Miami University

Department of Electrical and Computer Engineering

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 Algorithmic approaches for allocating spectrum in mesh Network topologies

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

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 Scheduling Algorithm (SA-LF/WF) for $P|fix_j|C_{max}$

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, i.e., the time S_j when each task j starts execution on the multi-processor system

```
 1: Sort the tasks in list $L$ based on longest-first or widest-first criteria
 2: $L_p[1, \ldots, n] \leftarrow false$
 3: $t \leftarrow 0$
 4: $F_p \leftarrow m$
 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 3 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:              $L_p[j] \leftarrow true$
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
```

Algorithm 4 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
```

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.

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.

3 Longest first and widest first analysis

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.

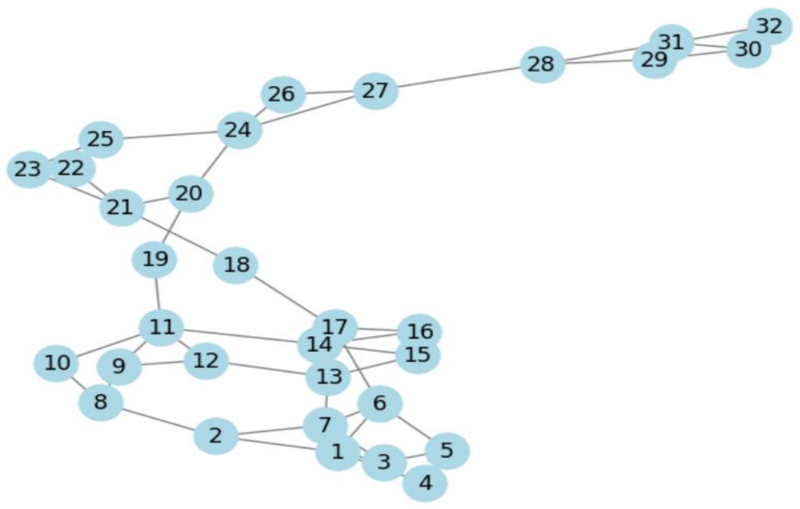


Figure 1: 32 node 108 link topology

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

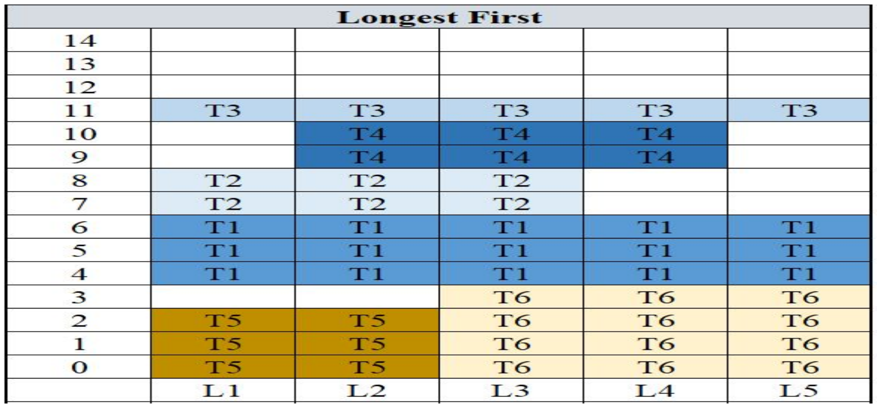
No of Spectrum Slot	**Link Requirement**
3	L1,L2,L3,L4,L5
1	L1,L2,L3,L4,L5
2	L1,L2,L3
2	L2,L3,L4
4	L3,L4,L5
3	L1,L2

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.

