8

Scheduling-Inspired Spectrum Assignment Algorithms for Mesh Elastic Optical Networks

Mahmoud Fayez[3], Iyad Katib[1], George N. Rouskas[1,2] and Hossam M. Faheem[3,4]

[1]King Abdulaziz University, Jeddah, Saudi Arabia
[2]North Carolina State University, Raleigh, NC 27695, United States
[3]Fujitsu Technology Solution, Munich, Germany
[4]Ain Shams University, Khalifa El-Maamon St‘ Cairo, Cairo

Abstract

Spectrum assignment has emerged as the key design and control problem in elastic optical networks. We have shown that spectrum assignment in networks of general topology is a special case of scheduling multiprocessor tasks on dedicated processors. Based on this insight, we develop and evaluate efficient and effective algorithms for mesh and chain networks that build upon list scheduling concepts.

8.1 Introduction

Optical networking technologies are crucial to the operation of the Internet and its ability to support critical and reliable communication services. In response to rapidly growing IP traffic demands, 40 and 100 Gbps line rates over long distances have been deployed, while there is a substantial research and development activity targeted to commercializing 400 and 1000 Gbps rates [1]. On the other hand, emerging applications, including IPTV, video-on-demand, and interdatacenter networking, have heterogeneous bandwidth demand granularities that may change dynamically over time. Accordingly, it has been proposed that mixed line rate (MLR) networks [2] may be able

to accommodate variable traffic demands. Nevertheless, optical networks operating on a fixed wavelength grid [3] allocate a full wavelength even to traffic demands that do not fill its entire capacity [4]. This inefficient utilization of spectral resources is expected to become an even more serious issue with the deployment of higher data rates [5, 6].

Elastic optical networks [7, 8] 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 [6]. The enabling technology for such an agile network infrastructure is orthogonal frequency division multiplexing (OFDM), a modulation format that has been widely adopted in broadband wireless and copper-based communication systems, and is a promising candidate for high-speed (i.e., beyond 100 Gbps) optical transmission [9]. Other key technologies include distance-adaptive modulation, bandwidth-variable transponders, and flexible spectrum selective switches; for a recent survey of optical OFDM and related technologies, refer to [9].

OFDM, a multiple carrier modulation scheme, splits a data stream into a large number of substreams [10]. Each data substream is carried on a narrowband subchannel created by modulating a corresponding carrier with a conventional scheme such as quadrature amplitude modulation (QAM) or quadrature phase shift keying (QPSK). The modulated signals are further multiplexed by frequency division multiplexing to form what is referred to as multicarrier transmission. The composite signal is a broadband signal that is more immune to multipath fading (in wireless communications) and intersymbol interference. The main feature of OFDM is the orthogonality of subcarriers that allows data to travel in parallel, over subchannels constituted by these orthogonal subcarriers, in a tight frequency space without interference from each other. Consequently, OFDM has found many applications, including in ADSL and VDSL broadband access, power line communications, wireless LANs (IEEE 802.11 a/g/n), WiMAX, and terrestrial digital TV systems.

In recent years, OFDM has been the focus of extensive research efforts in optical transmission and networking, initially as a means to overcome physical impairments in optical communications [11, 12]. However, unlike, with wireless LANs or xDSL systems where OFDM is deployed as a transmission technology in a *single link*, in optical networks it is considered as the technology underlying the novel elastic network paradigm [6]. Consequently,

in the quest for a truly agile, resource-efficient optical infrastructure, *network-wide spectrum management* arises as a key challenge; the routing and spectrum assignment (RSA) problem has emerged as an essential network design and control problem [13, 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 takes into account various design aspects including the reach versus modulation level (spectral efficiency) trade-off [15], traffic grooming [16], and restoration [17]. These problem variants are NP-hard, as RSA is a generalization of the well-known routing and wavelength assignment (RWA) problem [18]. 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; hence, they cannot be easily adapted to other problem variants. Second, it is quite difficult to characterize the performance of heuristic algorithms; our recent work has demonstrated that heuristics for the related RWA problem produce solutions that are far from optimal even for problem instances of moderate sizes [19]. For a survey of spectrum management techniques in elastic optical networks, including a review of solution approaches to RSA problem variants, we refer to our recent survey [20].

When the path of each demand is provided as part of the input and is not subject to optimization, the RSA problem reduces to the spectrum assignment (SA) problem. The main contribution of this work is that we build upon well-understood scheduling theory techniques to develop efficient and effective algorithms for the SA problem in optical networks. We consider two types of networks: *mesh networks* of general topology, which are representative of commercial wide area networks, and *chain (linear) networks* consisting of an acyclic sequence of nodes connected in a linear chain. Our results are important because (1) the SA algorithms for mesh networks may be used to tackle the RSA problem by applying them in parallel to different routing configurations (which are independent of each other) and selecting the best solution; (2) the SA algorithms for chain networks may be used to analyze approximately large networks of general topology, e.g., by extending path-based decomposition approaches that we have developed for the case of

wavelength assignment [21]; and (3) these algorithms may be applied to large-scale task scheduling problems in multiprocessor environments.

Following the introduction, we review our earlier work in Section 8.2 and provide insight into the properties and structure of the spectrum assignment problem as a special case of a general multiprocessor scheduling problem, in which a task must be executed by multiple machines simultaneously. In Section 8.3, we develop algorithms for multiprocessor scheduling, which can be used to solve the corresponding spectrum assignment problem; we present two algorithms—one for general topology (mesh) networks and a faster one for chain networks. We present numerical results to evaluate the performance of the algorithms in Section 8.4 and conclude the chapter in Section 8.5.

