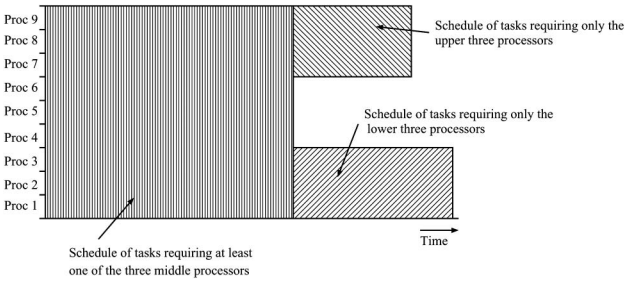

Fig. 3. Two-approximation schedule for $m = 9$ processors.

For problems with $m = 10$–13 processors, we consider the middle three processors to obtain task set $\mathcal{T}_{mid}$, and apply the 1.5-approximation algorithm for four or five processors to schedule the other two task sets, resulting in an approximation ratio of 2.5. For problems with $m = 14, 15$ processors, we obtain an approximation ratio of 3 by considering task sets on four or five processors. Finally, we note that for $m > 15$, we may apply the two-stage algorithm recursively to schedule each set of tasks. For instance, for $m = 19$, we can schedule the tasks that require at least one of the middle nine processors with a makespan that is no more than twice the optimal, as in Fig. 3. Applying the 1.5-approximation algorithm to schedule the tasks requiring at least one of the five lower (respectively, higher) index processors, the overall approximation ratio of the two-stage algorithm is 3.5.

More generally, the approximation ratio $\alpha(m)$ for any number m of processors can be computed using the recurrence relationship:

$$\alpha(m) = \begin{cases} 1 & m = 1, 2, 3 \\ 1.5 & m = 4, 5 \\ \min\limits_{l=1,\ldots,m-2} \left\{ \alpha(l) + \alpha\left(\left\lceil \frac{m-l}{2} \right\rceil \right) \right\} & m \geq 6 \end{cases} . \quad (2)$$

The recursive expression can be explained by making two observations. First, for a given number l of processors in the set $\mathcal{T}_{mid}$, the approximation ratio in Eq. (1) is minimized when the number of processors in the sets $\mathcal{T}_{lo}$ and $\mathcal{T}_{hi}$ differs by at most one; in this case, the number of processors in the larger of the two sets is $\lceil m - l/2 \rceil$, and determines the result of the maximum operation in expression (1). Second, all three sets of processors must be nonempty, hence the range of values of l in the expression. Therefore, the approximation ratio for a given value of l is $\alpha(l) + \alpha(\lceil m - l/2 \rceil)$, and the best ratio can be obtained by taking the minimum over all possible values of l.

V. List Scheduling Algorithms for $Pm|line_j|C_{max}$

In this section, we present a suite of list scheduling algorithms for solving the $Pm|line_j|C_{max}$ problem; due to Lemma 2.4, these algorithms may also be used to solve the SA problem on chains. The algorithms attempt to minimize the makespan C_{max} by identifying *compatible* tasks, i.e., sets of tasks that may be executed simultaneously on the multiprocessor system. More formally, we have the following definition.

Definition 5.1: A set $\mathcal{T}$ of tasks for the $Pm|line_j|C_{max}$ problem is said to be compatible if and only if their prespecified sets of processors are pairwise disjoint, i.e., $line_j \cap line_j = \emptyset, \forall i, j \in \mathcal{T}$.

We present two broad classes of algorithms that differ in the granularity at which they make scheduling decisions. The first class of algorithms assembles compatible tasks into *blocks*, and schedules a whole block of tasks at a time. The second class of algorithms operates at finer granularity and makes scheduling decisions at the level of individual tasks and at finer time scales.

A. Block-Based Scheduling Algorithms

These algorithms proceed by constructing *blocks* of compatible tasks. Specifically, blocks of compatible tasks are scheduled such that

- all tasks in a block begin execution at the same time t, and
- there is no idle time between the completion of the longest task in a block and the beginning of the next block.

