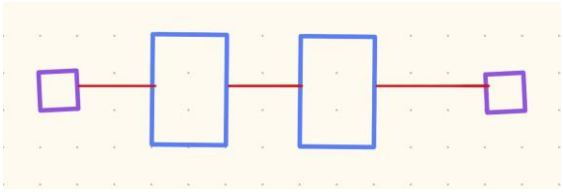
The #P-Complete problem in RSA (Routing and Spectrum Allocation)

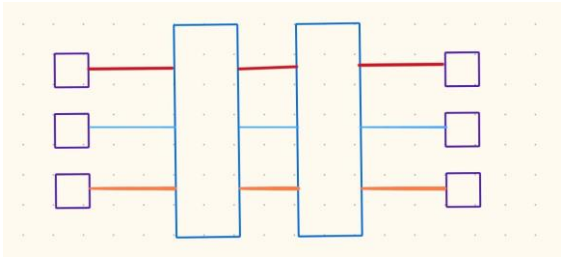
This is a classical problem in network routing algorithms. In our case the problem is same. To demonstrate the problem let us analyse some topologies. We will start with a simple topology:



In this topology, consider:

- 2 Transponders(violet), 2 ROADMs(blue)
- Only one path from source (TP1) to destination (TP2)
- One port from each device is used for the connection
- Computed path: TP1 –> RDM1 -> RDM2 -> TP2
- Perform RSA, setup intent

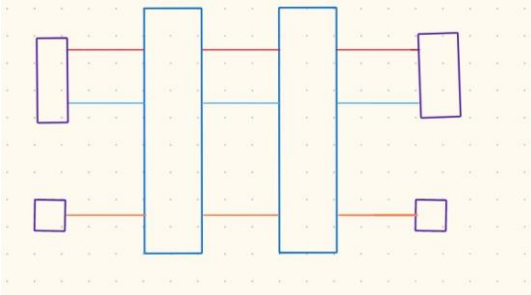
So, far there are no complications. But the problem gets worse when we have P parallel links or P, P – N, P parallel links from source to destination. Let us analyse the second topology:



Consider:

- We have 6 TPs and 2 ROADMs
- Each TPs are connected to another TPs (TP1 -> TP3), (TP2 -> TP4), (TP3 -> TP6)
- The common devices among these connections are RDM1 and RDM2
- Like our previous topology we can consider each of the path as an individual path
- So, the topology is still simple

Let us analyse the third topology:



Consider:

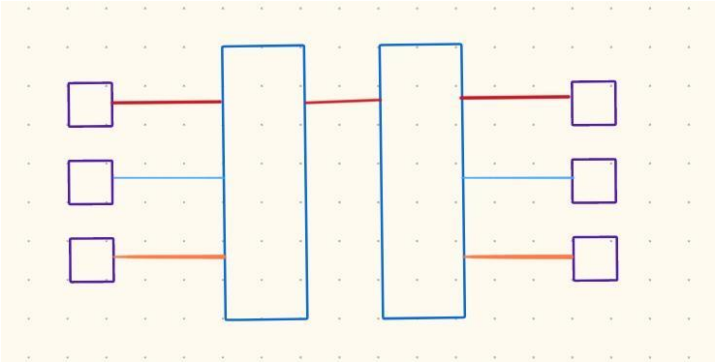
- 4 Transponders are connected, 2:2
- Common devices are RDM1 and RDM2
- TP1 and TP3 have two connections using two ports, we have two different Fibers
- TP2 and TP4 have one connection only
- Let us say TP1 and TP3 is already in operation and using 12.5GHz 10 slots in two Fibers (5 slots each)
- We want to setup another intent from TP2 and TP4. Even though the Fibers are different, can we re-use frequencies?

  This depends on the optical engine. If the transponder device has separate optical engines for each transceiver, then yes frequencies are re-usable if the ROADM supports (we are coming to this). But if the transceivers are sharing the same optical engine, then no! We cannot use the same frequency in different ports even if the Fiber is different.
- In the current intent (TP1->TP3) let us say we have different optical engines in each transceiver and we want to transmit 193.1THz from both ports of TP1. The common carrier in between is RDM1 and RDM2. Consider the first hop, TP1 -> RDM1:

  - RDM1 receives two frequencies (193.1) in two different ports but they are same.

