The Spectrum Assigment (SA) Problem in Optical Networks: A Multiprocesor Scheduling Perspective

Sahar Talebi[†], Evripidis Bampis[‡], Giorgio Lucarelli[‡], Iyad Katib[⋆], George N. Rouskas[†⋆]
[†]Operations Research and Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206 USA
[‡]Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
[⋆]King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—The routing and spectrum assignment (RSA) problem has emerged as the key design and control problem in elastic optical networks. In this work, we provide new insight into the spectrum assignment (SA) problem in mesh networks by showing that it transforms to the problem of scheduling multiprocessor tasks on dedicated processors. Based on this new perspective, we show that the SA problem in paths 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 paths when the number of links is fixed. Finally, we introduce a suite of 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.**

I. Introduction

The ongoing transition from a fixed wavelength grid to an elastic network paradigm [5] opens up many exciting and promising directions for optical network research and development. This transition is facilitated by key enabling technologies including optical OFDM, adaptive modulation, bandwidth-variable transponders and flexible spectrum selective switches [16]. Elastic optical networks have the potential to overcome the fixed, coarse granularity of existing WDM technology and are expected to support flexible data rates, adapt dynamically to variable bandwidth demands by applications, and utilize the available spectrum more efficiently [8].

In the quest for a truly agile, resource-efficient optical network, the routing and spectrum assignment (RSA) problem has emerged as the key network design and control problem [9], [14]. 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). 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) tradeoff [3], traffic grooming [17], and restoration [15]. These problem variants are NP-hard, as RSA is a generalization of the well-known routing and wavelength assignment (RWA) problem [11]. 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 [10]. For a survey of spectrum management techniques in elastic optical networks, including a review of solution approaches to RSA problem variants, we refer the reader to [13].

A recent study [12] considered the complexity of the RSA problem in chain (path) networks. In path networks, the routing aspect of the problem is completely determined, and it reduces to a spectrum allocation (SA) problem. Using results from graph coloring theory, it was shown in [12] that the SA problem in paths 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 $(4+2\epsilon)$.

The main contribution of our work is to provide new insight into the spectrum assignment problem in mesh networks by showing in Section II that it transforms to the problem of scheduling multiprocessor tasks on dedicated processors [1], [6]. Based on this new perspective, we show in Section III that the SA problem in paths 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 paths when the number of links is fixed, 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. The Spectrum Assignment (SA) Problem

We consider the following basic definition of the routing and spectrum assignment (RSA) problem.

Definition 2.1 (RSA): Given a graph $G = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{A}$ 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

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 No. 2-611-1434-HiCi.

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 non-overlapping parts of the available spectrum (non-overlapping 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 spectrum assignment (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 [1], [6]:

Definition 2.3 ($P|fix_j|C_{max}$): Given a set of n tasks and a set of identical processors, a procesing 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.

It has been shown [6] 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 problem variants [2].

We have the following result.

Lemma 2.1: SA transforms to $P|fix_j|C_{max}$.

Proof. Consider an instance of SA on graph $G = (\mathcal{V}, \mathcal{A})$, matrix $T = [t_{sd}]$, and path set $\{r_{sd}\}$. Construct an instance of $P|fix_j|C_{max}$ such that for each arc $a_k \in \mathcal{A}$, there is a processor k, and for each spectrum demand t_{sd}, there is a task j with $p_j = t_{sd}$ and $fix_j = \{k : a_k \in r_{sd}\}$. Hence, 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 non-overlapping 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, whereas 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 $P|fix_j|C_{max}$. ■

Since any instance of the SA problem can be transformed into an instance of the $P|fix_j|C_{max}$ problem, an algorithm for 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. Consider an instance of $P4|fix_j|C_{max}$ with five tasks (the processing times of the tasks 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 rings with as few as three links is NP-hard.

Lemma 2.3: The SA problem in 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 [6], the same is true for the SA problem on the three-link ring. ■

A. The SA Problem in Paths

In path (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 processors are labeled $1, 2, 3, \ldots$, 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 directed path with m links is equivalent to $Pm|line_j|C_{max}$, hence an algorithm for solving $Pm|line_j|C_{max}$ may be used to solve SA, and vice versa.

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

Proof. First consider an instance of the SA problem on a directed path G with $m+1$ nodes labeled $1, 2, \ldots, m+1$, and m arcs $a_1 = <1, 2>$, $a_2 = <2, 3>, \ldots, a_m < m, m+1>$, 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 $Pm|line_j|C_{max}$ since the sets fix_j consist of processors with consecutive labels.

Given an instance of $Pm|line_j|C_{max}$, we construct an instance of the SA problem on a path 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 $a_k = <k, k+1>$. For each task j with $line_j = \{s, \ldots, d-1\}$, there is a traffic demand with $t_{sd} = p_j$ and route $r_{sd} = \{<s, s+1>, \ldots, <d-1, d>\}$. It is not difficult to verify that the three constraints of $Pm|line_j|C_{max}$ 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. ■

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 [6], 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 [12] that the SA problem in paths is NP-hard, this result implies that the problem is in fact polynomial on three-link paths.