The input to the algorithm is a list of tasks. The algorithm assembles a block by selecting the first task in the list, and then scanning the remaining tasks (in the order listed) to identify tasks compatible with the ones already in the block. A block is considered complete if no additional compatible tasks exist; the algorithm removes all the tasks of the block from the list and continues to build the next block, until all tasks have been scheduled.

A pseudocode description of a block-based scheduling algorithm is presented in Algorithm 2. The running time complexity of the algorithm is determined by the two **while** loops. In the worst case, each loop is executed at most n times, where n is the number of tasks. Hence, the overall running time of the algorithm is $O(n^2)$. This overall complexity is not affected by accounting for the time it takes to create the list of tasks given as input to the algorithm, as creating this list would typically involve sorting the n tasks according to some attribute.

Algorithm 2 Block-Based Scheduling Algorithm for $Pm|line_j|C_{max}$

Input: A list L of n tasks on m processors, each task j having a processing time p_j and a set $line_j \subseteq \{1, 2, \ldots, m\}$ of required processors

Output: A schedule of task blocks, each block consisting of compatible tasks, such that each block starts execution immediately after the previous block completes

begin
1. $b \leftarrow 1$ //b is the block index
2. $S_b \leftarrow 0$ //The time that block b starts execution
3. $P_b \leftarrow \emptyset$ //The union of sets $line_j$ of tasks in block b
4. **while** list $L \neq \emptyset$ **do**
5. Remove the first task j from list L
6. Add task j to block b
7. $P_b \leftarrow P_b \cup line_j$
8. **while** not at end of list L **or** $P_b \neq \{1, 2, \ldots, m\}$ **do**
9. $k \leftarrow$ first task in list
10. **if** $line_k \cap P_b = \emptyset$ **then**
11. Remove the task k from list L
12. Add task k to block b
13. $P_b \leftarrow P_b \cup line_k$
14. **end while** //no more tasks may be added to block b
15. $b \leftarrow b + 1$
16. $S_b \leftarrow S_{b-1} + \max_{j \in b-1} p_j$
17. $P_b \leftarrow \emptyset$
18. **end while**
19. **return** tasks in each block, block starting times S_b

end

We identify two block-based scheduling algorithms that differ in the order in which the tasks are sorted in the initial list of tasks passed to the algorithm:

- **Longest first block-based algorithm (LFB)**: tasks are sorted in decreasing order of their processing times p_j.
- **Widest first block-based algorithm (WFB)**: tasks are sorted in decreasing order of the number $|line_j|$ of processors they require.

B. Compact Scheduling Algorithms

Block-based schedules are simple to create and implement in that each task in a block starts execution at exactly the same time. However, the fact that tasks within a block have varying processing times may result in long idle times for some processors. Consequently, the makespan of the final schedule may be longer than necessary. We now present a class of scheduling algorithms that attempt to minimize the makespan by eliminating or reducing such idle times. Rather than assembling blocks of tasks and making scheduling decisions at the end of each block, these algorithms select individual tasks for execution at finer scheduling instants, resulting in more compact schedules. The scheduling instants consist of the following:

- the start time of the schedule (i.e., $t = 0$), and
- the instant each task completes execution.

The input to a compact algorithm is a list of tasks. The algorithm maintains a list of idle processors. At each scheduling instant t, the algorithm scans the list to identify tasks that are compatible with the currently executing ones, i.e., tasks with a set $line_j$ that is a subset of the free processors. When such a task is identified, the algorithm removes it from the list, updates the set of free processors, and continues scanning the list until no other compatible task is found. It then advances to the next scheduling instant and repeats the process while the list is not empty.

Algorithm 3 presents a pseudocode description of a compact scheduling algorithm. The running time complexity of the algorithm is $O(n^2)$. Similar to block-based algorithms, we distinguish two algorithms based on the order in which tasks appear in the list:

- **Longest first compact algorithm (LFC)**: tasks are sorted in decreasing order of their processing times p_j.
- **Widest first compact algorithm (WFC)**: tasks are sorted in decreasing order of the number $|line_j|$ of processors they require.