  - SRG receives the frequencies and sends to WSS - signals get destroyed in SRG (I am confused here, will need your comments. Signal gets destroyed in SRG or in WSS!)
- So, we cannot propagate same frequencies through the SRG ports (if we combine multiple degrees together this might be possible but for now let us consider the simple analogy)
- Now, we want to setup another intent from TP2 -> TP4. The whole C band (there are other bands but let us consider C band for simplicity) is free in TP2 port 1101 (for example), but we cannot use just any slot. Why? We already discussed it before, because the same SRG is receiving the signals.

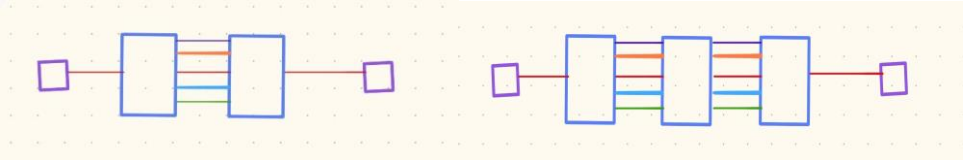
From this point the problem starts scaling, bad. Before coming to solutions, we want to discuss some more topologies:



We have:

- 6 TPs connected to each other with just one connection in between two ROADMs
- If you want a connection from TP3 -> TP6, can you use any slot even though you have all the slots free in TP3 and TP6? No! Because if the frequency is already used by other TPs, the connection between RDM1->RDM2 will refuse it.
- So, when you compute the RSA, you must consider the reserved frequencies in other transponder ports. From here the computational complexity starts scaling bad.

The final topology we will discuss is this type:



If I want to:

- Setup an intent from TP1 -> TP2 (consider other transponders are also connected to the same ROADM, so we must consider RSA too)
- How many paths we have from TP1 – TP2?
- In total 5 paths. Even though we must choose one path, we must calculate all paths for routing and spectrum allocation (RSA).
- Let us add another hop with uniform links and see how the computation changes. Now, we have $5 * 5 = 5^2$ possibilities. But, with another uniform hop? It gets worse $5 * 5 * 5 = 5^3 = 125$.

Now to represent this situation we can form some equations: (want comments from prof.: if these mathematical equations are valid for this scenario)

- It is not always true that we will have same number of links between hops
- It will be heterogeneous, which means uniform and non-uniform
- Let us consider hops as “H” and Paths as “λ ”

- If the situation is uniform (current topology that we are discussing): $\Phi uniform\,(H) = \lambda^H$ paths
  $\Phi uniform\,(2) = 5^2 = 25$
- But if the situation is non-uniform (4th topology we discussed before):
  - $\varphi_{non-uniform = \prod_{i=1}^{H} = \lambda_i}$; $3 \cdot 1 \cdot 3 = 9\,paths$
  - we perform each hop and count parallel links

This is one part of the mathematical complexity, and the other part is to perform the RSA on these paths. Let us discuss about my professor's approach, which is as per standards and an efficient one. So,

- I have 25 paths from source to destination (consider the uniform situation)
- I want to setup my intent in one of these paths (most suitable one)
- First, I will check my request: what is the channel bandwidth? Let us say 75GHz (for this we have ITU standards and mathematics for the computation).
- I want 6 slots of 12.5GHz (consider Flex-Grid)
- I start with the source device port (TP1, port 1101)
- I compute the available slots (convert the C_SLOT/L_SLOT values into bitmaps [1111…11..000..11..11]
- Perform an AND operation to determine usable slots [for example this is what we found: 00000000111111]
- Using LSB we can say first 6 slots (using first-fit policy)
- Now we will compute each path to check how many of them succeeds
- And finally, we choose one path and reserve those slots for our intent

Everything is nice and simple, so what is the problem? Well, the problem is we are now performing 25 computations for one RSA. Imagine a path with 100 parallel links and 5 devices in between (4 hops), what is the computational complexity (consider the worst-case scenario)?