8.2 SA in Mesh Networks: A Special Case of Multiprocessor Scheduling

We consider the following general definition of the spectrum assignment (SA) problem in elastic optical networks:

- *SA Inputs*: (1) A graph $G = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{A}$ is the set of arcs (directed edges); (2) a spectrum demand matrix $T = [t_{sd}]$, such that t_{sd} is the number of spectrum slots required to carry the traffic from source s to destination d; and (3) a fixed route r_{sd} from node s to node d.
- *SA Objective*: For each traffic demand, assign spectrum slots along all the arcs of its route such that the total required amount of spectrum used on any arc in the network is minimized.
- *RSA Constraints*: (1) Spectrum contiguity: each demand is assigned contiguous spectrum slots; (2) spectrum continuity: each demand uses the same spectrum slots along all arcs of its route; and (3) non-overlapping spectrum: demands that share an arc are assigned non-overlapping parts of the available spectrum.

Now, consider the multiprocessor scheduling problem $P|fix_j|C_{max}$, defined as [22]:

- $P|fix_j|C_{max}$ *Inputs*: A set of m identical processors, a set of n tasks, the processing time p_j of task j, and a set fix_j of processor sets that will execute each task j.
- $P|fix_j|C_{max}$ *Objective*: Schedule the tasks so as to minimize the makespan $C_{max} = \max_j C_j$ of the schedule, where C_j indicates the completion time of task j.

- $P|fix_j|C_{max}$ *Constraints*: (1) No preemption is allowed, (2) all the processors in the selected set must work on task j simultaneously, and (3) each processor may execute one task at most at any given time.

In earlier work [23], we have proved that the SA problem in mesh networks transforms to the $P|fix_j|C_{max}$ multiprocessor scheduling problem; however, the reverse is not true. In other words, SA is a special case of $P|fix_j|C_{max}$; hence, any algorithm for the latter problem may also solve the former. A formal proof of the transformation is available in [23]. In the transformation, the various entities of the elastic network domain transform to entities in the task scheduling domain as shown in Table 8.1. Specifically, each arc in the SA problem maps to a processor in the scheduling problem, each traffic demand to a task, the number of spectrum slots of a demand to the processing time of the corresponding task, and the maximum number of spectrum slots used on any link to the makespan of the schedule. Accordingly, minimizing the maximum spectrum allocation on any arc of the SA problem is equivalent to minimizing the makespan of the schedule in the corresponding problem $P|fix_j|C_{max}$. Furthermore, the spectrum contiguity constraint of SA is equivalent to the no preemption constraint of $P|fix_j|C_{max}$; the spectrum continuity constraint maps to the constraint that all required processors must execute a task simultaneously; and the non-overlapping spectrum constraint maps to the constraint that a processor works on at most one task at a time.

8.2.1 Illustrative Example

As an example, Figure 8.1(a) shows an instance of the SA problem on a mesh network with five directed links, $L1, L2, L3, L4$, and $L5$. There are six demands (shown as dotted lines) with the number of slots required by each demand shown next to the corresponding line. These demands are the inputs to the SA problem and are also listed in the left three columns of Table 8.2, which provides the transformation from traffic demands and corresponding paths in the elastic optical network domain to tasks and sets of processors in the task scheduling domain under the assumption that link $L1$ maps to processor

Table 8.1 Transformation of entities between the elastic network and task scheduling domains

Elastic Network Domain	Task Scheduling Domain
Arc	Processor
Traffic demand	Task
Spectrum slots of a demand	Processing time of a task
Path of a demand	Set of processors for a task
Highest assigned slot on an arc	Completion time of a processor

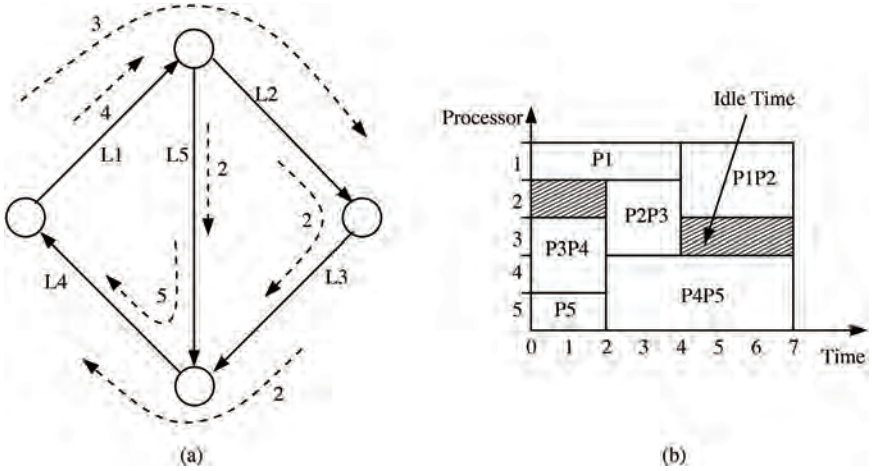

Figure 8.1 (a) Instance of the SA problem on a mesh network with five directed links (arcs). (b) Optimal schedule of the corresponding $P|fix_j|C_{max}$ problem.