Since compact scheduling algorithms make decisions at finer time scales, we expect that they perform better than block-based ones.

Algorithm 3 Compact Scheduling Algorithm for $Pm|line_j|C_{max}$

Input: A list L of n tasks on m processors, each task j having a processing time p_j and a set $line_j \subseteq \{1, 2, \ldots, m\}$ of required processors
Output: A schedule of tasks, i.e., the time S_j when each task j starts execution on the multiprocessor system
begin
1. $t \leftarrow 0$ //Scheduling instant
2. $F \leftarrow \{1, \ldots, m\}$ //Set of currently idle processors
3. **while** list $L \neq \emptyset$ **do**
4. $j \leftarrow$ first task in list L
5. **if** $line_j \subseteq F$ **then**
6. Remove the task j from list L
7. $S_j \leftarrow t$ //Task j starts execution at time t
8. $F \leftarrow F \backslash line_j$
9. **while** not at the end of list L **or** $F \neq \emptyset$ **do**
10. $k \leftarrow$ first task in list
11. **if** $line_k \subseteq F$ **then**
12. Remove the task k from list L
13. $S_k \leftarrow t$ Task k starts execution at time t
14. $F \leftarrow F \backslash line_k$
15. **end while** //no more tasks may start at time t
16. $j \leftarrow$ the first task executing at time t to complete
17. $t \leftarrow S_j + p_j$
18. $P \leftarrow P \cup line_j$
19. **end while**
20. **return** the task start times S_j
end

VI. Numerical Results

In the following two subsections, we discuss two sets of experiments we carried out to evaluate the performance of the list scheduling algorithms.

A. Spectrum Assignment in a Chain Network

In the first set of experiments, we consider instances of the $Pm|line_j|C_{max}$ problem with a relatively small number of processors, namely, $m = 5, 10, 15, 20$; such instances correspond to instances of the SA problem on chains of length typical of a commercial wide-area network. For each problem instance, we generated traffic demands between every source and destination node on the chain. The size of the traffic demands (i.e., task times) was generated using three different distributions:

- *Discrete uniform:* traffic demands may take any of the five discrete values in the set {10, 40, 100, 400, 1000} with equal probability; these values correspond to data rates (in Gbps) to be supported by EONs.
- *Discrete high:* traffic demands may take one of the five discrete values above with probabilities 0.10, 0.15, 0.20, 0.25, and 0.30, respectively; in other words, higher data rates have higher probability to be selected.

- *Discrete low:* traffic demands may take one of the five discrete values above with probabilities 0.30, 0.25, 0.20, 0.15, and 0.10, respectively, such that lower data rates have higher probability to be selected.

We considered various other probability distributions on the same set of values, but the results are similar to the ones we present below and hence are omitted.

We use a distance-adaptive SA strategy to allocate spectrum to each traffic demand based on its data rate (in Gbps) and the length of its path (i.e., number of links, or processors in the scheduling problem) [1,23]. We assume a 12.5 GHz slot width, and consider two modulation formats: for paths with up to (respectively, more than) 10 links we assume 16-QAM (respectively, QPSK), such that demands of size 10, 40, 100, 400, and 1000 Gbps are assigned 1, 1, 2, 8, and 20 (respectively, 1, 2, 4, 16, and 40) slots, consistent with the values used in [23, Table 1].

Figures 4–6 plot the average ratio achieved by the four list scheduling algorithms, LFB, LFC, WFB, and WFC, for each of the three traffic distributions; specifically, each data point in the figures represents the average over 30 randomly generated problem instances. The average ratio for a given algorithm is defined as the ratio of the makespan value C_{max} (i.e., maximum spectrum used on any link of the corresponding chain) obtained by the algorithm for a given problem instance over the lower bound (i.e., the total processing time for the dominant processor) for the same instance. Recall that the $Pm|line_j|C_{max}$ problem is NP-hard for the number $m \geq 5$ of processors considered in this experiment, and the optimal value is not known. Since the optimal value is no less than the lower bound, the average ratio shown in the figures *overestimates* the average gap between the C_{max} values obtained by the algorithms and the optimal one.