$$\varphi_{uniform = \lambda^H = 100^4 = 100'000'000 = 100M}\text{Paths.}$$

This is not just bad; this is the worst. Why? Here I want to discuss some of the concepts I learned from HPC and SPM courses. Let us decide the machine which will compute this:

- A **4 core CPU** where each core is 3GHz, capable of 16 floating point operations
- Computational bandwidth = 3*4*16= 192GFlops = 1.9GHz
- Memory Bandwidth = 3200 MHz * 8Bytes (64 bits) * 2(dual channel) = 51.2GBps (true only for DRAM >= DDR4)

The scenario is extremely fast, so where is the problem? Let us calculate how much data we can compute in one flop:

- 16 FP operation in one Flop. So, assume 16Bytes in each flop.
- Consider we have 1GB data in our memory to process
- CPU Time: 0.0625/192 = 0.00032 = 0.326ms **(but remember we are considering all 4 cores are working together in parallel)**
- Memory Time: 1/51.2 = 0.0195 = 19.5ms (this is the highest time we pay)

Still now we do not see any problem with our machine except the 4 cores working all together part. But this is not the main problem, the main problem starts with communication costs. This is what we pay for each RSA check:

- We have 100M path saved in the memory
- 100M path contains 100M * N ports, we must perform the check for all of them
- Each port contains a bitmap [slots free or used]
- Database query latency for each request we make to the database (we can reduce this by using pre-computation) to fetch information about the ports
- We map that information in the memory for further processing
- Then we perform a loop which iterates 100M * N = N100M where N > 0
- If we pay 1ms for a fetch from the database, 1ms to store it in memory, 1ms to send it to the CPU + process it + write back (because the CPU is the fastest part); in total we are paying 3ms for one RSA check.
- How much shall we pay for N * 100M?

Now imagine we have a code which is sequential (we use one thread, but we have 4 available):

- This one thread is performing all these queries, computations, compare and swap, result write-back for N*100M items. The time you need for this is: 3*N*100M. It will run for minutes, even hours!
- In optical communication where everything happens in a blink of an eye, this much time is spent is not a good sign.

<u>From here we can think of some solutions: (proposed solutions)</u>

Before starting the main process, we can propose a pre-computation step. By following these steps:

- Make one query from source to destination, to find all the links available
- Create a data-structure (list in python)
- Create a MultiDiGraph type graph using **networkX** package (it is hard because there are some limitations of the package as well, but we can do this)
- Store the graph as our base (links, ports, integer value for the bitmap conversion)

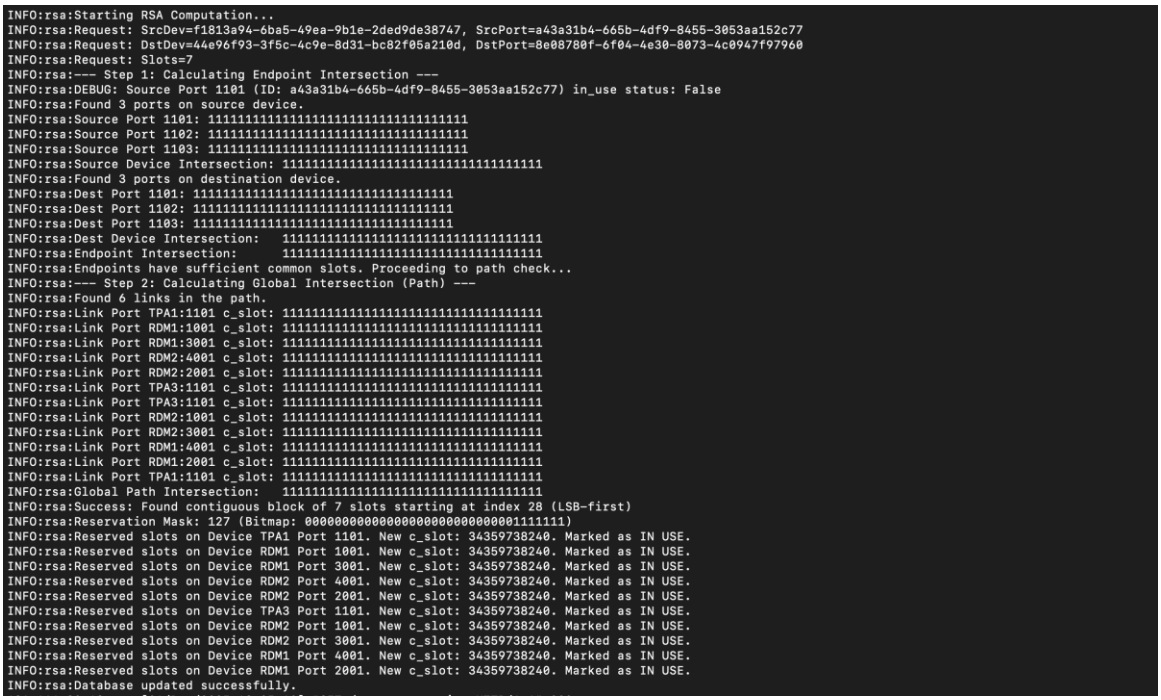
- Then we will call “all_simple_paths()” and store them in another “list”: will hold enormous data 100M paths, if the topology is very much complex.

