TELECOMMUNICATION NETWORK ROUTING SYSTEM OCCUPYING MINIMUM NUMBER OF CHANNELS

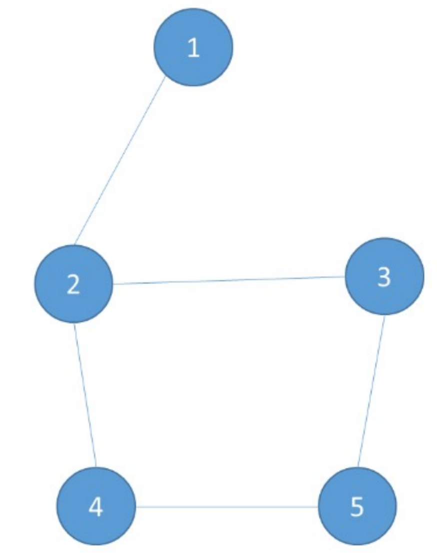
TANI HOSSAIN

FATMA NOOR

KUSHAL KHATRI

DHRUVAL PATEL

JAHNAVI BOYA



(s1,t1) = (1,3), d1 = 2
(s2,t2) = (1,5), d2 = 2
(s3,t3) = (3,4), d3 = 3
B = 4

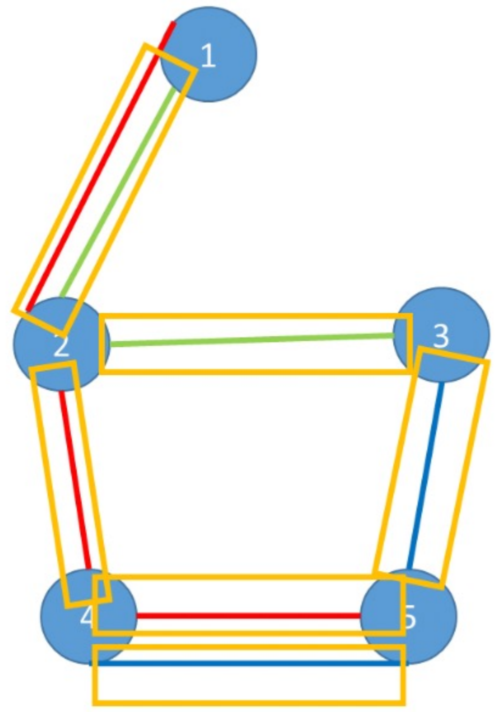
Input:

PROBLEM PARAMETERS

- Telecommunication optical network represented by a multigraph G
- Each edge of the graph represents a signal transmission channel with a bandwidth B
- k transmission demands
- Each demand i has
  - a source s_i
  - a destination t_i
  - size d_i

PROBLEM STATEMENT

- Routing of all transmission demands, in such a way that number of used transmission channels are minimum



P1 = (1-2-3),
P2 = (1-2-4-5),
P3 = (3-5-4),

ASSUMPTIONS

- The optical network will be a connected graph: there will always be a path between source and destination
- The network graph edges are undirected
- There can be multiple paths for each transmission demands
- Demand size will always be less than Bandwidth
- Multiple demands possible from same source and destination

PROBLEM DIVISION – TWO PARTS

Path Finding

Bin Packing

WHY BIN PACKING?

Let’s Assume,

- Channels with fixed bandwidth as bins with fixed capacity
- Transmission demands with different sizes as tasks with various durations
- **Need to route the demands in such a way so that minimum number of Channels/Bins needed**

