Assumptions

1. assume one segment corresponds to one transit line
    - the mapline in seg is a int
    - it may combine mutiple lines
2. create only one event at the departure node for the associated time interval
3. No walking links are considered
4. Board the first arrival
5. set daprture time = arrival time
6. double check the time order secquenc

Global Var

1. define a static class in the C#
2. EventCountID
    - a static variable for the event ID
3. EventArray []
    - add event sequentially
    - check the boudary of the event
4. create a temporal event
    - always compare with the temporal event
    - only if the temporal event is ok, then it is assocted with the event list

Parameters

1. intGranularity

inline function

- from travel time to travel cost

Classes

TripClass

Properties

1. Origin
2. Dest
3. TargetDepTime
4. TargetArrTime
5. DepMaxEarly
6. DepMaxLate
7. ArrMaxEarly
8. ArrMaxLate
9. ??? information to trace event tree
10. StartEvents
    - the set of starting events

Method

EventClass

Property

0. ID
1. FromEvent
    - refer to the from event ID
2. ToEvent
    - refer to the to event ID
3. isNodeEvent
    - if it is not node event then it is segment event
4. NodeId
    - if it is a node event, this associates with the node ID
5. SegId
    - if it is a segment event, this associates with segment ID
6. UpHeap
    - used in the scanned the heap and un scanned heap
7. DownHeap
    - used in the scaned heap and un scanned heap
8. EearlyNonDomEvent
    - earlier non dominated event
9. LateNoDomEvent
    - later nondomianted event
10. Time
    - tp
11. Cost
    - cp

Methods

SegClass

this is similar to the link type

Property

1. ID
2. Tail node
3. Head node
4. Travel time
5. MapLine (reversly we ahve MapSegs in the transit line clas)
    - map which transit line using this link/segment

Methods

NodeClass

Property

1. ID
2. InSeg
    - a set of in coming segments
3. OutSeg
    - a set of out going segments
4. EventHeapSet
    - a sorted list events associated with nodes
    - Non dominated event
5. InSegArrTime
    - the arrival time associated with the incoming segments
6. DepSegTime
    - the departure time associated with the outgoing segments
7. Type
    - Zone Node or stop name

Method

TransitLineClass

Property

1. Stops
    - set of stops
2. NumOfRun
    - number of run
3. DepTimes
    - Departure time associated with each run
4. Headway
5. SegTimes: time associate with each segment
6. Name

Method

Algorithms

ReadData

ReadLineData

```
    Add line element: schedule/frequency use different constructor
    Add line stops
    Get Num of Segment from the stops
    Add segment travel time
```

- read segment data from input file
- for small network arbitary set input data

CreateNodeFromSeg

- convert segment data to node data

EventPath

- find the nondominated least cost event path (tree)
- Step 1: Initialize nodes label
- Step 2: CurrentNode = origin
- Step 3: Create Initial events at origin node (start from one event)
- Step 4:

```
  start from the event at origin node
  find next activity cost
  if isNodeEvent
      next activity check all the outgoing links
  else
      next actvity check the following links and head node of the segment
  compare cost
  if the cost is non dominated
      create a new event
      insert into the heap
      set the property of the new event
      clean heap (remove the event that could reach the same element but
dominated)
  else
      do not create event
```

Functions in the EventPath

EnHeap

1. put element on the top the heap

DeHeap

1. delete element from the heap

EnDomSet

1. Insert event in the nondominated set based on the time

DeDomSet

1. Remove event from the dominated set

CreatNewEvent()

- Input
    1. from event
    2. Event ID (cont secutially)
    3. set event type
        - include is node event
        - node id, seg id
    4. set event time refer to the arrival time
    5. compute event cost refer to the value of time

CompareEvent

1. check whether the event is dominated or not
2. return

remove event

1. remove a event from a heap in the algorithm
2. remove a event from the maintained set

LpPathClass

Property

1. ID: id of the path number
2. StartEventID: Start event ID
3. NumOfWaits: number of waiting locations(including the start point)
4. MapWaitVar: Map waiting time at stops
5. MapDepVar: Map dep time at stop
6. MapArrVar: Map arr time at stop
7. MapTrainDep: number of schedule stops
8. MapBoardLine: at each node, which line the passengers board (this includes both fre and schedule)

Method

1. setLpPath

Lp

Property

Method

Implement Seat Constarints

Decision Varbiles

isSit

> indicate whether a passenger has a seat

1. Dimension: NumPath * NumLines
2. for each path a. trace number of boarding lines b. build the sequence

implementation steps

1. create data a. LineSec<int, list<>> * key is the line id * list is the secquence of stops
2. map data to decision variables a. create m_LineSec2Lamda: Dictionary<Dictionary<int, int>, int>
    - Dictionary: lineId, the stop node: the sequence of the seat variable

IsCon:

> indicate whether a link is congested or not consider frequency based service and schedule based service seperately

1. schedule based services Dictionary<Dictionary<int, int>, int> m_CongestionStatus2Var { get; set; } <<LineID, StopId>, position>
2. frequency based services <<LineID, StopId>, position>

1. isCongested

2. the congestion variable should be related to the variable a. vpathschedule cost b. vpathfre cost

3. neccessary varibles a. int skId = LpData.SchLineSet[l].Stops[k].ID; if (LpData.PathSet[p].VisitNodes.Contains(skId))

   b. for each path we know: v_PathLinkCap at each boarding node, we can trace the line id then, we can check this for all the path if either path the link path cap is zero

3. Public Dictionary<Dictionary<int, Dictionary<int, int>>, int> m_SeatLink2Var { get; set; } Dictionary<int, Dictionary<int, int> : <LineID, <startnode, endnode>> the value "int" is the count of used lines