Now that we have stored the main topology and paths saved in memory, we pay less costs for DB queries (I am saying less because while implementing the main solution there might be something new).

How to decide the required slots?

Well, right here we can use the concept of Professor Andrea. For example, the first node is TP1, and the next hop is RDM1. We will check how many ports we have in TPA1 and RDM1. Then:

- We will generate the bitmaps of those ports and perform an intersection with the whole C/L band (**The big pipe).**
- The intersection returns slots with 1s and rest 0s. Then we iterate to determine the slot number. I have tested this; here are the results.



- **This is not TeraFlowSDN, I have developed a separate FLASK application. It is running through docker, it saves time. Otherwise, we have to rebuild TeraFlowSDN each time we make changes in the code.**

First proposed solution: Parallelization of the RSA computation

We can now span THREADs (logical cores) = machine's logical cores – 2. The minus 2 is not necessary, but the Operating System has other things running as well. If we spawn threads = machine's logical cores, then either other processes will be slowed down, or our computation will be slowed down (this situation is experienced by me from the SPM project, but it is not always true. How many threads will scale best our program; we can only tell after running tests in the machine which is running the SDN controller (mascara)). We can come to this later, let us go through the process:

- We span T threads and divide our paths by T: 100M/T = 12.5M (using 8 threads)
- Now we pay 8 times less than before
- Why will this work?
  It will work because our operation is logical, not arithmetical. Logical in a sense because we are just doing AND operation and then compare and swap (CAS). None of these threads will have to write any data in the memory (because only shared write creates contention and race conditions). There are cache related issues, but we can solve this by studying the size of each path in Bytes. Each core has 31KiB local cache. If we can utilize full 31KiB, we pay nothing for the AND operation and CAS for paths = 31KiB.
- Now each thread will have their own private list (first private)
- And they will store the successful path information where all the ports have available slots.
- If any port from any path fails, the whole path will be ignored and not suggested
- Finally, I will combine all the results into one list and check which one fits me better. At this point you can use Dijkstra's algorithm for the shortest path. Or you can perform the Dijkstra algorithm in the first networkX graph and perform the RSA on that.

At this point something came to mind is that this computation is not for finding the best path. As my professor said, finding the path is not the challenge; the challenge is to find the available slots for RSA. Now it is clear to me, even if I have the shortest path I must pay this computational cost. (I want your comments based on this understanding of mine).

Second proposal: Edge Pruning of the graph using networkX

Even though I do not know if it is a feasible solution because I have questions about this, but I found some references on this solution. I will put my questions after the steps:

- We perform the same pre-compute
- We remove the links which are already in use (first question)
- We produce a simplified graph

- Now we can perform either sequential computation or parallel one. But I think even after pruning we can have parallel links and it can be complex.
- We decide the slots
- We perform Dijkstra on the simplified graph and then on that path we set up our intent.