Table 8.2 Mapping from the elastic network domain to the task scheduling domain

Elastic Network Domain			Task Scheduling Domain		
Demand #	Path	Spectrum Slots	Task #	Processors	Time Slots
1	L1	4	1	P1	4
2	L1, L2	3	2	P1,P2	3
3	L5	2	3	P5	2
4	L5, L4	5	4	P5, P4	5
5	L2, L3	2	5	P2, P3	2
6	L3, L4	2	6	P3, P4	2

$P1$, link $L2$ to processor $P2$, and so on. Figure 8.1(b) shows the optimal schedule for the $P|fix_j|C_{max}$ problem corresponding to this SA instance. As we can see, the demand of size 3 that follows the path $L1$–$L2$ is mapped to a task that is scheduled in the time interval $[4, 7]$ on the corresponding processors $P1$ and $P2$, similarly for the other demands. The schedule is optimal in that $C_{max} = 7$ is equal to the total processing time required for processors $P1, P4$, and $P5$. Also, the value of C_{max} is equal to the total number of spectrum slots required for links $L1, L4$, and $L5$.

8.2.2 Complexity Results

It has been shown [22] that the three-processor problem $P3|fix_j|C_{max}$ is strongly NP-hard for general processing times, but 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 (PTASs) have been developed for several versions of the problem [24].

By building upon this new perspective, it was shown in [23] that (1) $P3|fix_j|C_{max}$ transforms to the SA problem in a unidirectional ring with three links; hence, the latter is NP-hard, and (2) the SA problem is solvable in polynomial time on chain networks with at most three links, but is NP-hard on chains with four or more links. The latter result confirms the conclusion in [25] that the SA problem is harder than the wavelength assignment problem which can be solved in polynomial time on chains of any length. In [23], we also developed a set of list scheduling algorithms specifically designed for the SA problem in chains; the algorithms are both fast and effective, in that they produce solutions that are within 5% of the lower bound on average.

8.3 Scheduling Algorithms for Spectrum Assignment in Mesh Networks

We now present a new efficient scheduling algorithm for the $P|fix_j|C_{max}$ problem, which may also be used to solve the SA problem in mesh networks. The input to the algorithm is a list of n tasks, $j = 1, \ldots, n$, along with their corresponding processing times, p_j, and sets of processors, fix_j. Tasks in the list may be sorted in differently; in this work, we consider and compare the following two distinct orders:

- *Longest First (LF)*: Tasks appear in the list in decreasing order of their processing time p_j.
- *Widest First (WF)*: Tasks are listed in decreasing order of the size $|fix_j|$ of their processor set.

In either case, we assume that ties are broken arbitrarily. We also define two processor sets as *compatible* if they are disjointed, in which case the two processor sets can be scheduled simultaneously.

Figure 8.2 presents a pseudocode description of the scheduling algorithm for the $P|fix_j|C_{max}|$ problem. Depending on whether the tasks are listed in longest first or widest first order, we will refer to the scheduling algorithm as *SA-LF* or *SA-WF*, respectively[1].

[1] In this notation, we use the acronym "SA" to emphasize both that this is a *scheduling algorithm* for the $P|fix_j|C_{max}$ problem and the fact that this algorithm solves the *spectrum assignment* problem.

Scheduling Algorithm (SA-LF/WF) for $P|fix_j|C_{max}$
Input: A list L of n tasks on m processors, each task j having a processing time p_j and a set $fix_j \subseteq \{1, 2, \ldots, m\}$ of required processors
Output: A schedule of tasks, i.e., the time S_j when each task j starts execution on the multi-processor system

begin
1. Sort the tasks in list L based on longest-first or widest-first criteria
2. $L_p[1, \ldots, n] \leftarrow false$ //The list of in-progress tasks
3. $t \leftarrow 0$ //Scheduling instant
4. $F_p \leftarrow m$ //Counter of free processors
5. $Counter \leftarrow 0$ //Counter of finished tasks
6. $F[1, \ldots, m] \leftarrow true$ //The list of idle (free) processors
7. **while** $Counter \neq n$ **do**
8. $j \leftarrow 0$
9. ScheduleTasks(L, L_p, F, t, j)
10. AdvanceTime($L_p, F, t, Counter$)
11. **end while**
12. **return** the task start times S_j

end

Procedure ScheduleTasks(L, L_p, F, t, j)
Operation: Schedules as many tasks from the input list L to start execution at time t, and moves these tasks from L to the list of in-progress tasks L_p

begin
1. **while** $j \neq n$ **and** $F_p > 0$ **do**
2. **if** $L_{p_j} = false$ **and** $F_{fix_j} = true$ **then**
3. $S_j \leftarrow t$ // Task j starts execution at time t
4. $L_{p_j} = true$
5. $F_{fix_j} = false$
6. $F_p \leftarrow F_p - count(fix_j)$
7. ScheduleTasks($L, L_p, F, t, j + 1$)
8. **break**
9. **endif**
10. $j \leftarrow j + 1$
11. **end while** //no more tasks may start at time t

end

Procedure AdvanceTime($L_p, F, t, Counter$)
Operation: Finds the first task or tasks to complete after time t, removes them from the list of in-progress tasks, and advances time to the time these tasks end

begin
1. $j \leftarrow 0$
2. $j_{min} \leftarrow$ –1 //Index of earliest task to finish
3. $t_{min} \leftarrow \infty$ //the default value to find minimum finish time
4. **while** $j \neq n$ **do**
5. **if** $L_{p_j} = true$ **and** $S_j + p_j > t$ **and** $S_j + p_j < t_{min}$ **then**
6. $j_{min} \leftarrow j$
7. $t_{min} \leftarrow S_j + p_j$
8. **endif**
8. $j \leftarrow j + 1$
9. **end while**
10. $F_{fix_{j_{min}}} \leftarrow true$ //Set processors to free again
11. $F_p \leftarrow F_p + count(fix_{j_{min}})$
12. $t \leftarrow t_{min}$ //Advance time

end

Figure 8.2 The scheduling algorithm SA-LF/WF for $P|fix_j|C_{max}$ and the corresponding spectrum assignment problem.