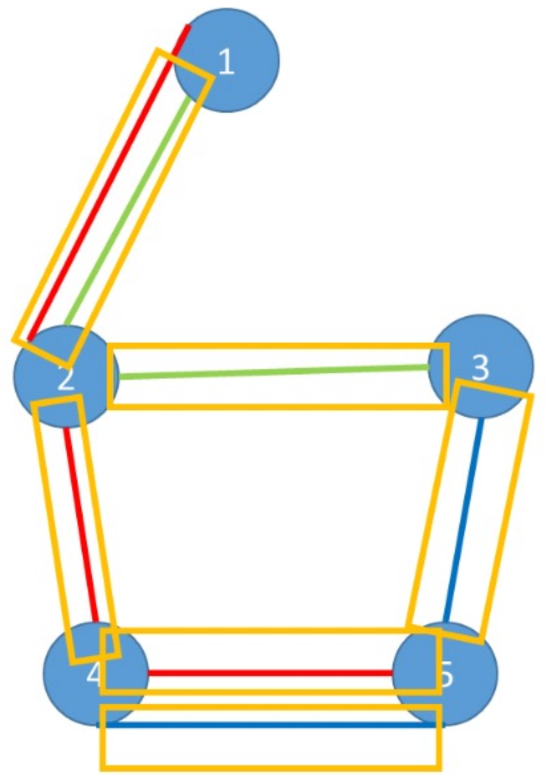
INITIAL NAIVE APPROACH

1. Took any of the existing path between source and destination
2. Generated cost list for all the demand paths
3. Size of cost list: ∑(# of edges in path) for all demands
4. Passed the **whole** cost list to Bin Packing Algorithm
5. Find number of channels needed

CHALLENGES WITH INITIAL APPROACH

For the Given Graph:

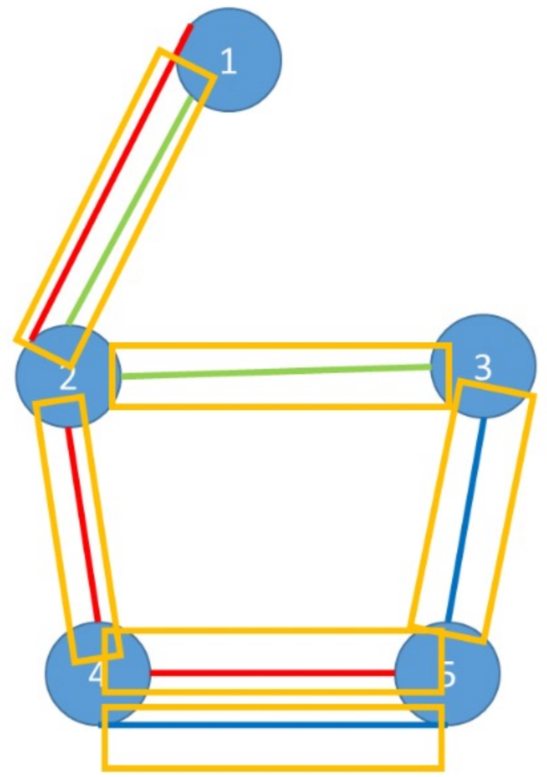
- Expected output = 6
- Output Received = 5



P1 = (1-2-3),
P2 = (1-2-4-5),
P3 = (3-5-4),

ANALYZING INITIAL APPROACH

- List of sizes = [2, 2, 2, 2, 2, 3, 3]
- Capacity = 4
- Bins = [[2, 2], [2, 2], [2], [3], [3]] => 5
- Channels = [[2, 2], [2], [2], [2], [3], [3]] => 6



P1 = (1-2-3), =>2
P2 = (1-2-4-5), =>2
P3 = (3-5-4), =>3

DECIDING FACTORS

- Edges not shared by demands will always need a whole channel, even if channel is not filled to capacity
- We need to apply bin packing only to shared edges
- Number of channels required can differ largely based on the selected paths
- Therefore, **path selection** is the most critical part for solving this problem

TWO APPROACHES

- Greedy Algorithm
- Local Search

APPROACH 1 – GREEDY ALGORITHM

- A greedy algorithm is an algorithmic strategy that makes the best optimal choice at each small stage with the goal of this eventually leading to a globally optimum solution.
- Here, we have taken **shortest path** from source to destination for each demand

