ROUTE FINDER

Optimal Train Route Generator Using
Modified A* Search

ITIT-2203 Artificial Intelligence Project

Aaryak Shah
2019IMT-001
IPG MTech

ABSTRACT

Route Finder is an Artificial Intelligence tool that allows users to find the most time-efficient route for their train journey. The program makes use of a *heuristic algorithm* that is based on a standard *tree approach A* search*. The user may enter the *origin* and *destination* stations, as well as the time at which they would like to *start* the trip. The algorithm proceeds to calculate the fastest route considering both the *travel time* between stations and *waiting time* for switching trains.

> Note: The stations and travel durations used to test the project are loosely based on real conditions but do not directly represent accurate travel times. As such the given data should be treated as dummy data.

INTRODUCTION

> The project code can be found linked here and at the end of this document:
>
> **GitHub** - https://github.com/aaryak-shah/route-finder

This project deals with the problem of applying heuristic search AI to a scheduled transportation system, like a train, plane or bus network. What differentiates this class of transport is that on arriving at any node, the agent must wait for fixed departure times to travel to future nodes.

The A* search algorithm is a very popular choice for common search problems as it guarantees an optimal solution (given right conditions).

More importantly it is a heuristic algorithm, and hence it is able to find the best solution faster and while using less space.

Hence, it can be applied very easily and conveniently to a simple travel problem requiring the agent to go from some location A to some B. However such a problem makes an assumption that traversal between nodes can freely occur at any time, i.e. there is virtually no delay between arriving at a node and departing to the next one in the path. Practically, this makes sense for personal travel such as cars and bikes, or even on foot.

However we cannot directly apply A* on a problem where transport is scheduled. For example any journey that requires travel by trains, airplanes or buses. The reason for this is that the shortest travel path may not be the most optimal if you factor in waiting duration for the next departure at each stop. However this drawback can be fixed easily by slightly modifying the evaluation function, as will be explained in the next section.

METHODOLOGY

As we already know, A* uses the evaluation function

$$f(n) = h(n) + g(n)$$

Where $h(n)$ is the heuristic value of the node and $g(n)$ is the true cumulative cost. We modify this by introducing in a third term $w(n)$ to the function. It represents the waiting duration till the current node. In practise we can calculate waiting between previous and current and merge it with the $g(n)$ term at each step, then perform an A* search as usual.

Now, we can write the algorithm as follows (next page):

```
Algorithm findRoute():
    current := startState
    frontier := [ current ]
    While frontier is not empty:
        Generate successors
        Remove current from frontier
        For Each successor:
            Generate wait durations
            Apply evaluation function
            Append successor to frontier
        current := node in frontier with min value
        If current is destination:
            Generate route
            Return
```

The code corresponding to this algorithm can be found in the GitHub repository linked above, in the file `/ai.py`

The algorithm follows the Tree Search paradigm. Which means that the agent does not keep track of states that have already been fully explored. The reason for this is that durations in scheduled travel depend on many factors like the type of vehicle, route traffic, waiting times etc. and so it is not easy to ensure consistency in the graph. However, the heuristic is at least admissible. Hence we choose Tree Search which guarantees optimality.

The agent performs search on an Undirected Graph where each node maintains track of a Propositional State, as well as a method to generate successor nodes. Practically the node also needs to store it's parent node in a recursive form.

The path costs are the number of hours an average train would require to travel between two connected nodes. The heuristic is the amount of time the same train would take to travel a straight line distance.

RESULT

We can use the Heuristic Approach to search problems in a very flexible manner, to fit the requirements of our problem. This modified A* algorithm factors in the strict, scheduled nature of our problem and provides us with an optimal solution. Demo run provided on page 8.

DISCUSSION

This project is based on a simple approximation of a real scheduled transport system. We make various assumptions for the sake of simplicity.

For instance, we assume that all trains between two directly connected cities travel at the same speed. However in reality this is not true. With more complex data and a better $g(n)$ calculating logic, this can be solved.

Also, we have ignored the monetary aspect of the problem. A possible way to address this could be to change $g(n)$ to mean price/hour. Alternatively we could add a quaternary term similarly to $w(n)$.

CONCLUSION

This project explored the heuristic search paradigm and provided a different way to understand the A* algorithm through modification. I was able to better understand how search AI function and is implemented practically.

CODE

The most important parts of the code are given here for quick reference. Use the GitHub Repository to view the entire project.

`> project > ai.py > find_route() >`

```
def find_route(depart, arrive, start):
  current = Node(depart, arrive)
  current.arrival_time = start
  exploring = [current]
  while (exploring):
    successors = current.successors()
    exploring.remove(current)
    for successor in successors:
      exploring.append(successor)
    current = exploring[0]
    for node in exploring:
      if (node.evaluated_value < current.evaluated_value):
        current = node
    if (current.station is arrive):
      return util.generate_route(current)
```

[P.T.O]

> project > models.py > Node >

```
class Node:
  def __init__(self, station, destination):
    self.station = station
    self.name = data.cities[station]
    self.destination = destination
    self.departures = data.departures[station]
    self.next_stop_costs = data.costs[station]
    self.parent_node = None
    self.arrival_time = -1
    self.chosen_departure = -1
    self.heuristic = data.heuristics[station][destination] # h(n)
    self.cumulative = 0 # g(n)
    self.wait_time = 0 # w(n)
    self.evaluated_value = self.heuristic # f(n)

  def successors(self):
    result = []
    for i in range(len(self.next_stop_costs)):
      if (self.next_stop_costs[i] > 0):
        node = Node(i, self.destination)
        node.parent_node = self
        available_departures =
node.parent_node.departures[node.station]
        wait_times = []
        for departure_time in available_departures:
          if (departure_time < node.parent_node.arrival_time):
            wait_times.append((24 - node.parent_node.arrival_time) +
departure_time)
          else:
            wait_times.append(departure_time -
node.parent_node.arrival_time)
        node.wait_time = min(wait_times)
        min_index = wait_times.index(node.wait_time)
        node.chosen_departure = available_departures[min_index]
        node.arrival_time = (node.chosen_departure +
node.parent_node.next_stop_costs[node.station]) % 24
        node.cumulative = node.parent_node.cumulative +
node.parent_node.next_stop_costs[node.station] + node.wait_time
        node.evaluated_value = node.cumulative + node.heuristic
        result.append(node)
    return result
```

> project > util.py > generate_route() >

```
def generate_route(final_node):
  route = []
  current = final_node
  while (current is not None):
    route.append(current)
    current = current.parent_node
  if (is_valid_route(route)):
    return formatted(route[::-1])
  else:
    return []
```

> project > util.py > formatted() >

```
def formatted(route):
  #   > FORMAT
  #   > [(station, arrival-time, departure-time, elapsed-time)]
  #   > sample route for bangalore to gwalior:
  #   route = [
  #     ('BLR', -1, 14,  0),
  #     ('MUM', 10, 10, 20),
  #     ('GWL',  2, -1, 36)
  #   ]
  final = []
  path_length = len(route)
  for i in range(path_length):
    route.append(Node(0, 0))
    node = route[i]
    next_node = route[i+1]
    final.append((node.name, node.arrival_time,
next_node.chosen_departure, node.cumulative))
  return final
```

[P.T.O]

DEMO

A demo run of the project has been given for easy understanding of the functioning.