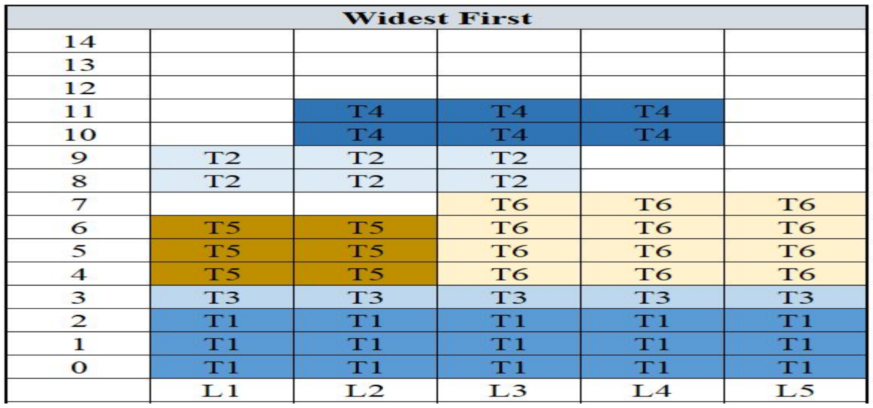


Longest First					
14					
13					
12					
11	T3	T3	T3	T3	T3
10		T4	T4	T4	
9		T4	T4	T4	
8	T2	T2	T2		
7	T2	T2	T2		
6	T1	T1	T1	T1	T1
5	T1	T1	T1	T1	T1
4	T1	T1	T1	T1	T1
3			T6	T6	T6
2	T5	T5	T6	T6	T6
1	T5	T5	T6	T6	T6
0	T5	T5	T6	T6	T6
	L1	L2	L3	L4	L5

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.

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.



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

4 Brief description of genetic algorithm

In a genetic algorithm, the search space is represented by a sequence of genes and new search nodes are generated by genetic operators. Then these nodes are evaluated using a fitness function, and stochastic components are incorporated to regulate the genetic operations. The stages of a typical genetic algorithm are as follows [1]:

1. Genetic representation: This is needed for optimizing problems.
2. Population: On a population of encoded solutions, the GA functions.
3. Fitness Function: Each solution in the population is assessed for optimally using a fitness function.
4. Original Operators: These operators are in charge of creating a new population from the preexisting one.
5. Control Parameters: A range of parameters that influences on how the genetic algorithm behaves.

One way to conceptualize the GA is, as a natural process in which a population of solutions goes through several generations of evolution. Every generation evaluates eachsolution for fitness, and solutions that are considered fit for reproduction are selected accordingly. This process of selection is consistent with the Survival of the fittest, in which the best solutions are kept for future generations and the worst ones are eliminated [1]. A solution's fitness value determines how good it is.

Four separate but related tasks can be divided into the development of a genetic algorithm for any given problem [2]:

1. selecting the string representation;
2. creating the genetic operators;
3. defining the fitness function;

4. determining the probabilities that govern the genetic operators.

Each of these four elements has a substantial effect on the genetic algorithm's overall performance as well as the resultant solution.

4.1 String representation

The present section describes a method for creating a first population of search nodes as well as the string representation used to solve the multiprocessor scheduling problem. All search nodes in a particular search space must be represented uniquely. In this scenario, a one-to-one correlation between the strings and the search nodes is preferable. For a valid search node (schedule) for the multiprocessor scheduling problem, some criteria needs to be achieved like, (a) the tasks' dependencies of precedence are satisfied, and (b) each task needs to be present and appear exactly once in the schedule to guarantee completeness and uniqueness [2]. The necessity of considering precedence relationships between tasks scheduled on various processors into account is eliminated by this model. Maintaining precedence relations within each processor is still required though.

4.2 Initial population

Genetic algorithms have the advantage of being able to search across many nodes in the search space at once. This requires the initial population of search nodes to be generated at random. In a multiprocessor system with p processors, the Generate-Schedule algorithm is intended to generate a schedule for the task graph (TG) at random. The following is how the algorithm works [2]:

1. Determine the height of each task.
2. Divide work according to height.
3. For each of the first p - 1 processors, perform the following steps
4. Create the processor's schedule.
5. Assign the remaining tasks

We can generate the initial population of search nodes needed by the genetic algorithm by repeatedly using the Generate-Schedule technique.

4.3 Mutation

Small, random changes to individual solutions within the population are called mutations in genetic algorithms. Every gene within a solution signifies a distinct component of the solution, and mutation introduces diversity through the modification of these genes with a probability p. In genetic algorithms, the main function of mutation is to introduce new genetic material into the population, which allows it to search new areas of the search space [3]. The genetic algorithm's adaptability is enhanced by mutation. By adding new genetic material, it enables the algorithm to adapt to changes in the surrounding environment or the problem surface. Mutation acts as an exploration mechanism, preventing the genetic algorithm from getting stuck in suboptimal solutions. It ensures that the algorithm continues to search the solution space broadly, increasing the likelihood of finding high-quality solutions.

1. Keeping diversity:

   Genetic variety within a population is preserved through mutation. The genetic algorithm may converge too quickly to a local optimum in the absence of mutation, which would restrict the exploration of the whole solution space.

