Project Proposal

Kyle Rumpza

rumpz020

1 Problem Brief

The traveling salesman problem (TSP) is a widely studied puzzle in mathematics and computer science in which you are trying to find the shortest path to a destination. We have studied it in class through looking at how different search algorithms work on graphs to solve the puzzle with varying success. However TSP is also applicable to the world of computer networks. Because really the internet is one big graph where nodes can represent routers and hosts, and edges the distances of physical mediums. That is my approach to design a graph network in which packets will travel through a path generated by various algorithms and which incur delays simalar to real-world routers. By testing different algorithms on the network it will give insight in how machine learning can improve routing in networks.

Approach to solving

1.1 Network Representation

I have created a graph representation of a network. The nodes represent routers and have attributes for the amount of delay, propagation delay, and traffic congestion at that router. Edges between nodes represents the physical distance of the link. I have also created a packet object which has an attribute for length, and if its the first packet. This is the basic layout of the network.

1.2 Algorithms

For testing the algorithm there are two types to consider: 1. Uninformed 2. Informed. For the informed algorithms they use a heuristic which for these tests will be based on avoiding routers with high traffic congestion, or high delays. This is because even if the distance is physically longer from the start node to the end node, the transmission speed may still be faster

1.3 Simulation

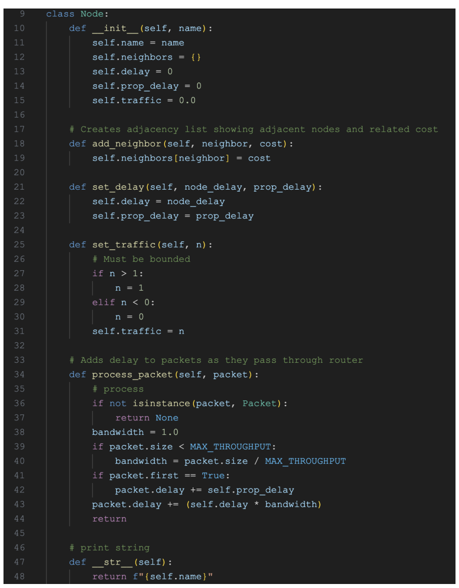
The algorithms I plan on testing are UCS, A*, Genetic, Simulated Annealing, Monte Carlo, and Ant Colony Optimization. To get the run time packets are sent to each node given by the path generated by the various algorithms, incurring delay along the way. The algorithms will be tested in two different types of networks. One will be static where traffic intensity doesn't change, and a dynamic network where traffic intensity does change as packets pass through.

2 Software

I will be building this project using python. I am taking an OOP approach to a graph based network to run the simulations on. I am using textbook pseudo-code and publically available python code (from websites like Medium) to formulate the implemenation of UCS, A*, Genetic, Simulated Annealing, Monte Carlo, and ACO algorithms. The software implementation will see each algorithm tested on the network and the results will be printed out in a table. The main file will run the various tests without requiring user input.

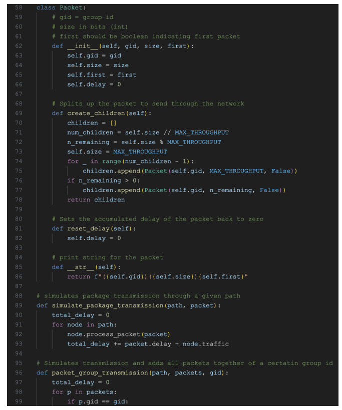
3 Preliminary Work

I have already started working on the project here is some of the code i have completed so far. The first image is the node class and its related functions. Process packet being the most important one since it adds the delay to packet objects as they encounter each node.