From the figures, we can make several observations. First, the compact algorithms (LFC and WFC) perform better than the corresponding block-based algorithms (LFB and WFB, respectively). Second, three of the algorithms (LFC, LFB, and WFC) obtain solutions that are within 5% (and in many cases, within 2%–3%) of the lower bound. Furthermore, the average ratio performance of these three

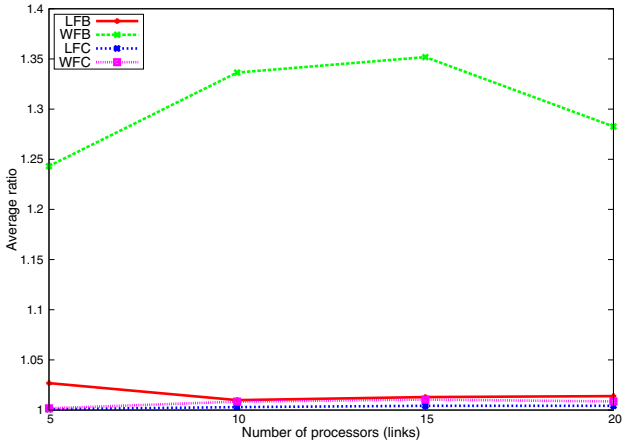

Fig. 4. Average ratio versus number of processors, discrete uniform distribution.

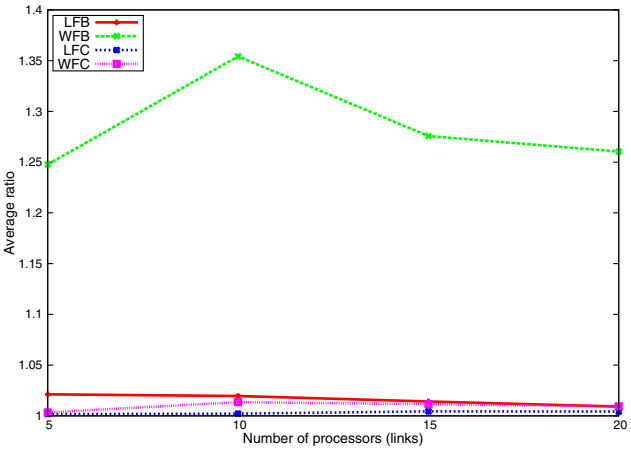

Fig. 5. Average ratio versus number of processors, discrete high distribution.

algorithms is not sensitive to the number of processors (chain length) or demand distribution. The block-based WFB algorithm has the worst performance over all problem instances considered in this study. This result indicates that processing tasks in the order of decreasing number of processors they require may pair short tasks with long tasks, creating large idle times within blocks. On the other hand, the WFC algorithm that processes tasks in the same order is successful in reducing these idle times, demonstrating the importance of taking into consideration the idle times in the scheduling process. Overall, these results indicate that, for problem instances representative of SA problems arising in chains typical of the diameter of metropolitan or wide-area networks, the two compact algorithms (LFC and WFC) may obtain solutions very close to the optimal with low computational requirements.