2. Exploration:

   Mutation enables the algorithm to explore areas of the search space that may not have been initially represented in the population by introducing random modifications. In order to prevent premature convergence to inferior solutions, this is essential.

3. Escape local optima:

   By bringing forward new solutions, mutation offers a way to get out of local optimal conditions. This is particularly crucial in intricate, multimodal search environments.

4. Adaptation:

   The genetic algorithm's adaptability is enhanced by mutation. It enables the program to react to environmental modification.

4.4 Crossover

The crossover mechanism is the reason genetic algorithms work so well. Crossover allows genetic information to be exchanged between solutions in a systematic yet random way, opening the possibility that good solutions could lead to even better ones. In order to perform a crossover, a random spot on the population is chosen where the parental segments are exchanged. A new offspring is produced by this procedure [3]. The probability of two chromosomes switching segments is indicated by the crossover rate, which is the frequency of crossover events for chromosomes in a single generation. If the rate of crossover is 100 %, then all offspring are the result of crossover; if the rate is 0%, then the new generation is completely derived from the older population.

4.5 Termination Criteria

Conditions that indicate the algorithm's end are known as termination criteria for genetic algorithms. Among the many, terms of termination are: (a) Reaching a fixed number of generation, (b) Attaining a fitness level, and (c) No improvement observed in the best fitness value over time.

5 Comparative result analysis of LF, WF and GA

In this section, we will discuss the comparative analysis of spectrum scheduling in a mesh optical network. There are three different algorithms: longest first, widest first, and genetic algorithm. We will push 40 connections and 100 connection requests in the network and we will discuss briefly how each of these algorithms have been performed. The visualization of the network is found in Fig 4.

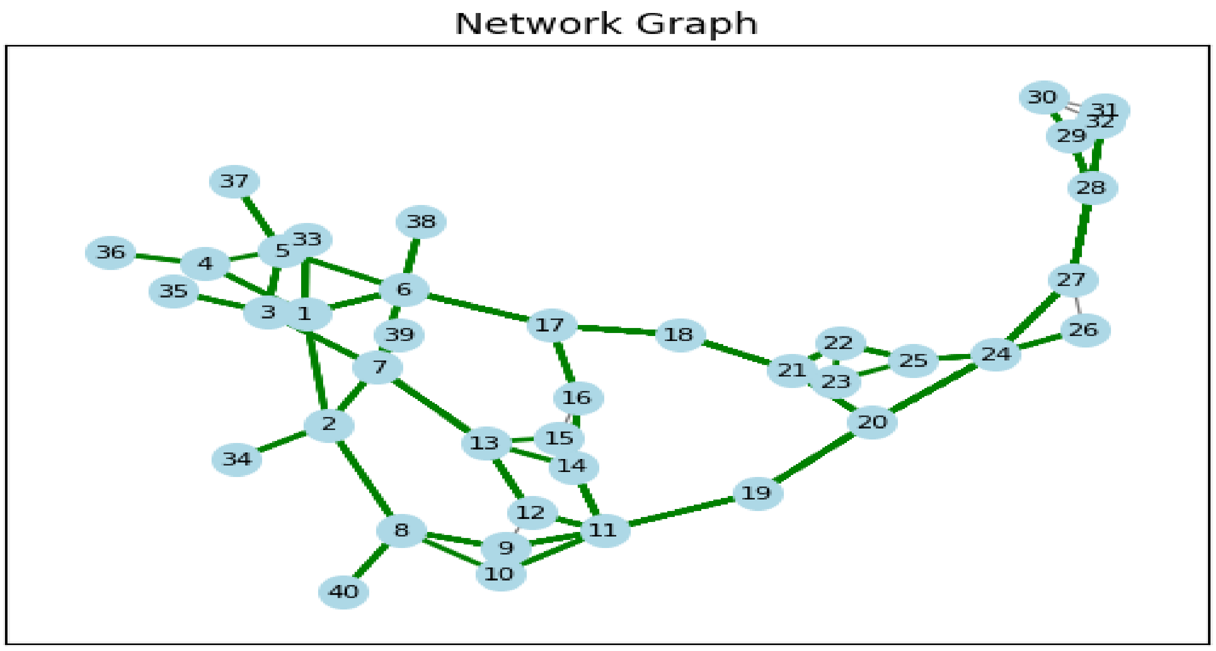


Figure 4: Network topology (Node count:40)

The network consists of 40 nodes and from the network we have created our 40 and 100 connection request datasets. The programming interface of the longest first and widest first algorithm is Matlab and the genetic algorithm is Python. Hence we have created 40 and 100 connection requests for two interfaces. The python code file *networkdata_creation.ipynb* has the code details of this 40 and 100 connection request dataset creation based on a 40 node network data in Fig 4. The key steps we have considered for this dataset are:

1. Created a random network (Node count:40).
2. Randomly selected source and destination pair.
3. Shortest path identification of all source and destination pairs.
4. Provided a unique ID to link in each shortest path and tagged with global mapping.
5. Randomly assigned the processor timing. [Max processing time: 5]
6. Get task list for Genetic Algorithm and LF/WF criteria

Table 4: 40 connection request dataset

Name	P	Fixj	Start
T1	5	[67, 7, 41, 65]	-1
T2	3	[50, 51, 23, 13, 54]	-1
T3	4	[55, 37, 18, 10]	-1
T4	2	[59, 61, 2, 16]	-1
T5	3	[47, 68]	-1
T6	5	[53, 18, 10, 12, 44]	-1
T7	5	[27, 23, 64]	-1
T8	5	[5, 55, 14, 23, 13, 54, 59, 61, 2, 16]	-1
T9	4	[57, 44, 45, 61, 2, 16, 70]	-1
T10	3	[45, 61]	-1
T11	4	[36, 14, 23, 13, 54, 56]	-1
T12	1	[48, 26, 19, 42, 21, 58, 28, 3, 9, 43, 24]	-1
T13	3	[69, 6, 14, 23, 13, 54, 56]	-1
T14	4	[38, 44, 45, 61]	-1
T15	2	[28, 3, 9, 43, 15]	-1
T16	2	[36, 14, 23, 13, 54, 59, 61, 2, 16, 46]	-1
T17	2	[48, 26, 19, 42, 21, 58, 28, 3, 9, 43, 66, 17]	-1
T18	5	[54, 59, 29]	-1
T19	4	[25, 30, 6, 37]	-1
T20	3	[31, 28, 3, 9, 43, 15]	-1
T21	2	[53, 39, 14, 23, 13]	-1
T22	3	[3, 9, 43, 37, 18, 73]	-1
T23	5	[35, 67, 7, 41, 47, 60]	-1
T24	1	[43]	-1
T25	4	[38, 44, 45, 58, 56]	-1
T26	3	[28, 3, 9, 43, 37]	-1
T27	1	[71, 18, 10]	-1
T28	5	[20, 62, 28, 3, 9, 43, 24]	-1
T29	2	[4, 32, 40, 22, 72]	-1
T30	2	[6, 37]	-1
T31	4	[51, 23, 13, 54, 59, 61, 2, 16, 46, 63]	-1
T32	5	[9, 43, 66]	-1
T33	4	[71, 39, 24]	-1
T34	3	[25, 27, 23, 64]	-1
T35	3	[33, 19]	-1
T36	3	[42, 49, 1]	-1
T37	3	[25, 27, 23, 13, 54, 8, 34]	-1
T38	1	[48, 26, 19, 42, 49]	-1
T39	3	[11, 32, 39, 66, 17]	-1
T40	3	[52, 23]	-1

As mentioned in the earlier sections we will solve the spectrum assignment problem considering it is a multiprocessor scheduling problem. Hence the processing time of each processor is the number of spectrum slots and the processor count or number of processors is associated with the number of links in the shortest path for each connection request. In table 4, the details of 40 connection requests details are provided, from there we will explain two instances of the task. One of these two instances is selected based on the longest processing time and a moderate processing time and the task number is T6 the other one is the widest task with the lowest processing time and the task number is T12. We will now discuss the LF, WF, and GA algorithm implementation 40 and 100 connection request. And the task is specific to T6 and T12. In the following picture the x-axis is representing the link in shortest part as processor number, and y-axis represents the processing time or the spectrum assignment. Both the T6 and T12 tasks are marked in the picture for better understanding.