APPROACH 1 – STEPS

Select shortest path for each demand and store it

To find number of transmission channels for the selected path list:

1. Find overlapping edges
2. List costs for each overlapping edge
3. Do bin packing for edge overlapping edge cost list
4. Add number of acquired bins for edge to # of channels
5. Add 1 channel for each Non-Overlapping edge
6. Return total # of channels

APPROACH 2 – LOCAL SEARCH

- Local search is a heuristic method for solving computationally hard optimization problems
- The local search algorithm explores and evaluates different solutions (search space) by applying local changes until an optimal solution is achieved or certain iterations are computed.
- Hill climbing, simulated annealing, tabu search are some of the local search algorithms
- Here, we are using hill climbing approach for finding minimum # of channels

HILL CLIMBING APPROACH

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.
- It terminates when it reaches a peak value where no neighbor has a higher value.
- A node of hill climbing algorithm has two components which are state and value.
- Local maximum is a state which is better than its neighbor states, but other better states exist
- Global maximum is the best possible state of state space landscape.
- Current state is a state in a landscape diagram where an agent is currently present.

HILL CLIMBING COMPONENTS

- State: Index of paths to be selected from all possible paths
- Value: List of selected paths for each iteration
- Neighbors: Any state other than the current state
- Optimal Neighbor: neighbor whose objective function value is less than current state
- Objective Function(Min): Calculating number of channels for selected path
- Iteration bound: Number of times to run the iterations

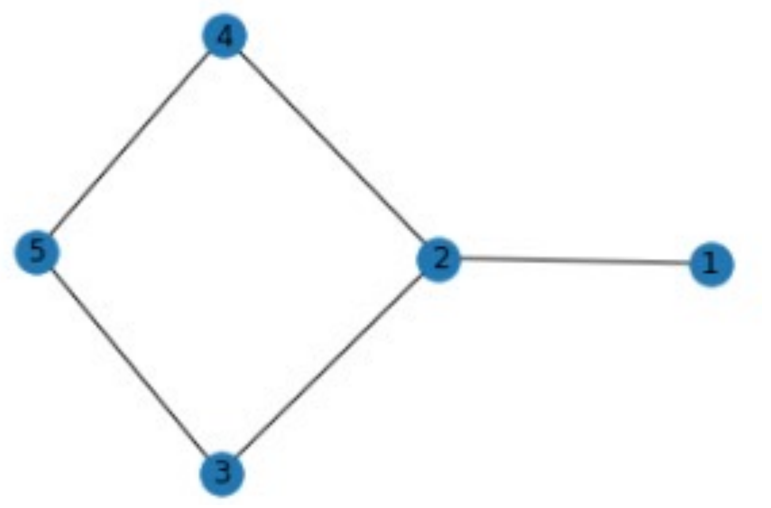
APPROACH 2 – STEPS

- Find all possible paths for each demand
- At first, Select one path for every demand randomly – this is our initial state value
- Find number of transmission channels for the selected path list
- Get optimal neighbor for the current state by comparing objective functions
- If different current state and neighbor state have same objective value, take random neighbor state to avoid local maxima
- If neighbor state is same as current state terminate because that means no other optimal solution is left
- Otherwise, terminate when exceeds the iteration bound

INPUT AND VISUALIZING OUTPUTS

- Random input generator for testing
  - Generated Random graph
  - Generated number of demands and demand configuration randomly based on the given graph
- Output
  - Showing necessary steps for each algorithm
  - Finally showing number of channels acquired from both the approaches