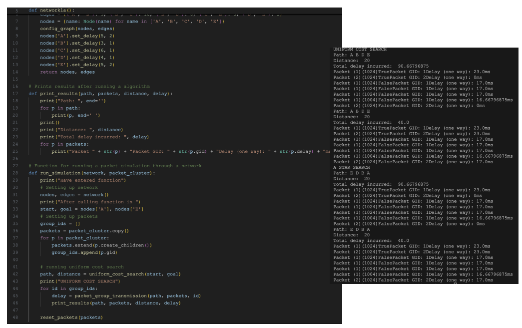
```python
class Node:
    def __init__(self, name):
        self.name = name
        self.neighbors = {}
        self.delay = 0
        self.prop_delay = 0
        self.traffic = 0.0

    # Creates adjacency list showing adjacent nodes and related cost
    def add_neighbor(self, neighbor, cost):
        self.neighbors[neighbor] = cost

    def set_delay(self, node_delay, prop_delay):
        self.delay = node_delay
        self.prop_delay = prop_delay

    def set_traffic(self, n):
        # Must be bounded
        if n > 1:
            n = 1
        elif n < 0:
            n = 0
        self.traffic = n

    # Adds delay to packets as they pass through router
    def process_packet(self, packet):
        # process
        if not isinstance(packet, Packet):
            return None
        bandwidth = 1.0
        if packet.size < MAX_THROUGHPUT:
            bandwidth = packet.size / MAX_THROUGHPUT
        if packet.first == True:
            packet.delay += self.prop_delay
        packet.delay += (self.delay * bandwidth)
        return

    # print string
    def __str__(self):
        return f"{self.name}"
```

This is the packet class which carries a size attribute, group id attribute, and a first boolean. It has a function for creating children when the size is over a max throughput limit of 1024, each child has the same group id as parent but it is not the first or header packet. Below the packet class are functions for sending packets through a path of nodes, getting processed at each one.



```python
class Packet:
    # gid = group id
    # size in bits (int)
    # first should be boolean indicating first packet
    def __init__(self, gid, size, first):
        self.gid = gid
        self.size = size
        self.first = first
        self.delay = 0

    # Splits up the packet to send through the network
    def create_children(self):
        children = []
        num_children = self.size // MAX_THROUGHPUT
        n_remaining = self.size % MAX_THROUGHPUT
        self.size = MAX_THROUGHPUT
        for _ in range(num_children - 1):
            children.append(Packet(self.gid, MAX_THROUGHPUT, False))
        if n_remaining > 0:
            children.append(Packet(self.gid, n_remaining, False))
        return children

    # Sets the accumulated delay of the packet back to zero
    def reset_delay(self):
        self.delay = 0

    # print string for the packet
    def __str__(self):
        return f"({self.gid})({self.size}){self.first}"

# simulates package transmission through a given path
def simulate_package_transmission(path, packet):
    total_delay = 0
    for node in path:
        node.process_packet(packet)
        total_delay += packet.delay + node.traffic

# Simulates transmission and adds all packets together of a certatin group id
def packet_group_transmission(path, packets, gid):
    total_delay = 0
    for p in packets:
        if p.gid == gid:
```