The algorithm maintains an array of m Booleans to keep track of the set of free processors, initialized to all processors, and a list of in-progress tasks initialized to the empty set. Initially, the tasks in the input list L are sorted by the longest first or widest first order. The algorithm then repeatedly calls two procedures, *ScheduleTasks()* and *AdvanceTime()*, until all the tasks in list L have been scheduled (i.e., until L becomes empty).

The *ScheduleTasks()* procedure takes the list L of unscheduled tasks, the list L_p of in-progress tasks, and the set F of free processors at time t as arguments. It then considers tasks in L one at a time in an attempt to schedule them starting at time t; note that a task j can be scheduled if all processors in fix_j are free at time t; i.e., it is pairwise compatible with all in-progress tasks. Every task that can be scheduled is marked as running in list L_p of in-progress tasks. This process continues until either the end of list L is reached or all processors become busy. At that point, procedure *AdvanceTime()* is called. This procedure finds the first in-progress task that ends and advances the time

to the time t' this task ended. It then frees all involved processors for this task. Consequently, the procedure *ScheduleTasks()* is called again to schedule any remaining tasks starting at time t', and this process repeats until all tasks have been scheduled.

Each of the two procedures *ScheduleTasks()* and *AdvanceTime()* is called n times in the worst case, where n is the number of tasks. In worst-case scenarios, the *ScheduleTasks()* will consider all n tasks in list L, and for each task, it will check whether its processor set is a subset of the set F of free processors (line 2), and if so, it will remove these processors from the set (line 5); these operations take time $O(m)$ in the worst case, where m is the number of processors. Since all other operations are constant, the running time of this procedure is $O(nm)$. Procedure *AdvanceTime()* checks all in-progress tasks to identify the ones with minimum finish time and frees their processors; therefore, its worst-case running time is $O(m + n)$. Therefore, the overall running time complexity of the algorithm is $O(mn^2)$.

8.3.1 Scheduling Algorithm for Chain Networks

In the special case of chain networks, the corresponding scheduling problem is such that the m processors, each corresponding to a link of the chain, can be labeled linearly as $1, \ldots, m$. Furthermore, the processors required by each task are contiguous and may be represented as a range, a fact that has two implications. First, checking whether a task's processors are free at some time t can be performed in constant time, rather than in time $O(m)$ as in the general case of mesh networks. Second, assigning processors to a task naturally divides the previous free range of processors into (at most) two parts: one part consisting of free processors with labels smaller than that of the processor with the lowest label required by the task and one consisting of free processors with labels larger than that of the processor with the highest label required by the task. Therefore, it is possible to schedule tasks by recursively searching the (at most) two free ranges of processors created when a task is scheduled.

The recursive version of *ScheduleTasks()* for chain networks is shown in Figure 8.3. Since, as we mentioned above, the operations in Steps 2 and 5 now take constant time, the worst-case running time of the procedure for chain networks is $O(n)$. Therefore, the overall running time of the scheduling algorithm is $O(n^2)$ and is independent of the number m of processors (or links of the chain).

Procedure ScheduleTasks(L, L_p, F, t)
Operation: Schedules as many tasks from the input list L to start execution at time t, and moves these tasks from L to the list of in-progress tasks L_p

begin
begin
1. **while** $j \neq n$ **and** $F_p > 0$ **do**
2. **if** $L_{p_j} = false$ **and** $F_{fix_j} = true$ **then**
3. $S_j \leftarrow t$ // Task j starts execution at time t
4. $L_{p_j} = true$
5. $F_{fix_j} = false$
6. $F_p \leftarrow F_p - count(fix_j)$
8. $F_l \leftarrow$ new left range of free processors
9. $F_r \leftarrow$ new right range of free processors
7. ScheduleTasks($L, L_p, F_l, t, j + 1$)
7. ScheduleTasks($L, L_p, F_r, t, j + 1$)
8. **break**
9. **endif**
10. $j \leftarrow j + 1$
11. **end while** // no more tasks may start at time t

end

Figure 8.3 A specialized version of the *ScheduleTasks()* procedure for the $P|fix_j|C_{max}$ problem corresponding to a chain network.

8.4 Numerical Results

In this section, we present the results of simulation experiments we carried out to evaluate the performance of the scheduling algorithms for mesh and chain networks. We assume that the elastic optical network supports the following data rates (in Gbps): 10, 40, 100, 400, and 1000. For each problem instance, we generate random traffic rates between each pair of nodes based on one of the three distributions:

1. *Uniform*: Traffic demands may take any of the five discrete values in the set $\{10, 40, 100, 400, 1000\}$ with equal probability;
2. *Skewed 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 (i.e., the lower data rates have higher probability to be selected); or

3. *Skewed 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 (i.e., the higher data rates have higher probability to be selected).

In our experiments, we also used various other probability values the skewed low and high distributions, but the results are very similar to those shown below.

Once the traffic rates between every source–destination pair have been generated, we calculate the corresponding spectrum slots as follows. We assume that the slot width is 12.5 GHz and that there is a 16-QAM modulation format, such that demands of size 10, 40, 100, 400, and 1000 Gbps require 1, 1, 2, 8, and 20 slots, respectively, consistent with the values used in [26, Table 8.1]. We then transform the SA problem instance to the equivalent instance of $P|fix_j|C_{max}$ and run the scheduling algorithm described in the previous section to schedule the tasks.