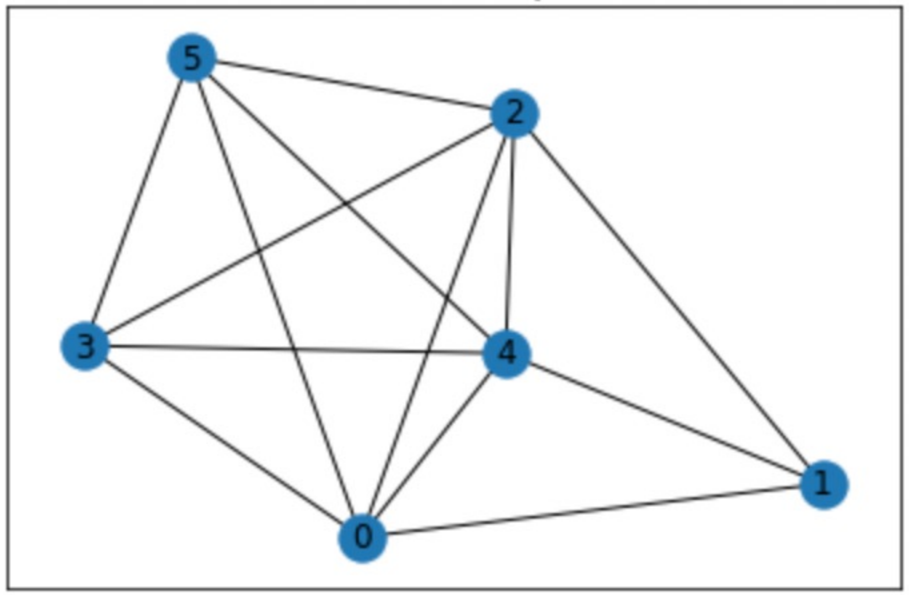
OUTPUT FOR GIVEN GRAPH



Shortest Path List for Greedy: [[1, 2, 3], [1, 2, 3, 5], [3, 2, 4]]

```
Results:
Minimum Number of Transmission Channels (Local Search) =  6
Minimum Number of Transmission Channels (Greedy  Algo) =  5
```

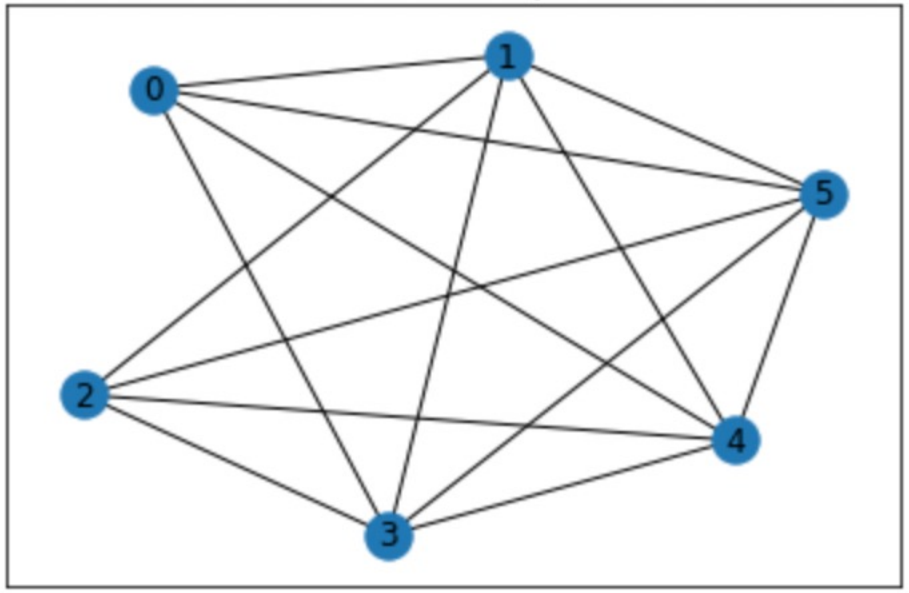
SOME EXAMPLE RESULTS



```
Results:
Minimum Number of Transmission Channels (Local Search) =  8
Minimum Number of Transmission Channels (Greedy  Algo) =  5
```

```
Num of demands:  4
All source sinks:  [[4, 2], [4, 5], [1, 3], [0, 3]]
Bandwidth:  6
Cost of all demands:  [1, 1, 6, 1]
```