B. Scheduling on Large Multiprocessor Systems

In this set of experiments, we consider instances of the $Pm|line_j|C_{max}$ problem applicable to large-scale

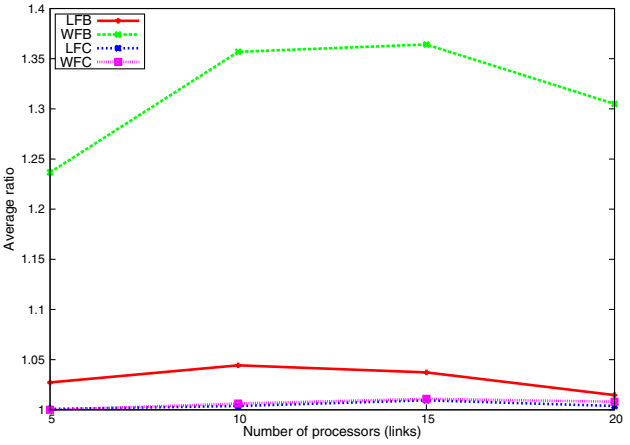

Fig. 6. Average ratio versus number of processors, discrete low distribution.

multiprocessor systems. Specifically, we let the number m of processors vary from 1000 to 6000, in increments of 1000. For each problem instance, we generated a number of tasks equal to twice the number of processors (i.e., $n = 2m$), while the task times were selected from three distributions:

- *Uniform:* task times may take any integer value in the interval [10, 1000] with equal probability.
- *Skewed high:* task times may take values in the intervals [10, 200], (200, 400], (400, 600], (600, 800], and (800, 1000] with probabilities 0.10, 0.15, 0.20, 0.25, and 0.30, respectively. Once a task has been assigned to one of these intervals, it is assigned any value in that interval uniformly and randomly.
- *Skewed low:* this is similar to the skewed high distribution, with the only difference that the probabilities that a task falls within one of the five intervals are 0.30, 0.25, 0.20, 0.15, and 0.10, respectively.

Figures 7–9 plot the average ratio achieved by the LFC, LFB, and WFC algorithms as a function of the number m of processors; each of the figures corresponds to problem instances generated by one of the three task length distributions above. As in the earlier figures, each data point represents the average over 30 random problem instances. Similar to the results presented in the previous subsection, the WFB algorithm produces solutions with an average ratio of 1.3–1.5, substantially higher than the ones achieved by these three algorithms; hence, we omit the WFB algorithm from these figures so as to focus on the behavior of the best algorithms. We observe that all three algorithms perform very close to the lower bound: their solutions are around 1%–3% away from the lower bound, on average, when $m = 1000$, and the average ratio improves (drops) as the number of processors increases to $m = 6000$. This behavior is consistent across all three distributions we considered for these experiments. Note that the *absolute* difference between the makespan of the solutions and the lower bound actually *increases* slowly as the number of processors increases, but the relative difference (i.e., the ratio plotted in the figures) decreases because, for a given distribution, the value of the lower bound increases with the number of processors.

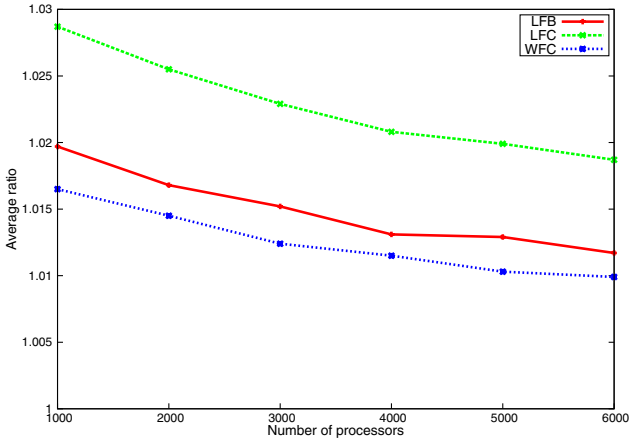

Fig. 7. Average ratio versus number of processors, uniform distribution.

