Solution Summary

Approach:

Translation to a Search Graph
The problem was translated into a graph structure where:

- Nodes represent train stations.
- Edges represent train routes between stations with attributes such as departure time, arrival time, and travel distance.

Two types of graphs were constructed:

1. Mini-Schedule Graph for simplified problems.
2. Full-Schedule Graph for complex problems.

These graphs were expanded to include virtual start and end nodes, allowing for dynamic calculations like waiting times and adjustments based on input times.

Choice of Search Algorithm

We used a custom graph traversal approach tailored to the problem's requirements:

- Dijkstra-like algorithm was implemented to handle weighted edges efficiently.
- The choice of traversal methods was guided by the nature of the cost function.Dijkstra ensures the shortest weighted paths.
- Graph Adjustments: For arrival time problems, we dynamically adjusted edge weights for specific start nodes to incorporate waiting times at stations. This added complexity but improved solution accuracy and performance.

Key Insights

Graph Expansion for Detailed Timing

By introducing detailed arrival and departure nodes with attributes like departuretime and arrivaltime, the solver captured realistic scheduling details.

- **Intra-Station Connections**: Arrival nodes were linked to departure nodes within the same station, modeling waiting times.
- **Virtual Nodes**: Virtual start and end nodes ensured all possible transitions were represented, even for complex schedules.

Adjusting Start Times for Arrival-Time Cost Function

- **Dynamic Wait Time Calculation**: The function recalculates the time from the virtual start node to neighboring departure nodes by computing the waiting time between the input time and the scheduled departure time.
- **Flexibility in Input Timing**: This adjustment ensures that the solver accurately incorporates user-provided input times.

Performance and Evaluation:

- Graph Expansion: Virtual start and end nodes reduced the complexity of handling multiple schedules.
- Efficient Iteration: A single loop was used to evaluate all problem types, minimizing redundant computations.

What Worked Well:

Use of NetworkX Library: Utilizing the NetworkX library for graph representation and manipulation was a significant advantage..

Adapting Dijkstra's Algorithm: It ensured that the shortest path calculation was efficient and accurate, leveraging the capabilities of Network