Lastly is the main function where i show how i create the network graph and run test simulations using a star search and uniform cost search. Attached are the results from running a packet cluster of two packets through the shown network.



```python
def network1a():
    nodes = {name: Node(name) for name in ['A', 'B', 'C', 'D', 'E']}
    config_graph(nodes, edges)
    nodes['A'].set_delay(5, 2)
    nodes['B'].set_delay(3, 1)
    nodes['C'].set_delay(6, 1)
    nodes['D'].set_delay(4, 1)
    nodes['E'].set_delay(5, 2)
    return nodes, edges

# Prints results after running a algorithm
def print_results(path, packets, distance, delay):
    print("Path: ", end='')
    for p in path:
        print(p, end=' ')
    print()
    print("Distance: ", distance)
    print("Total delay incurred: ", delay)
    for p in packets:
        print("Packet " + str(p) + "Packet GID: " + str(p.gid) + "Delay (one way): " + str(p.delay) + "m

# Function for running a packet simulation through a network
def run_simulation(network, packet_cluster):
    print("Have entered function")
    # Setting up network
    nodes, edges = network()
    print("After calling function in ")
    start, goal = nodes['A'], nodes['E']
    # Setting up packets
    group_ids = []
    packets = packet_cluster.copy()
    for p in packet_cluster:
        packets.extend(p.create_children())
        group_ids.append(p.gid)

    # running uniform cost search
    path, distance = uniform_cost_search(start, goal)
    print("UNIFORM COST SEARCH")
    for id in group_ids:
        delay = packet_group_transmission(path, packets, id)
        print_results(path, packets, distance, delay)

    reset_packets(packets)
```

```
UNIFORM COST SEARCH
Path: A B D E
Distance:  20
Total delay incurred:  90.66796875
Packet (1)(1024)TruePacket GID: 1Delay (one way): 23.0ms
Packet (2)(1024)TruePacket GID: 2Delay (one way): 0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1004)FalsePacket GID: 1Delay (one way): 16.66796875ms
Packet (2)(1024)FalsePacket GID: 2Delay (one way): 0ms
Path: A B D E
Distance:  20
Total delay incurred:  40.0
Packet (1)(1024)TruePacket GID: 1Delay (one way): 23.0ms
Packet (2)(1024)TruePacket GID: 2Delay (one way): 23.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1004)FalsePacket GID: 1Delay (one way): 16.66796875ms
Packet (2)(1024)FalsePacket GID: 2Delay (one way): 17.0ms
A STAR SEARCH
Path: E D B A
Distance:  20
Total delay incurred:  90.66796875
Packet (1)(1024)TruePacket GID: 1Delay (one way): 23.0ms
Packet (2)(1024)TruePacket GID: 2Delay (one way): 0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1004)FalsePacket GID: 1Delay (one way): 16.66796875ms
Packet (2)(1024)FalsePacket GID: 2Delay (one way): 0ms
Path: E D B A
Distance:  20
Total delay incurred:  40.0
Packet (1)(1024)TruePacket GID: 1Delay (one way): 23.0ms
Packet (2)(1024)TruePacket GID: 2Delay (one way): 23.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1024)FalsePacket GID: 1Delay (one way): 17.0ms
Packet (1)(1004)FalsePacket GID: 1Delay (one way): 16.66796875ms
Packet (2)(1024)FalsePacket GID: 2Delay (one way): 17.0ms
```

4 Evaluation

Each algorithms effectiveness will be tested by a variety of factors. The path generated by the algorithm will tell us if an optimal path is found, in other words if its complete. The packets sent through the network will give us the run time. In dynamic networks we can test the traffic intensity of the nodes to test for load balancing. And by contrasting results between the static and dynamic network we can see which algorithms are better in adapting to network changes.

Attributes to test for: Run time, completeness, load balancing, adaptability

5 Time Frame

If we assume each phase takes roughly a week - 2 weeks

Phase 1) Do more testing on different networks with more nodes, edges, and different delays. This will ensure the algorithms are consistent.

Phase 2) Add more search algorithms. This will include genetic algorithms, simulated annealing, monte carlo search, and possibly ant colony optimization.

Phase 3) Develop dynamic network where packets passing through a router increase traffic intensity. Test the algorithms on both the static and dynamic network. Make sure main produces a finalized table.

Phase 4) Collect results and finalize report.

References

[1] Christa Fernandes. *How to solve a routing problem with a genetic algorithm: A practical guide*. https://medium.com/data-and-beyond/how-to-solve-a-routing-problem-with-a-genetic-algorithm-a-practical-guide-a0f0f8aa36db. Accessed: 2024-10-21. Nov. 2023.

[2] Allanah Francis. *Traveling salesman problem using simulated annealing*. https://medium.com/@francis.allanah/travelling-salesman-problem-using-simulated-annealing-f547a71ab3c6. Accessed: 2024-10-21. Jan. 2022.

[3] Xunchi Ma. *A summary of the routing algorithm and their optimization,performance*. 2024.

[4] Peter Norvig Stuart J. Russel. *Artificial Intelligence: A modern approach*. Upper Saddle River: Pearson, 2010.

[5] Dongming Zhao, Liang Luo, and Kai Zhang. "An improved ant colony optimization for the communication network routing problem". In: *Mathematical and Computer Modelling* 52.11 (2010), pp. 1976–1981. DOI: https://doi.org/10.1016/j.mcm.2010.04.021. URL: %5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/S0895717710002116%7D.

[3] [1] [2] [4] [5]