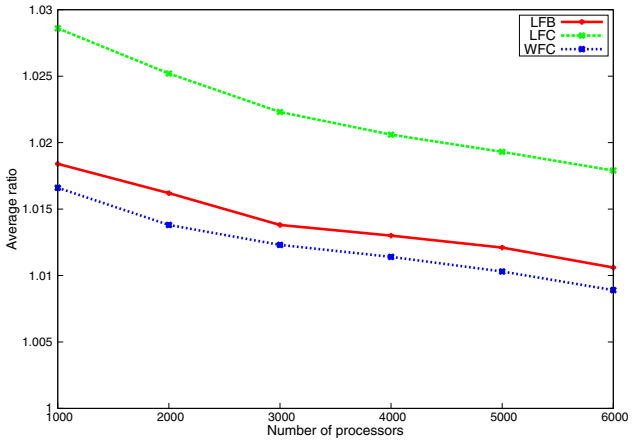

Fig. 8. Average ratio versus number of processors, skewed high distribution.

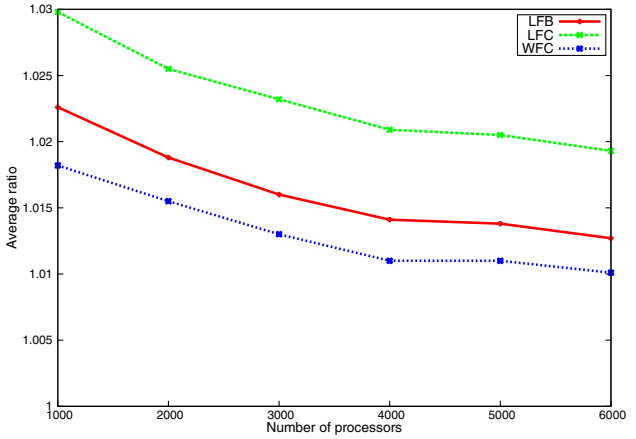

Fig. 9. Average ratio versus number of processors, skewed low distribution.

Overall, our experiments indicate that the three algorithms, LFC, LFB, and WFC, perform very close to the lower bound across a range of task length distributions and number of processors, while being computationally efficient and simple to implement.

VII. Concluding Remarks

We have studied the SA problem in EONs from a new perspective, and we have shown that it transforms to the problem of scheduling multiprocessor tasks on dedicated processors. Using this new insight, we have developed simple two-stage approximation algorithms for chain networks. We have also presented a suite of list scheduling algorithms that are computationally efficient and produce solutions that, on average, are very close to the lower bound for problem instances defined on chain networks. Our current research focuses on two directions: developing new approximation algorithms with a better

performance bound, and building upon multiprocessor scheduling theory to devise efficient algorithms with good performance for ring and mesh networks.

Acknowledgments

This work was supported by the National Science Foundation under grant CNS-1113191, and in part by the Deanship of Scientific Research (DSR), King Abdulaziz University, under grant 2-611-1434-HiCi. This is an extended version of a paper that appeared in ONDM 2014.

References

[1] O. Gerstel, M. Jinno, A. Lord, and S. J. B. Yoo, "Elastic optical networking: A new dawn for the optical layer?" *IEEE Commun. Mag.*, vol. 50, no. 2, pp. s12–s20, 2012.

[2] M. Jinno, H. Takara, B. Kozicki, Y. Tsukishima, Y. Sone, and S. Matsuoka, "Spectrum-efficient and scalable elastic optical path network: Architecture, benefits, and enabling technologies," *IEEE Commun. Mag.*, vol. 47, no. 11, pp. 66–73, 2009.

[3] W. Shieh, "OFDM for flexible high-speed optical networks," *J. Lightwave Technol.*, vol. 29, no. 10, pp. 1560–1577, May 2011.

[4] S. Frisken, G. Baxter, D. Abakoumov, H. Zhou, I. Clarke, and S. Poole, "Flexible and grid-less wavelength selective switch using LCOS technology," in *Proc. OFC/NFOEC*, 2011, paper OTuM3.

[5] R. Ryf, Y. Su, L. Moller, S. Chandrasekhar, X. Liu, D. T. Nelson, and C. R. Giles, "Wavelength blocking filter with flexible data rates and channel spacing," *J. Lightwave Technol.*, vol. 23, no. 1, pp. 54–61, Jan. 2005.

