Miami University

Department of Electrical and Computer Engineering

MIAMI UNIVERSITY

Spectrum Assignment in Mesh Elastic Optical Networks

Authors
Md Rakibul Ahasan
Farzana Alam

Advisor
Dr. Gokhan Sahin

1 Introduction

In the elastic optical network, spectrum assignment is a major design and control problem. The author finds that the spectrum assignment problem has a similarity with the multiprocessor scheduling problem on a dedicated processor. Based on this similarity authors built a scheduling algorithm that is proven effective for spectrum assignment in mesh and chain optical networks.

In the offline routing and spectrum assignment problem, the input has the forecasted traffic demand and the object is to assign spectrum in a physical path in such a way that a maximum number of spectrum slots is minimized. A couple of variant of this problem is mentioned and those are reached versus modulation level trade-off, traffic grooming, or restoration and most researchers use integer linear programming (iLp) formulation and heuristic algorithm to achieve the optimal solution. However, this heuristic algorithm is not transferable to other problem variants and it is also difficult to measure the performance of heuristic algorithms.

Hence the authors aims to discuss the similarity of spectrum assignment with multiprocessor scheduling problem using a dedicated processor and develop a scheduling algorithm. This scheduling algorithm is then performed over a real network topology. They also evaluated the performance of their algorithm by calculating the lower bound (LB) of each path first and showed how close or far is the solution by their scheduling algorithm.

2 Scheduling Algorithm At a Glance

To schedule multiprocessors, firstly authors have considered the following mesh network with some predefined terms for spectrum assignment (SA) problem. For inputs, they have started with defining the graph, G = (V, A) where V is a collection of nodes and A is a set of arcs (directed edges). To define the traffic demand, authors have chosen T = $[t_{sd}]$, which indicates the number of spectrum slots that is required to support traffic from a source node (s) to a destination node (d). In the context of each traffic demand, there is a predefined route labeled as $[r_{sd}]$, specifying the path from source to destination. The main objective of the SA problem is to allocate spectrum slots along the complete route of each traffic demand in a manner that minimizes the overall spectrum utilization on any arc within the network.

RSA Constraints: To address the SA problem effectively, the following constraints have been observed carefully by the authors:

1. Spectrum Contiguity: To guarantee a smooth and continuous succession, adjacent spectrum slots should be allotted to each demand.

2. Spectrum Continuity: Consistency must be maintained throughout, with demands that follow the same path using the same spectrum slots throughout each arc.

3. Non-Overlapping Spectrum: Different demands should be allotted separate, non-overlapping segments of the available spectrum when they share a common arc. By using this method, any overlap or interference between them is avoided.

To explore the multiprocessor scheduling problem represented by $P|fix_j|C_{max}$, it is needed to be defined as follows:

$P|fix_j|C_{max}$ **Inputs:**
There are m processors that are all the same, n tasks in total, and each one is identified by its processing time, P_j. There is a certain set fix_j of processor subsets that can perform a given task j.

$P|fix_j|C_{max}$ **Objective:** The goal is to create a schedule for these tasks in a way that minimizes the makespan, C_{max}, which is the longest time that any task in the schedule can take to complete.

$P|fix_j|C_{max}$ **Constraints**: To address this problem, following constraints have been considered:

1. No preemption is allowed: Once a task is assigned to a processor subset, it cannot be interrupted or divided.

2. Task j requires all processors in the selected subset to be active at the same time.

3. All processors are limited to executing just one task at a time; no multitasking is allowed.

In essence, the challenge is to efficiently schedule tasks across the available processors to minimize the time it takes for all tasks to be completed, with the given constraints in mind.

2.1 SCHEDULING ALGORITHMS FOR SPECTRUM ASSIGNMENT IN MESH NETWORKS

In this section, an effective scheduling algorithm have been discussed which is used to find out the $P|fix_j|C_{max}$ and SA problem in mesh network. A list of n tasks, each denoted by a task number j (which can range from 1 to n), sets of processors, fix_j, corresponding processing time, P_j, of the tasks, that are available for each task are inputted into the algorithm. The tasks on the list can be arranged in various orders, but for the purposes of this study, we'll concentrate on two particular ordering techniques:

Longest-First (LF): According to this method, tasks are listed in decreasing order of processing time, starting with the ones that require the longest processing times P_j.

Widest-First (WF): Tasks in this method are presented in decreasing order of the size of their processor sets, $|fix_j|$. As a result, tasks with the biggest processor sets are listed first.

This algorithm is called either SA-LF or SA-WF, depending on whether the tasks are ordered widest-first (SA-WF) or longest-first (SA-LF), respectively. The algorithm keeps track of the available (free) processors as well as a list of tasks that are currently in progress, initially was free. According to the chosen algorithm, the tasks in the input list L are first arranged based on either the longest-first or widest-first order. Subsequently, the algorithm iteratively employs two essential procedures, ScheduleTasks() and AdvanceTime(), until every task in the list L has been scheduled, resulting in an empty list.