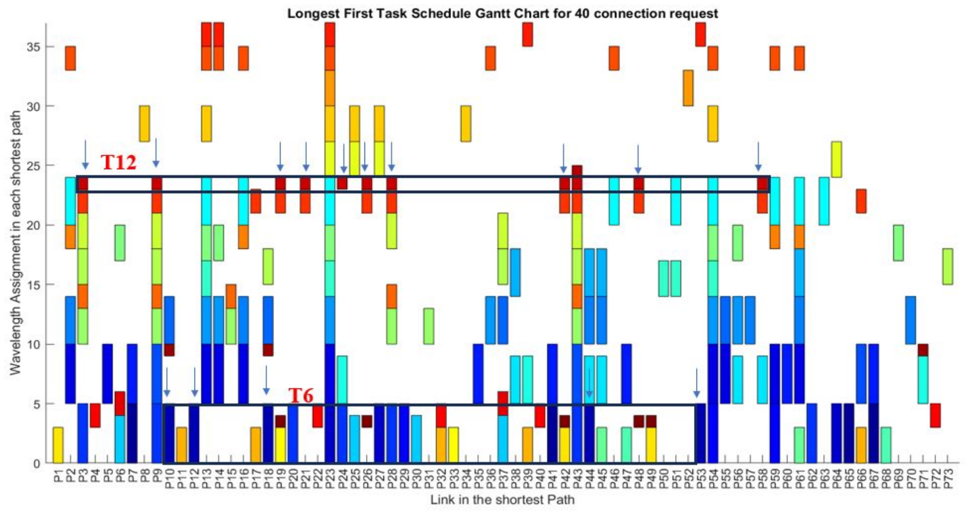


Figure 5: LF algorithm implication for 40 connection requests

In Figure 5 we are observing the implication of the LF algorithm for 40 connection requests for random processing time. The term longest first means the longest processing time will be processed first. And we are observing that task T6 is scheduled first because it has the highest processing time 5. On the other hand, task T12 is started at time 23 though it has the lowest processing time of 1. This task was supposed to start at the end of all tasks but we observed it started earlier because the processor associated with the T12 got free and the task got scheduled. So, the longest first algorithm is working as expected for 40 connection requests. We have observed a similar scenario for 100 connection requests.

Figure 6 has the widest first implementation visualization. The term widest first means the task that has the highest processor count will be processed first. The difference between the longest first and widest first is, the scheduling of task T6 remains the same but task T12 is scheduled earlier and it is started at 10. The reason is in the widest first algorithm T12 has precedence with another task because the number of processors is 11. The widest first algorithm operates as anticipated for 40 connection requests, and a comparable performance has been noted when handling 100 connection requests.

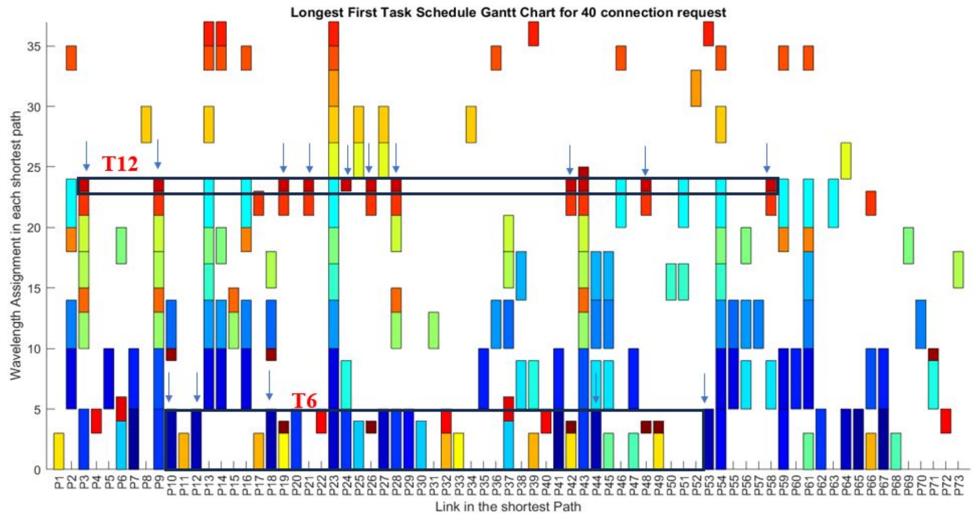


Figure 6: WF algorithm implication for 40 connection requests

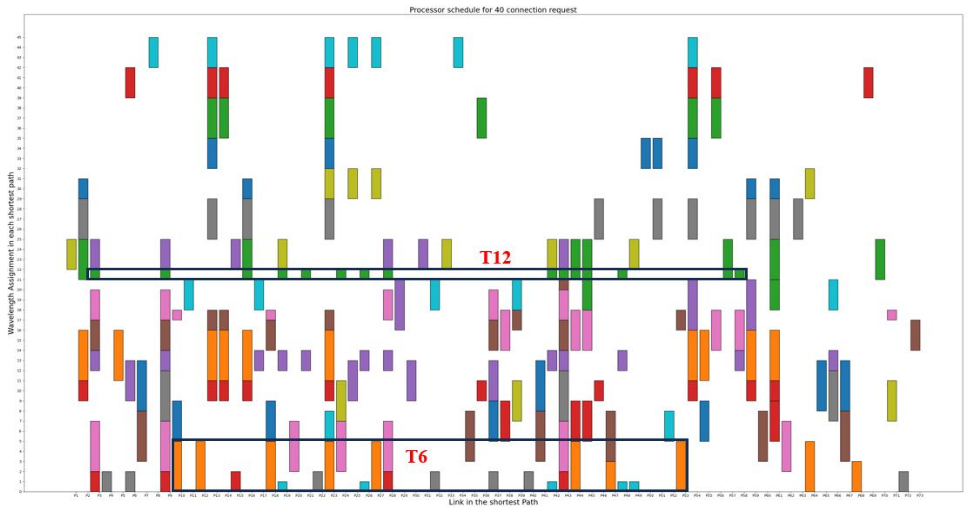


Figure 7: GA algorithm implication for 40 connection requests

Figure 7 contains the genetic algorithm implementation for 40 connection requests. We have considered some factors of the genetic algorithm so that it can cope up with the multiprocessor scheduling algorithm. First, we are expecting the fitness function will return the maximum of the start time and Processing Time of the task. As a selection factor, the genetic algorithm will automatically select the best schedules or the population that has the best fitness. The factor crossover means we first create parent1 and parent2, choose a crossover point, and observe the performance of the schedule. A sample representation can be P1 and P2 as two parents containing the complete schedule and we randomly select a crossover point for example T20. Based on this crossover point a new scheme is created and the genetic algorithm will evaluate its performance. The last factor is mutation that means for a schedule it randomly swaps some task and observe whether the new schedule performance is better or not. The parameter value for genetic algorithm is:

Table 5: Genetic Algorithm Parameters

Parameter	Value
Population Size	10
Crossover Rate	0.7
Mutation Rate	0.2
Maximum Repeats	10000

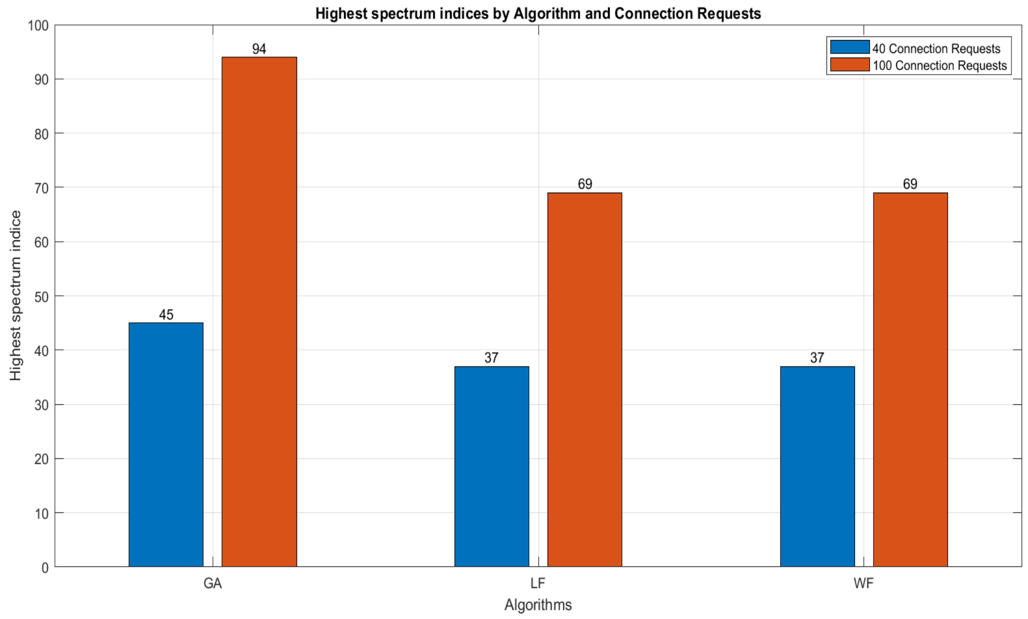


Figure 8: Comparison result of different algorithms for different connection request

The goal of spectrum assignment allocation using different algorithms, the maximum spectrum indices of the link should be minimal. Figure 8 has a comparison analysis of different algorithms during spectrum allocation. This analysis is performed for both 40 and

100 connection requests. Both the longest first and widest first perform similarly for 40 and 100 connection requests. The highest spectrum for both cases is 37 and 69 respectively. One of the key reason is that, both LF and WF has presorting criteria for the tasks. The LF algorithm first sorts the last according to the processing time and then schedules the task, and the WF algorithm first sorts the task according to the number of processors and then starts scheduling. However, the genetic algorithm-based spectrum allocation is not performed well compared to the LF and WF. We have used different parameter values and gone up to 10,000 iterations but the outcome is not improved. The highest spectrum indices of GA for 40 and 100 connection request is 45 and 94 respectively. To find the root cause of the problem, we did some further study on genetic algorithms and concluded. The reason for this big difference between GA with LF and WF is that in GA there are no presorting criteria, meaning the GA is working on the task concerning two constraints. The GA has to look for the lowest or highest processing time and the lowest or highest number of processors at each time of task scheduling. Due to this GA has less flexibility when scheduling the task and eventually the maximum indices of spectrum are not as minimum as compared to LF and WF.

6 Conclusion

This project compared spectrum allocation methods in mesh elastic optical networks using Longest First (LF), Widest First (WF), and Genetic Algorithm (GA). We assessed these on 40 and 100 connection requests, aiming to minimize the maximum spectrum index.

LF and WF had similar performances, efficiently allocating the spectrum with indices of 37 for 40 requests and 69 for 100 requests. However, the GA, while robust, had higher spectrum indices of 45 and 94 for 40 and 100 requests, respectively, due to its non-pre-sorted, multi-constraint nature.