[6] G. Zhang, M. De Leenheer, A. Morea, and B. Mukherjee, "A survey on OFDM-based elastic core optical networking," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 1, pp. 65–87, 2013.

[7] M. Klinkowski and K. Walkowiak, "Routing and spectrum assignment in spectrum sliced elastic optical path network," *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 884–886, 2011.

[8] Y. Wang, X. Cao, and Y. Pan, "A study of the routing and spectrum allocation in spectrum-sliced elastic optical path networks," in *Proc. IEEE INFOCOM*, 2011, pp. 1503–1511.

[9] K. Christodoulopoulos, I. Tomkos, and E. A. Varvarigos, "Elastic bandwidth allocation in flexible OFDM–based optical networks," *J. Lightwave Technol.*, vol. 29, no. 9, pp. 1354–1366, 2011.

[10] Y. Zhang, X. Zheng, Q. Li, N. Hua, Y. Li, and H. Zhang, "Traffic grooming in spectrum-elastic optical path networks," in *Proc. of Optical Fiber Communication Conf. and the National Fiber Optic Engineers Conf. (OFC/NFOEC)*, Mar. 2011, paper OTuI1.

[11] Y. Wei, G. Shen, and S. You, "Span restoration for CO-OFDM-based elastic optical networks under spectrum conversion," in *Proc. of Asia Communications and Photonics Conf. (ACP)*, Nov. 2012, paper AF3E.7.

[12] G. N. Rouskas, "Routing and wavelength assignment in optical WDM networks," in *Wiley Encyclopedia of Telecommunications*, J. Proakis, Ed. Wiley, 2001.

[13] Z. Liu and G. N. Rouskas, "A fast path-based ILP formulation for offline RWA in mesh optical networks," in *Proc. IEEE GLOBECOM*, Anaheim, CA, Dec. 2012, pp. 2990–2995.

[14] S. Talebi, F. Alam, I. Katib, M. Khamis, R. Khalifah, and G. N. Rouskas, "Spectrum management techniques for elastic optical networks: A survey," *Opt. Switching Netw.*, vol. 13, pp. 34–48, July 2014.

[15] S. Shirazipourazad, C. Zhou, Z. Derakhshandeh, and A. Sen, "On routing and spectrum allocation in spectrum-sliced optical networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 385–389.

[16] E. Bampis, M. Caramia, J. Fiala, A. Fishkin, and A. Iovanella, "Scheduling of independent dedicated multiprocessor tasks," in *Proc. of the 13th Annu. Int. Symp. on Algorithms and Computation*, vol. 2518, Lecture Notes in Computer Science, 2002, pp. 391–402.

[17] J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman, "Complexity of scheduling multiprocessor tasks with prespecified processor allocations," *Discrete Appl. Math.*, vol. 55, pp. 259–272, 1994.

[18] S. Talebi, F. Alam, I. Katib, and G. N. Rouskas, "Spectrum assignment in rings with shortest path routing: Complexity and approximation algorithms," to be presented at *Proc. ICNC*, Anaheim, CA.

[19] Y. Zhu, G. N. Rouskas, and H. G. Perros, "A path decomposition approach for computing blocking probabilities in wavelength routing networks," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 747–762, Dec. 2000.

[20] E. Bampis and A. Kononov, "On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements," in *Proc. of the 15th Int. Parallel and Distributed Processing Symp.*, San Francisco, CA, 2001.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability.* New York: W. H. Freeman, 1979.

[22] J. Huang, J. Chen, S. Chen, and J. Wang, "A simple linear time approximation algorithm for multi-processor job scheduling on four processors," *J. Comb. Opt.*, vol. 13, pp. 33–45, 2007.

[23] M. Jinno, B. Kozicki, H. Takara, A. Watanabe, Y. Sone, T. Tanaka, and A. Hirano, "Distance-adaptive spectrum resource allocation in spectrum-sliced elastic optical path network," *IEEE Commun. Mag.*, vol. 48, no. 8, pp. 138–145, 2010.