The ScheduleTasks() method takes three inputs: the list of tasks currently being worked on (L_p), the list of tasks that haven't been scheduled yet (L), and the set of processors that are currently free (F) at the specific time (t). It aims to schedule each activity starting at time t from list L. If all of the processors from (fix_j) are available at time t, then task j can be scheduled. Until all tasks in list L have been completed or all processors are occupied, this process keeps going. The AdvanceTime() method is called when every single processor is occupied.

The first task that is running and has concluded its execution is identified by the AdvanceTime() method. When a task achieves this stage, the processors it was utilizing become available again and allows the algorithm to go on to the next step and this procedure is repeated till the scheduling is finished.

Algorithm 1 Algorithm for Scheduling Tasks on Multiprocessors

Require: 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

Ensure: A schedule of tasks, indicating when each task j commences execution on the multi-processor system.

1: Start by sorting the tasks in list L according to the widest or longest first criteria.
2: Track task status, initialize a list (Lp) with n elements set to "false".
3: $Time(t)t \leftarrow 0$
4: $Thefreeprocessors(Fp)F_p \leftarrow m$
5: $Counter \leftarrow 0$
6: $F[1, \ldots, m] \leftarrow true$
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

Algorithm 2 Procedure ScheduleTasks

1: **procedure** ScheduleTasks(L, L_p, F, t, j)
2: **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.
3: **while** $j \neq n$ and $F_p > 0$ **do**
4: **if** $L_p[j] = false$ and $F[fix_j] = true$ **then**
5: $S_j \leftarrow t$
6: $L_p[j] \leftarrow nexttrue$
7: $F[fix_j] \leftarrow false$
8: $F_p \leftarrow F_p - count(fix_j)$
9: ScheduleTasks$(L, L_p, F, t, j + 1)$
10: **break**
11: **end if**
12: $j \leftarrow j + 1$
13: **end while**
14: **end procedure**

The algorithm advances time to the next expected completion time (time t) of the current task. After that, every processor used by this task is released. To schedule any unfinished tasks, beginning at the new time (t), the ScheduleTasks() process is therefore called again. This process keeps going until every task on the list has been scheduled. It is to be noted that, in the worst-case situation, both the ScheduleTasks() and AdvanceTime() can be called a maximum of n times, where n is the total number of tasks.

Algorithm 3 Procedure AdvanceTime

1: **procedure** AdvanceTime(L_p, F, t, $Counter$)
2: **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.
3: $j \leftarrow 0$
4: $j_{min} \leftarrow -1$
5: $t_{min} \leftarrow \infty$
6: **while** $j \neq n$ **do**
7: **if** $L_p[j] = true$ and $S_j + p_j > t$ and $S_j + p_j < t_{min}$ **then**
8: $j_{min} \leftarrow j$
9: $t_{min} \leftarrow S_j + p_j$
10: **end if**
11: $j \leftarrow j + 1$
12: **end while**
13: $F[fix_{j_{min}}] \leftarrow true$
14: $F_p \leftarrow F_p + count(fix_{j_{min}})$
15: $t \leftarrow t_{min}$
16: **end procedure**

3 Numerical Result

The objective of this report is to create an optical network spectrum assignment problem and solve it using a multiprocessor scheduling solution. To do that we have first created an optical network for 32 node 108 link topology. This is created using the networkx python library. We have carefully created the links between each node and completed the topology. The figure 1 has the network visualization of 32 node and 108 link topology.

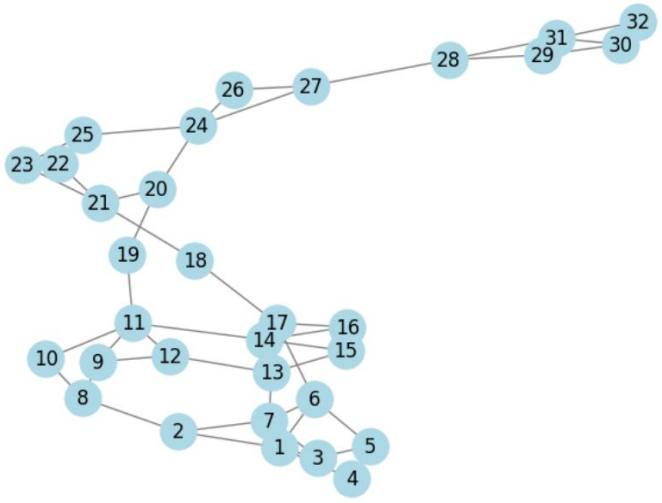

Figure 1: 32 node 108 link topology

Once the network is created we have selected two random nodes for our numerical analysis. We have taken the node pair [10, 26]. Now using a random shortest path calculation we have determined the shortest path between the node pair [10, 26] and we figured out the links of this shortest path. Details of this is in Table 1.