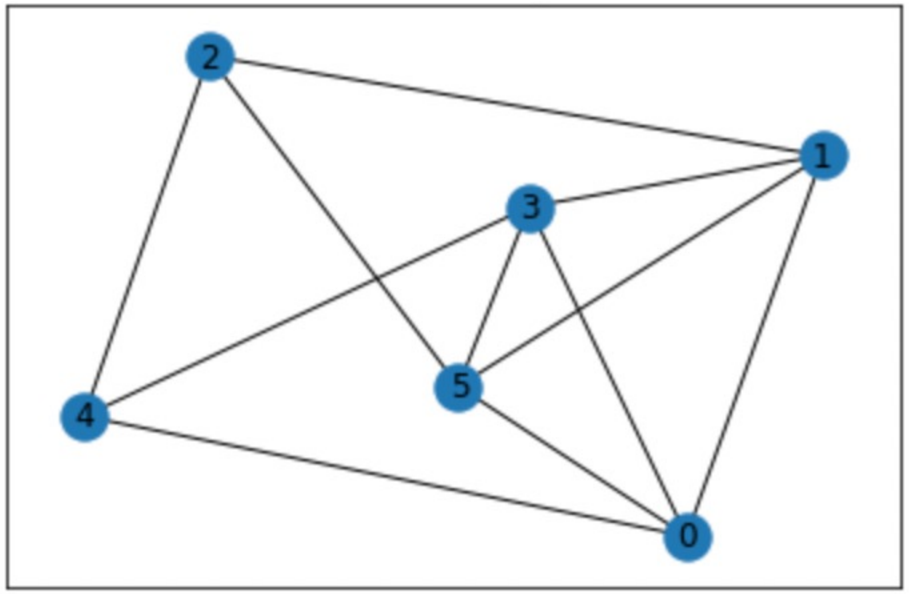
SOME EXAMPLE RESULTS



```
Num of demands:  2
All source sinks:  [[1, 2], [0, 1]]
Bandwidth:  6
Cost of all demands:  [3, 3]
```

```
Results:
Minimum Number of Transmission Channels (Local Search) =  2
Minimum Number of Transmission Channels (Greedy  Algo) =  2
```

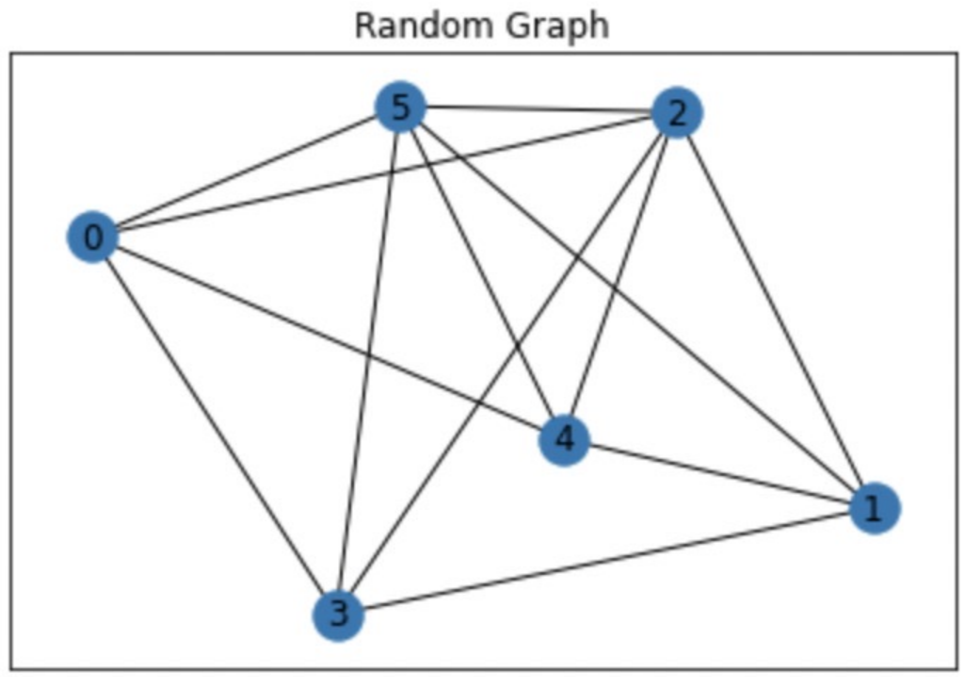
SOME EXAMPLE RESULTS



```
Num of demands:  1
All source sinks:  [[2, 4]]
Bandwidth:  7
Cost of all demands:  [3]
```

```
Iterations left:  18
Results:
Minimum Number of Transmission Channels (Local Search) =  1
Minimum Number of Transmission Channels (Greedy  Algo) =  1
```