Lemma 3.1: $P3|line_j|C_{max}$ is 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 where either processor 2 or processor 3 are 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 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 procesor is never idle, and hence the makespan $C_{max} = t$, the time required to execute the tasks on the dominant processor. ■

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 [4] which is defined as:

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 $\sum_{a_j \in A_1} a_j = \sum_{j \in A_2} a_j = \frac{B}{2}$?

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

Proof. Given an instance of PARTITION, we create an instance of $P_4|line_j|C_{max}$ 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 the following figure and a get a feasible schedule with $C_{max} = 4B$.

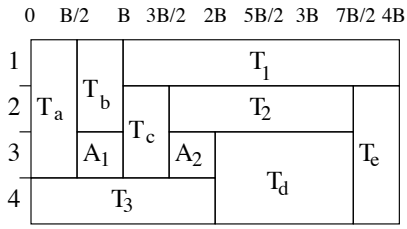

Fig. 1. Feasible schedule with $C_{max} = 4B$

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, T_3 must be 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 Figure 1. Moreover, tasks T_a and T_b must be scheduled before T_1, while T_c must be before T_2 for the schedule length to be at most $4B$. If T_a is scheduled before T_b, 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 $[0, B/2]$ and $[3B/2, 2B]$ are available for the PARTITION jobs. In both cases, a partition exists. ■

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

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 [7] 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 intoduce a two-stage algorithm for $Pm|line_j|C_{max}$, and show that it yields a constant approximation ratio for any fixed number of processors $m \geq 6$. The algorithm 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.

Lemma 4.3: Let α_{mid}, α_{lo}, and α_{hi} be the approximation ratio 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 is used). Then the two-stage algorithm is an approximation algorithm for the original problem with ratio $\alpha = \alpha_{mid} + \max\{\alpha_{lo}, \alpha_{hi}\}$.

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. ■

Figure 2 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 2-approximation algorithm for $m = 9$ processors (and also for $m = 6, 7, 8$).

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 tasks 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 tasks 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 Figure 2. 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.

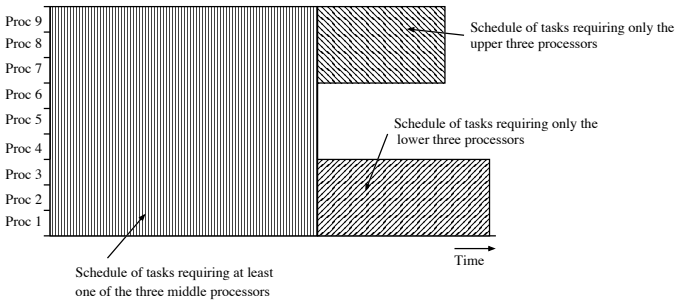

Fig. 2. 2-approximation schedule for $m = 9$ processors

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. 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 multi-processor 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 are said to be *compatible* if and only if their prespecified sets of processors are pairwise disjoint, i.e., $line_j \cap line_i = \emptyset, \forall\ i, j \in \mathcal{T}$.

We present two classes of algorithms that differ in the granularity at which they make scheduling decisions. The first class of algorithms assemble compatible tasks into *blocks*, and schedule a whole block of tasks at a time. The second class of algorithms operate at finer granularity and make 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: (1) all tasks in a block begin execution at the same time t, and (2) 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. The running time of the algorithm is $O(n^2)$.

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 (1) the start time of the schedule (i.e., $t = 0$), and (2) 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. 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.

VI. Numerical Results

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 paths of length typical of a commercial wide area network[1]. For each problem instance, we generated traffic demands between every source and destination node on the path. The size of the traffic demands (i.e., task times) were generated using four 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.
- *Continuous uniform:* traffic demands may take any (integer) value in the interval $[10, 1000]$ with equal probability.

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.

Figures 3-6 plot the average ratio achieved by the four list scheduling algorithms, LFB, LFC, WFB, and WFC, for each of the four 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 path) 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 algorithms is not sensitive to the number of processors (path 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 time, demonstrating the importance of taking into consideration the idle times in the scheduling process. Overall, these results indicate that, for problem instances representative of spectrum allocation prob-

[1]We have also carried out experiments with problem instances consisting of as many as $m = 10,000$ processors. Although we cannot include the results due to page constraints, the performance of the algorithms on these instances is similar or better than on the instances considered here.