The study shows that while GAs are versatile, they face challenges in optimizing spectrum allocation. Improving GAs could involve pre-sorting or specialized fitness functions. LF and WF are effective for deterministic tasks, but GAs hold potential for future advancements in managing spectrum resources in increasingly complex optical networks.

A Appendix: How to run the project

Network Dataset Creation

1. Open Google Colab at https://colab.research.google.com/.
2. Go to **File** → **Upload Notebook** → browse and load the file `"networkdata_creation.ipynb"`.
3. Change the value in `random_pairs = generate_random_pairs(G,100)` to 40 or 100 for the connection request.
4. Select **Runtime** → **Run All**.
5. Output: 40 and 100 connection request dataset for Python and MATLAB interface.

These text files contain the 40 and 100 connection datasets used in this project:

- `40connectionrequest.txt`
- `100connectionrequest.txt`

Generating Connection Request Results

40 Connection Requests code files:

- Python: `code_connection_request_40/genetic_algorithm_40connection.ipynb`
- MATLAB:
  - `code_connection_request_40/scheduleTasksDemoLF_40_connection_request.m`
  - `code_connection_request_40/scheduleTasksDemoWF_40_connection_request.m`

100 Connection Requests code files:

- Python: `code_connection_request_100/genetic_algorithm_100connection.ipynb`
- MATLAB:
  - `code_connection_request_100/scheduleTasksDemoLF_100_connection_request.m`
  - `code_connection_request_100/scheduleTasksDemoWF_100_connection_request.m`

Instructions to run the code

1. Open Google Colab at https://colab.research.google.com/.
2. Go to **File** → **Upload Notebook** → browse and load the file `"genetic_algorithm_40connection.i` or `"genetic_algorithm_100connection.ipynb"`.
3. Select **Runtime** → **Run All**.

Comparison Results of LF, WF, and GA

1. Run the `summary.m` MATLAB file.
2. The output will be produced accordingly.

Output Results

All output results are in the `alloutputs` folder.

References

[1] M. Srinivas and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” in IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, no. 4, pp. 656-667, April 1994, doi: 10.1109/21.286385.

[2] E. S. H. Hou, N. Ansari and Hong Ren, “A genetic algorithm for multiprocessor scheduling,” in IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp. 113-120, Feb. 1994, doi: 10.1109/71.265940.

[3] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri and V.B.S. Prasath, “Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach”, vol. 10, pp. 390, 2019 doi: 10.3390/10.120390