My Question:

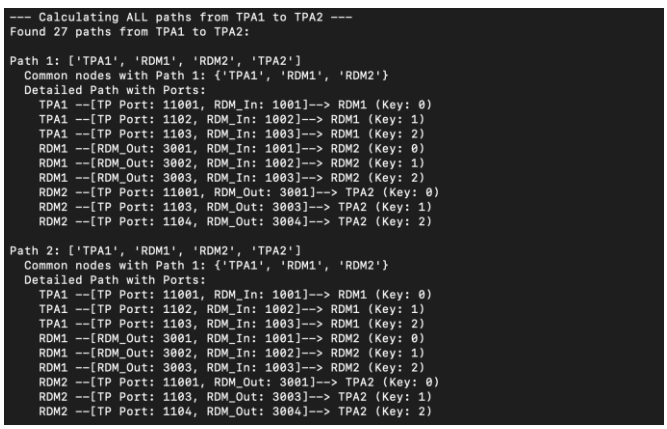
- Why would I prune the edge?
- For example, I can have 10 slots from C band used in TP1, port 1101. I still have N usable slots in the same port.
- Or if we consider an OMS port which is in use but still can have N slots available, if we prune the edges based on port in_use=true we are not taking the benefits of DWDM. Are we?
- Now, this is something I am not sure professor. Because I searched for references, some says it happens in real life. Some says it is hard to modify an existing channel to push more channels. I want your comments on this.

Third Solution: AI based solutions

This can be very stupid, but I am just sharing because these days AI is a hype. If we can parallelize the code and be sure about the data-structure, we can save the current RSA in a shared memory space (disk is preferred). Then we can generate a prompt to perform a pre-computation using AI for the new intent. If the result is true positive, we can start calculating, otherwise we discard the intent and save time. But this needs a lot of work and intensive tests. Defining the data structure will be most challenging part, but it is possible.

Tests that I have performed:

To test the theories, I have tried generating the simplified graphs using networkX. Then I have generated lists based on that graph. The programming part is not essential now but just to have an idea I am adding some screenshots:



Among 4 nodes and 3 hops with 3 parallel links (uniform) we had 27 paths (used networkX).

```
--- Calculating ALL paths from TPA1 to TPA2 (Narrow Topology) ---
Found 9 paths from TPA1 to TPA2:

Path 1: ['TPA1', 'RDM1', 'RDM2', 'TPA2']
  Detailed Path with Ports:
    TPA1 --[TP Port: 11001, RDM_In: 1001]--> RDM1 (Key: 0)
    TPA1 --[TP Port: 1102, RDM_In: 1002]--> RDM1 (Key: 1)
    TPA1 --[TP Port: 1103, RDM_In: 1003]--> RDM1 (Key: 2)
    RDM1 --[RDM_Out: 3001, RDM_In: 1001]--> RDM2 (Key: 0)
    RDM2 --[TP Port: 11001, RDM_Out: 3001]--> TPA2 (Key: 0)
    RDM2 --[TP Port: 1103, RDM_Out: 3003]--> TPA2 (Key: 1)
    RDM2 --[TP Port: 1104, RDM_Out: 3004]--> TPA2 (Key: 2)

Path 2: ['TPA1', 'RDM1', 'RDM2', 'TPA2']
  Detailed Path with Ports:
    TPA1 --[TP Port: 11001, RDM_In: 1001]--> RDM1 (Key: 0)
    TPA1 --[TP Port: 1102, RDM_In: 1002]--> RDM1 (Key: 1)
    TPA1 --[TP Port: 1103, RDM_In: 1003]--> RDM1 (Key: 2)
    RDM1 --[RDM_Out: 3001, RDM_In: 1001]--> RDM2 (Key: 0)
    RDM2 --[TP Port: 11001, RDM_Out: 3001]--> TPA2 (Key: 0)
    RDM2 --[TP Port: 1103, RDM_Out: 3003]--> TPA2 (Key: 1)
    RDM2 --[TP Port: 1104, RDM_Out: 3004]--> TPA2 (Key: 2)
```

Here I have narrowed down the intermediate links to form a non-uniform link pattern, and it scaled to the second equation we derived.

<u>Some topologies using NetworkX:</u>