Table 1: Shortest path and link details

Link Nodes	**Link Acronym**
10 to 11	L1
11 to 19	L2
19 to 20	L3
20 to 24	L4
24 to 26	L5

The next step is to generate traffic demand between this node pair of each link. However, it is needed to mention that the entire path traffic rate is the same and it is taken as a uniform traffic distribution. This distribution has traffic demand in any of five discrete values $\{10, 40, 100, 400, 1000\}$. Let's consider the spectrum slot width is 1.25 GHz with 16-QAM modulation each

traffic rate is equivalent to $\{1, 1, 2, 8, 20\}$ spectrum slots. For our convenience, we have considered the 1000 Gbps traffic rate of our node pair [10, 26] and it will support up to 20 spectrum slots.

Then we have converted the SA problem instance into a multiprocessor scheduling $P|fix_j|C_{max}$ instances. To do that we have tried multiple combinations of spectrum slot demands. We have calculated the lower bound of the spectrum slot of the links and that is 11 spectrum slots. The spectrum slot requirement is mentioned in Table 2.

Table 2: Spectrum Slot Requirement

No of Spectrum Slot	**Link Requirement**
3	L1,L2,L3,L4,L5
1	L1,L2,L3,L4,L5
2	L1,L2,L3
2	L2,L3,L4
4	L3,L4,L5
3	L1,L2

The above spectrum slot requirement is not possible to directly inject into the algorithm that we explained in the earlier section. However, the conversion is pretty simple. We can consider the number of spectrum slots as processing time P_j and the number of link requirements as the processor. And each processing time P_j and corresponding processor has clubbed to task number as $T \in T1, T2, ...T6$, because we have a total 6 traffic demand between the node pair [10, 26]. This detail conversion is in Table 3.

Table 3: Multiprocessor Scheduling Algorithm Input Details

Task	**Processing Time P_j**	**Processor**
T1	3	1,2,3,4,5
T2	2	1,2,3
T3	1	1,2,3,4,5
T4	2	2,3,4
T5	4	3,4,5
T6	3	1,2

The multiprocessor scheduling algorithm has the option to schedule either on the longest first(LF) criteria or the widest first(WF) criteria. The longest first criterion means the algorithm will schedule the task that has the highest processing time and continue to schedule the tasks until the list queue is empty. The widest first criterion means the algorithm will schedule the task that requires the highest number of processors and continue to schedule the task until the list queue is empty.

Longest First					
14					
13					
12					
11	T3	T3	T3	T3	T3
10		T4	T4	T4	
9		T4	T4	T4	
8	T2	T2	T2		
7	T2	T2	T2		
6	T1	T1	T1	T1	T1
5	T1	T1	T1	T1	T1
4	T1	T1	T1	T1	T1
3			T6	T6	T6
2	T5	T5	T6	T6	T6
1	T5	T5	T6	T6	T6
0	T5	T5	T6	T6	T6
	L1	L2	L3	L4	L5

Figure 2: SA Scheduling According to Longest First Criteria

Figure 2 has the abstract output of the LF scheduling algorithm and is presented on spectrum slot allocation in links $\{L1, L2, L3, L4, L5\}$. Similarly, Figure 3 has the abstract output of the WF scheduling algorithm.

Widest First					
14					
13					
12					
11		T4	T4	T4	
10		T4	T4	T4	
9	T2	T2	T2		
8	T2	T2	T2		
7			T6	T6	T6
6	T5	T5	T6	T6	T6
5	T5	T5	T6	T6	T6
4	T5	T5	T6	T6	T6
3	T3	T3	T3	T3	T3
2	T1	T1	T1	T1	T1
1	T1	T1	T1	T1	T1
0	T1	T1	T1	T1	T1
	L1	L2	L3	L4	L5

Figure 3: SA Scheduling According to Widest First Criteria

Both algorithms follow the same multiprocessor scheduling algorithm except for the sorting criteria but both algorithms effectively allocate the spectrum slots concerning the links. For example in Figure 3 the algorithm first process the T1 and T3 task because those has the highest number of 5 processors. After the according to preceding the algorithm can start T4, T2, or T6 task because all of those require 3 processors. But the algorithm took T6 because it is attached to processor 3,4,5 and with T6 we can also schedule T5 that need processor 1,2. Hence tasks T6 and T5 are scheduled at the same time. After that T2 and T4 task is scheduled. A similar approach is applied to the LF algorithm. Now for both of these cases, the proposed heuristic algorithm can allocate frequency slot 11 and that is exactly equal to the LB of this particular case. So we can say in our simulation scenario the heuristic algorithm is optimal meaning the max spectrum slot for each link is minimized.