In order to evaluate the performance of our algorithms, and since the optimal solution is not known due to the fact that $P|fix_j|C_{max}$ is NP-complete, we compute the lower bound as follows. Consider an instance of $P|fix_j|C_{max}$, and let T_k denote the set of tasks that require processor k, i.e., $T_k = \{j : k \in fix_j\}$. Clearly, all the tasks in T_k are pairwise incompatible; hence, they have to be executed sequentially. Let Π_k denote the sum of processing times of tasks that require processor k:

$$\Pi_k = \sum_{j \in \mathcal{T}_k} p_j, \; k = 1, \ldots, m. \tag{8.1}$$

Then, a lower bound LB for the problem instance can be obtained as follows:

$$LB = \max_{k=1,\ldots,m} \{\Pi_k\}. \tag{8.2}$$

We then compute the ratio of the makespan produced by the algorithm to this lower bound. It should be noted that the lower bound is not tight, as it ignores any gaps introduced by the scheduling of incompatible tasks in the optimal solution.

Each point in the figures shown in the remainder of this section is the average of 200 randomly generated problem instances for the specified parameters. We also estimated confidence intervals using the method of batch means, but they are so narrow that they are omitted. All the experiments were carried out using the resources of the high-performance computing cluster installed by Fujitsu at King Abdulaziz University, Jeddah, Saudi Arabia.

8.4.1 Mesh Networks

We have carried out simulation experiments on three network topologies[2]: (1) a 10-node, 32-link network; (2) the 32-node, 108-link network shown in Figure 8.4; and (3) a 75-node, 200-link topology. For each network topology, we generate random traffic demands between each source–destination pair[3] using each of the three distributions described earlier. Each demand is routed over the shortest path between its source and destination nodes.

The results for the 10-node network are shown in Figure 8.5. In four problem instances, the SA-LF algorithm found solutions with a makespan of about 10% higher than the lower bound, while for the remaining instances, the solutions constructed by the algorithm had a makespan equal to the lower bound, and hence, they are optimal. For the 32- and 75-node networks, the SA-LF produced optimal solutions for all three traffic distributions and all random problem instances we generated.

Figures 8.6 and 8.7 show the performance of the SA-LF and SA-WF algorithms, respectively, on complete mesh networks of varying sizes, in which traffic between each pair of nodes is routed over a randomly chosen path.

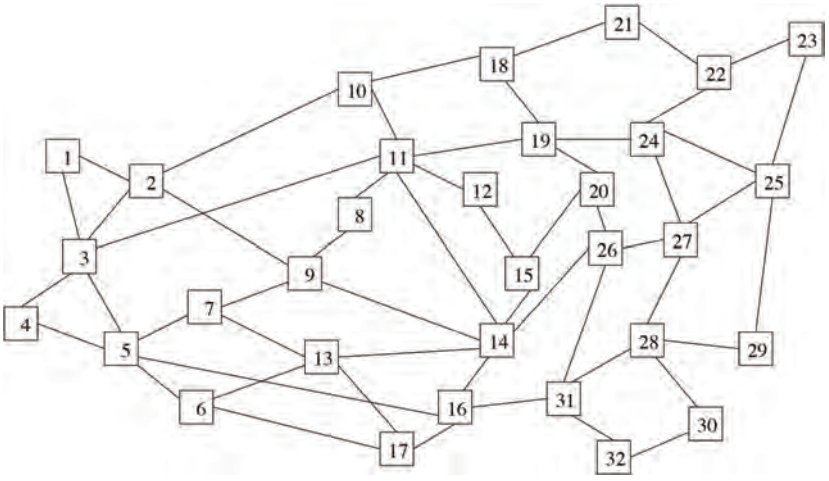

Figure 8.4 The 32-node, 108-link topology used in the experiments.

[2]The number of links of each network topology refers to directional links (arcs) since each direction of a link is considered independently for the purpose of spectrum assignment.

[3]The number of traffic demands for the 10-node, 32-node, and 75-node networks are 90, 992, and 5550, respectively.

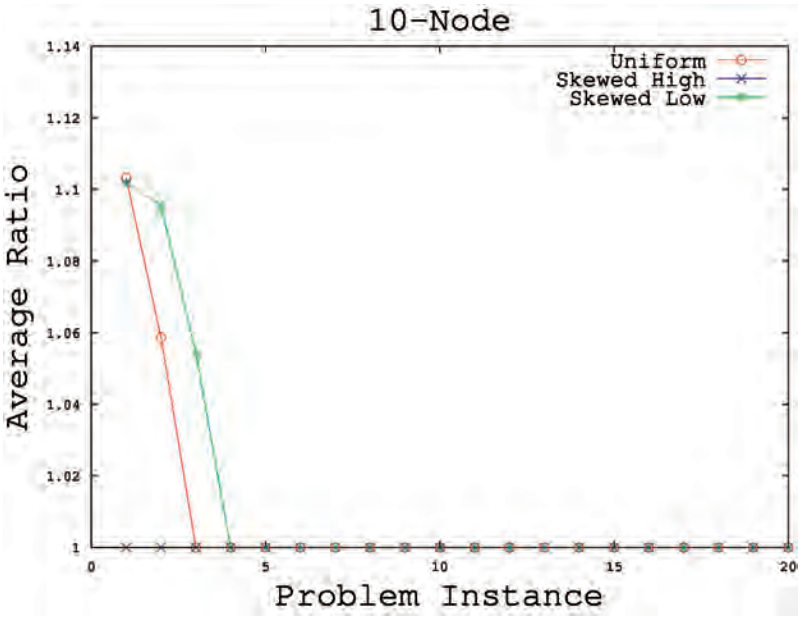

Figure 8.5 Average ratio of makespan to lower bound for 10-node network.