SOME EXAMPLE RESULTS

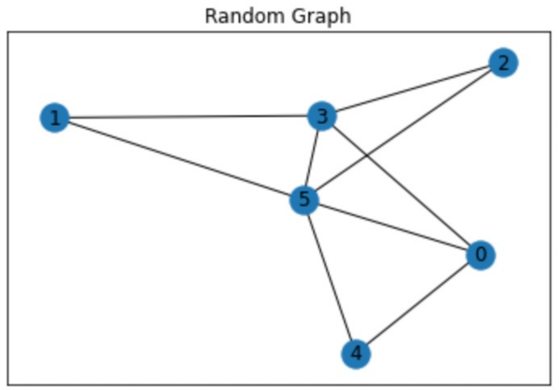


```
Num of demands:  3
All source sinks:  [[4, 0], [1, 5], [2, 4]]
Bandwidth:  5
Cost of all demands:  [3, 3, 5]
```

```
Iterations left:  16
Results:
Minimum Number of Transmission Channels (Local Search) =  3
Minimum Number of Transmission Channels (Greedy  Algo) =  3
```

```
Iterations left:  63
Results:
Minimum Number of Transmission Channels (Local Search) =  4
Minimum Number of Transmission Channels (Greedy  Algo) =  5
```

```
Input:
[(0, 5), (0, 4), (0, 3), (1, 3), (1, 5), (2, 5), (2, 3), (3, 5), (4, 5)]
```



```
Num of demands:  3
All source sinks:  [[1, 4], [4, 0], [2, 1]]
Bandwidth:  6
Cost of all demands:  [2, 2, 3]
```

SOME EXAMPLE RESULTS

- Local search solution state path: [[1, 5, 0, 4], [4, 0], [2, 5, 1]]
- Greedy Algorithm Shortest path: [[1, 5, 4], [4, 0], [2, 3, 1]]
- Local search does not take shortest path still gives optimal solution

COMPARISON BETWEEN TWO APPROACHES

Approach 1

- Greedy approach uses momentary optimal combination
- Assumes current solution is optimal so, no extra iterations needed
- Generates consistent results every time for the same set of input
- Results can vary depending on which shortest path we select if there exist multiple shortest paths

Approach 2

- Local Search checks a number of combinations of paths
- Local search takes much more iteration to give a result compared to greedy in this case
- Solution can defer for the same set of input as random states are taken every time
- Results can be same even if we start with different initial states

CONCLUSION

It is not guaranteed that any of the approaches used will give optimal solution

Local search will converge and give us optimal solution if it completes

However, to complete local search and get global maxima it takes a lot of time

An iteration bound was introduced to terminate the search after a certain time

When local search is terminated by iteration bound, there remains a possibility of getting local maximum

The larger the iteration bound is the better the local search outcome will be

Therefore, in our case we will suggest Greedy Approach as it consumes **less time/iteration** and produces same output for same input

Nevertheless, for **more complicated and larger graphs** Local search will be better than Greedy Algorithm

IMPLEMENTATION

- Install binpacking before executing the code
- Two versions of the code added
  - For given graph
  - For random graphs
- Each version includes both the approaches
- Google Colab Link

THANK YOU