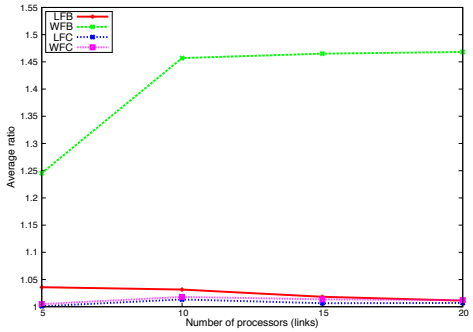

Fig. 3. Average ratio vs. number of processors, discrete uniform distribution

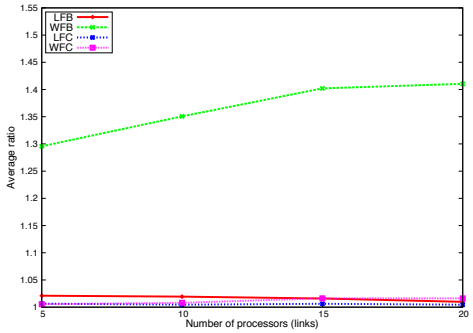

Fig. 4. Average ratio vs. number of processors, discrete high distribution

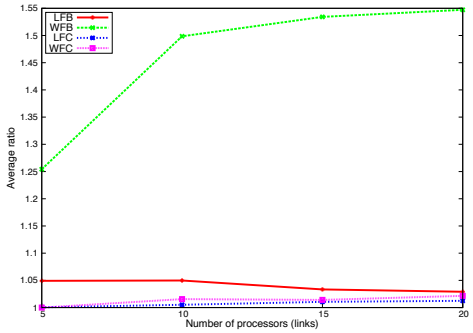

Fig. 5. Average ratio vs. number of processors, discrete low distribution

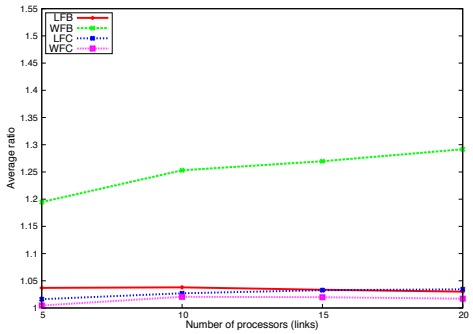

Fig. 6. Average ratio vs. number of processors, continuous uniform distribution

lems arising in paths 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.

VII. Concluding Remarks

We have studied the spectrum assignment (SA) problem in elastic optical networks 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 path 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 path networks.

References

[1] E. Bampis, M. Caramia, J. Fiala, A. Fishkin, and A. Iovanella. Scheduling of independent dedicated multiprocessor tasks. In *Proceedings of the 13th Annual International Symposium on Algorithms and Computation*, volume LNCS 2518, pages 391–402, 2002.

[2] E. Bampis and A. Kononov. On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements. In *Proceedings of IPDPS 2001*, San Francisco, 2001.

[3] K. Christodoulopoulos, I. Tomkos, and E.A. Varvarigos. Elastic bandwidth allocation in flexible OFDM–based optical networks. *Journal of Lightwave Technology*, 29(9):1354–1366, 2011.

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

[5] O. Gerstel, M. Jinno, A. Lord, and S. J B Yoo. Elastic optical networking: a new dawn for the optical layer? *IEEE Communications Magazine*, 50(2):s12–s20, 2012.

[6] J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discree Applied Mathematics*, 55:259–272, 1994.

[7] J. Huang, J. Chen, S. Chen, and J. Wang. A simple linear time approximation algorithm for multi-processor job scheduling on four processors. *Journal of Combinatorial Optimization*, 13:33–45, 2007.

[8] M. Jinno, H. Takara, and B. Kozicki. Dynamic optical mesh networks: Drivers, challenges and solutions for the future. In *Proceedings of 35th European Conference on Optical Communication (ECOC)*, page 7.7.4, September 2009.

[9] M. Klinkowski and K. Walkowiak. Routing and spectrum assignment in spectrum sliced elastic optical path network. *IEEE Communications Letters*, 15(8):884–886, 2011.

[10] Z. Liu and G. N. Rouskas. A fast path-based ILP formulation for offline RWA in mesh optical networks. In *Proceedings of IEEE GLOBECOM 2012*, December 2012.

[11] G. N. Rouskas. Routing and wavelength assignment in optical WDM networks. In *J. Proakis (Editor), Wiley Encyclopedia of Telecommunications*. John Wiley & Sons, 2001.

[12] S. Shirazipourazad, Ch. Zhou, Z. Derakhshandeh, and A. Sen. On routing and spectrum allocation in spectrum-sliced optical networks. In *Proceedings of IEEE INFOCOM*, pages 385–389, April 2013.

[13] S. Talebi, F. Alam, I. Katib, M. Khamis, R. Khalifah, and G. N. Rouskas. Spectrum management techniques for elastic optical networks. 2013. Submitted for Publication.

[14] Y. Wang, X. Cao, and Y. Pan. A study of the routing and spectrum allocation in spectrum-sliced elastic optical path networks. In *Proceedings of IEEE INFOCOM*, pages 1503–1511, 2011.

[15] Y. Wei, G. Shen, and Sh. You. Span restoration for CO-OFDM-based elastic optical networks under spectrum conversion. In *Proceedings of ACP 2012*, page AF3E.7, Novomber 2012.

[16] G. Zhang, M. De Leenheer, A. Morea, and B. Mukherjee. A survey on OFDM-based elastic core optical networking. *IEEE Communications Surveys & Tutorials*, 15(1):65–87, First Quarter 2013.

[17] Y. Zhang, X. Zheng, Q. Li, N. Hua, Y. Li, and H. Zhang. Traffic grooming in spectrum-elastic optical path networks. In *Proceedings of OFC/NFOEC 2011*, page OTuI1, March 2011.