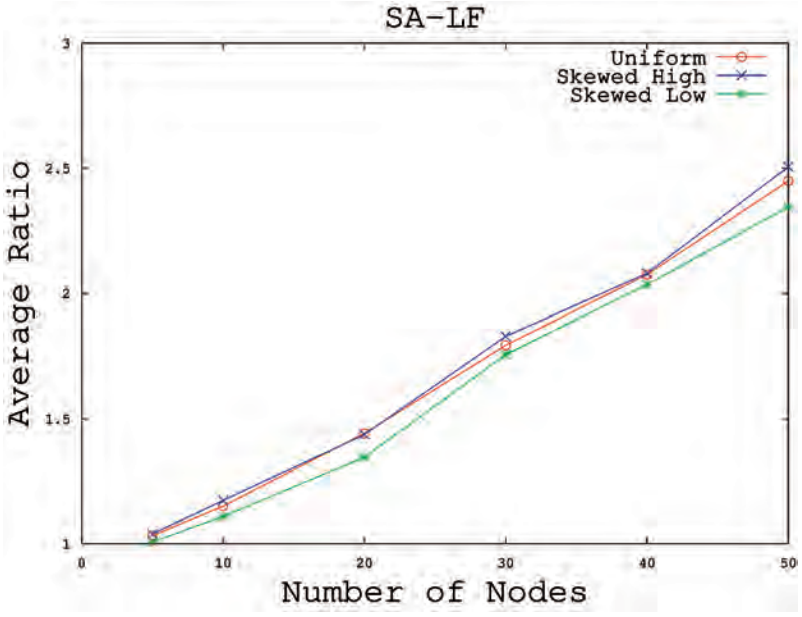

Figure 8.6 Average ratio of makespan to lower bound, complete mesh with SA-LF.

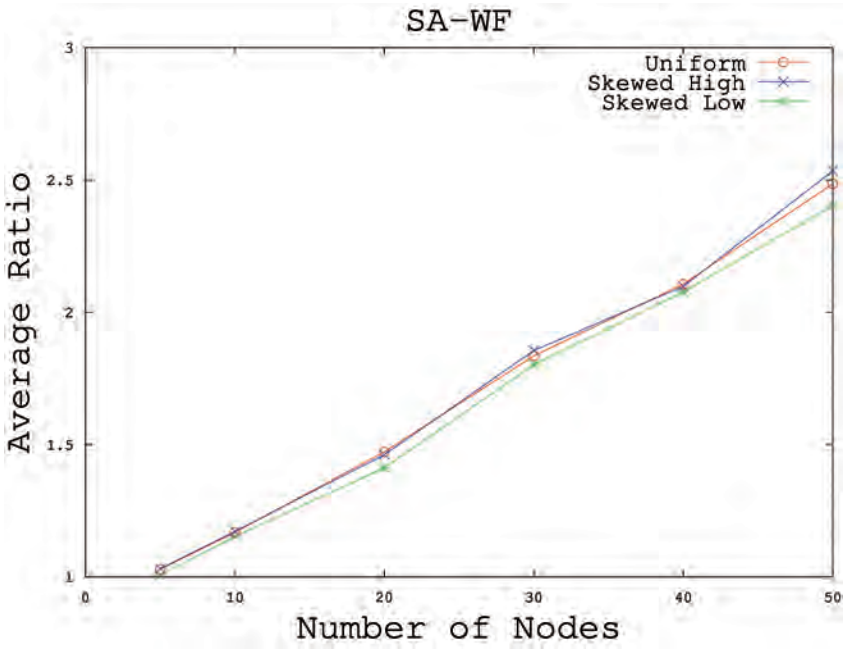

Figure 8.7 Average ratio of makespan to lower bound, complete mesh with SA-WF.

The performance of the two algorithms is similar and is not affected significantly by the traffic distribution (uniform, skewed high, or skewed low). The reason that the ratio of makespan to lower bound increases with the size of the network is due to the fact that, for a complete mesh, the optimal solution would be for each demand to take the single-link shortest path to its destination. However, since each demand is routed along a randomly selected path, and the lengths of these random paths increase with the size of the network, such a solution will move further from the lower bound (optimal) as the network size increases. The performance of the algorithms in the two figures simply reflects this observation.

8.4.2 Chain Networks

Figures 8.8 and 8.9 show results for chain networks of varying sizes and the SA-LF and SA-WF algorithms, respectively. We observe that the SA-LF algorithm performs better for all three traffic distributions and produces results that are within 5% of the lower bound. The SA-WF algorithm, which considers tasks for scheduling based on the number of processors they require,

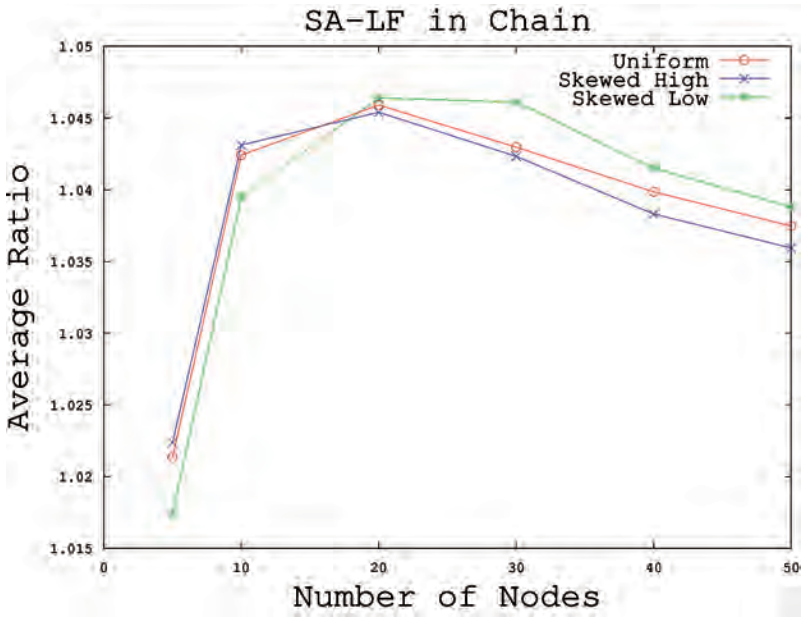

Figure 8.8 Average ratio of makespan to lower bound, chain networks with SA-LF.

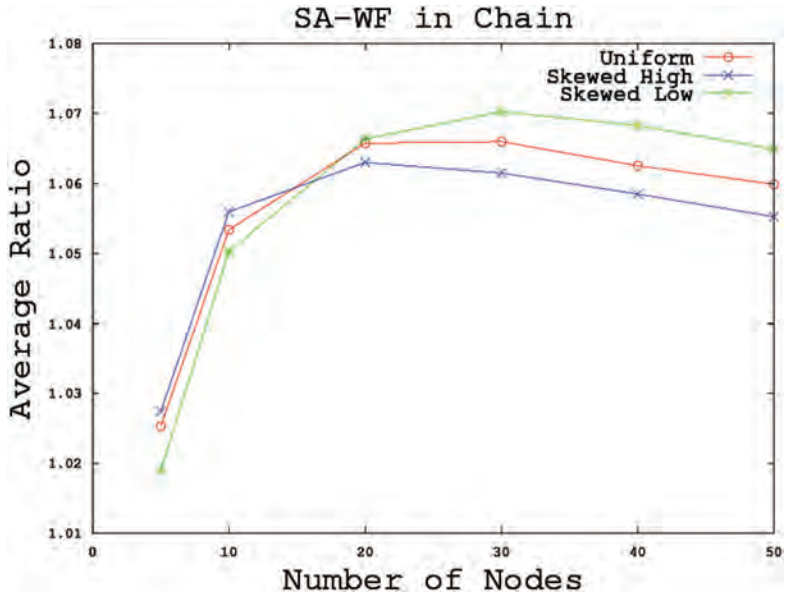

Figure 8.9 Average ratio of makespan to lower bound, chain networks with SA-WF.

may pair long tasks with short ones, thus creating gaps that result in a longer makespan.

8.4.3 Running Time Scalability

Let us now turn our attention to the scalability of the SA-LF and WA-LF algorithms in terms of running time. Figure 8.10 plots the running time of the two algorithms, in seconds, as a function of the number of tasks in the multiprocessor scheduling problem. Two plots are shown in the figure. For the curve labeled "general scheduling," the processors assigned to a particular task were randomly selected from the set of all processors without any restrictions; consequently, these problem instances correspond to the general version of the $P|fix_j|C_{max}$ problem. For the curve labeled "chain scheduling," on the other hand, processors were labeled from 1 to m. For each task, an integer k and start processor p, $1 \leq p \leq m - k$ were selected randomly, and the set of k sequentially labeled processors, $p, \ldots, p + k - 1$, were assigned to execute the task. These problem instances correspond to the special case of the $P|fix_j|C_{max}$ problem derived from chain networks.

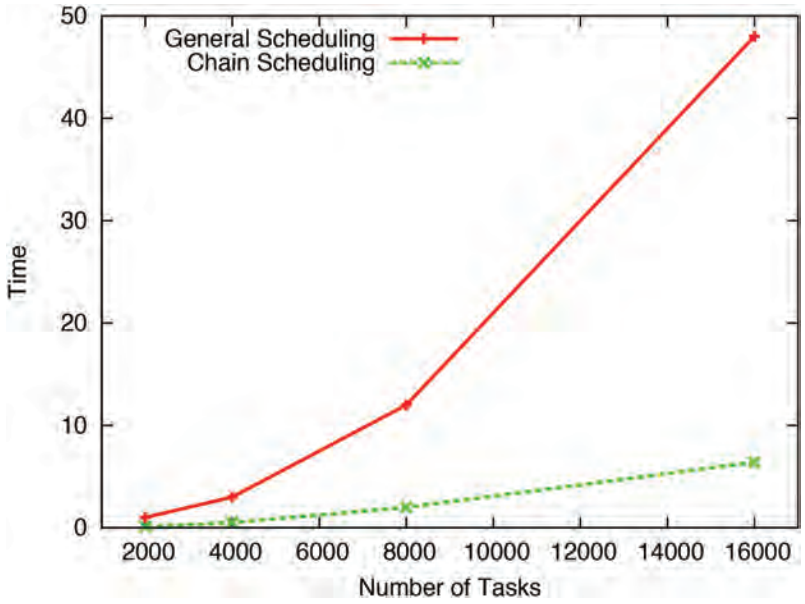

Figure 8.10 Running time, in seconds, of the scheduling algorithms as a function of the number of tasks.

As Figure 8.10 indicates, the scheduling algorithms scale to large instances of the two types of $P|fix_j|C_{max}$ problem. For the most general variant of the problem, instances of up to 16,000 tasks may be scheduled in less than one minute. Note that 16,000 tasks roughly correspond to the number of bidirectional traffic demands in a network with 125 nodes. For the special case of "chain scheduling," 16,000 tasks may be scheduled in less than seven seconds, due to the recursive nature of the corresponding algorithms. These results provide a strong indication that our algorithms may be used to tackle efficiently and effectively large instances of the spectrum assignment and corresponding task scheduling problems.

8.5 Concluding Remarks

We have developed scheduling algorithms that efficiently solve the spectrum assignment problem in mesh networks with good performance, under the assumption that the routing path of each traffic demand is fixed, i.e., it is part of the input to the problem and not subject to optimization. Our current research efforts are directed toward algorithms that jointly tackle the routing and spectrum assignment problems. More specifically, we are developing *parallel algorithms* for the RSA problem that applies the scheduling algorithms we presented in this chapter to a large number of reasonable routing configurations simultaneously.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant CNS-1113191 and by the High-Performance Computing Project at King Abdulaziz University.

References

[1] Winzer, P. J. (2010). Beyond 100G Ethernet. *IEEE Commun. Mag.*, 48(7), 26–30.

[2] Nag, A. and Tornatore, M. (2009). Optical network design with mixed line rates. *Opt. Switch. Netw.*, 6(3), 227–237.

[3] ITU-T G.694.1. Spectral grids for WDM applications: DWDM frequency grid, February 2002.

[4] Shen, G. and Zukerman, M. (2012). Spectrum-efficient and agile CO-OFDM optical transport networks: architecture, design, and operation. *IEEE Commun. Mag.*, 50(5), 82–89.

[5] Jinno, M., Ohara, T., Sone, Y., Hirano, A., Ishida, O., and Tomizawa, M. (2011). Elastic and adaptive optical networks: possible adoption scenarios and future standardization aspects. *IEEE Commun. Mag.*, 49(10), 164–172.

[6] Jinno, M., Takara, H., and Kozicki, B. (2009). 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.

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

[8] Jinno, M., Takara, H., Kozicki, B., Tsukishima, Y., Yoshimatsu, T., Kobayashi, T., Miyamoto, Y., Yonenaga, K., Takada, A., Ishida, O., and Matsuoka, S. (2008). Demonstration of novel spectrum-efficient elastic optical path network with per-channel variable capacity of 40 Gb/s to over 400 Gb/s. In *Proceedings of 34th European Conference on Optical Communication (ECOC)*, page Th.3.F.6, September.

[9] Zhang, G., De Leenheer, M., Morea, A., and Mukherjee, B. (2013). A survey on OFDM-based elastic core optical networking. *IEEE Commun. Surv. Tutor.* 15(1), 65–87, First Quarter.

[10] Shieh, W. (2011). OFDM for flexible high-speed optical networks. *J. Lightwave Technol.*, 29(10), 1560–1577.

[11] Lowery, A. J. and Armstrong, J. (2006). Orthogonal-frequency-division multiplexing for dispersion compensation of long-haul optical systems. *Opt. Express*, 14(6), 2079–2084.

[12] Lowery, A. J., Du, L. B., and Armstrong, J. (2007). Performance of optical OFDM in ultralong-haul WDM lightwave systems. *J. Lightwave Technol.*, 25(1), 131–138.

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

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

[15] Christodoulopoulos, K., Tomkos, I., and Varvarigos, E. A. (2011). Elastic bandwidth allocation in flexible OFDM-based optical networks. *J. Lightwave Technol.*, 29(9), 1354–1366.

[16] Zhang, Y., Zheng, X., Li, Q., Hua, N., Li, Y., and Zhang, H. (2011). Traffic grooming in spectrum-elastic optical path networks. In *Proceedings of*

Optical Fiber Communication Conference and the National Fiber Optic Engineers Conference (OFC/NFOEC), page OTuI1, March.

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

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

[19] Liu, Z. and Rouskas, G. N. (2012). A fast path-based ILP formulation for offline RWA in mesh optical networks. In *Proceedings of IEEE GLOBECOM 2012*, pages 2990–2995, December.

[20] Talebi, S., Alam, F., Katib, I., Khamis, M., Khalifah, R., and Rouskas, G. N. (2014). Spectrum management techniques for elastic optical networks: A survey. *Opt. Switch. Netw.*, 13, 34–48.

[21] Zhu, Y., Rouskas, G. N., and Perros, H. G. (2000). A path decomposition approach for computing blocking probabilities in wavelength routing networks. *IEEE/ACM Trans. Netw.*, 8(6), 747–762.

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

[23] Talebi, S., Bampis, E., Lucarelli, G., Katib, I., and Rouskas, G. N. (2014). Spectrum assignment in optical networks: A multiprocessor scheduling perspective. *J. Opt. Commun. Netw.*, 6(8), 754–763.

[24] Bampis, E. and Kononov, A. (2001). On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, San Francisco.

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

[26] Jinno, M., Kozicki, B., Takara, H., Watanabe, A., Sone, Y., Tanaka, T., and Hirano, A. (2010). Distance-adaptive spectrum resource allocation in spectrum-sliced elastic optical path network. *IEEE Commun. Mag.*, 48(8), 138–145.