Quota Travelling Salesman Problem with Passengers, Incomplete Ride and Collection Time

Bruno C. H. Silva and Islame F. C. Fernandes

Universidade Federal do Rio Grande do Norte, Campus Lagoa Nova, RN, Brazil
I.ppgsc@ppgsc.ufrn.br,
WWW home page:
https://sigaa.ufrn.br/sigaa/public/programa/portal.jsf?id=73

Abstract. The Quota Travelling Salesman Problem with Passengers, Incomplete Ride and Collection Time is a new version of the Quota Travelling Salesman Problem where the salesman uses a flexible ridesharing system to minimize his travel costs while visiting some vertices to satisfy a pre-established quota. Constraints regarding vehicle capacity, travel time, passengers limitations and penalty for passengers deliverance out of their destinations are considered. In this document, we present the conception method and the structure of the instances.

Keywords: quota travelling salesman problems, integer programming, experimental algorithms

1 Problem definition

The QTSP-PIC can be understood as a variant of the QTSP where the salesman can reduce his costs by sharing travel expenses with opportunity passengers. Passengers on board share the salesman's journey in order their trip easier. Each passenger can be transported directly to the desired destination or to intermediate points that aid in his journey in terms of safety or cost and time savings. The planning is carried out exclusively from the salesman's perspective, being considered only as mandatory transport conditions those of embarked passengers.

Let $G(N, E)$ be a connected graph, where N is the set of vertices and $E = \{(i, j) \mid i, j \in N\}$ is the set of edges. We denote by b_i the quota associated with vertex $i \in N$ and by g_i the time required to collect the vertex's quota. We denote by c_{ij} and by y_{ij}, respectively, the cost and the time required to travel by edge $(i, j) \in E$. Let L be the set of passengers. We denote by $L_i \subseteq L$ the subset of passengers who depart from $i \in N$. Let $org(l) \in N$ be the departing city of passenger l and $dst(l) \in N$ the city where passenger l would like to go in the end of his ride. However, out-of-destination delivery is allowed. The travelling salesman departs from $s \in N$, visits each city of a subset $N' \subseteq N$ only once and returns to s. The quota collected by the travelling salesman must be at least

K. Respecting the vehicle capacity R, the driver takes some passengers who are at cites of vehicle's route and chooses some city to deliver each of them, not necessarily their destination in fact. The ride's costs are shared with all vehicle's occupants. Each passenger $l \in L$ imposes a limit t_l about expense and a limit w_l about his ride time. Let h_{lj} be the penalty to deliver the passenger $l \in L$ at city $j \in N$, where $j \neq dst(l)$. That is, h_{lj} is added to the objective function whenever passenger j is delivered to city j. If $j = dst(l)$, thus $h_{lj} = 0$. Thus, the objective of QTSP-PIC problem is to find a Hamiltonian cycle on subset $N' \subseteq N$ such that the cost of ridesharing plus the eventual penalties are minimized and collected quota constraint is satisfied.

2 Instance generator procedure

Given the originality of the study, it was necessary to construct a set of test cases to validate the performance of the proposed solution algorithms. In order to facilitate the replicability of the experiments, a instance generator algorithm was implemented. This algorithm receives as input a number n of vertices, the capacity R of the vehicle, the instance class and a boolean indicating whether it will be symmetric or asymmetric. In order to minimize eventual biases in the set of test cases, three different classes of instances were constituted and denominated as A, B and C.

Class A, considered the easiest, has all randomly generated matrices and passenger configurations. Class B has the randomly generated cost matrix and all other components generated in a correlated and anti-correlated manner. Class C uses the same B mechanics, but out-of-destination delivery is stimulated through the use of a regulator that blurs out delivery penalties. This increases the number of feasible solutions to the problem, making the class C more difficult to solve.

The first step of the algorithm randomly generates elements c_{ij} of the cost matrix in a range of 100 to 999. The second step is to randomly generate elements y_{ij} of the time matrix in a range of 20 to 99 for instances of class A. If the creation of an instance of class B or C is in progress, it is calculated the average of all c_{ij}. For each c_{ij} greater than the average cost, the corresponding y_{ij} will be randomized between 40 and 99. If c_{ij} is equal or less than the average cost, then the corresponding y_{ij} will be randomized between 20 and 39. In this way time and cost of edges are anti-correlated.

In the third step, the quantity of passengers departing from each vertex is randomly picked according to one of three intervals:

- 0 to $R \cdot 3$, when $n \leq 20$
- R to $R \cdot 3^2$, when $20 < n \leq 100$
- R to $R \cdot 3^3$, when $n > 100$

In the fourth step the destination $dest(l)$, expenditure limit t_l, and time limit w_l of each passenger l are set. All $dest(l)$ is defined randomly. The t_l of each passenger is randomly given in the range of 15% and 30% of the cost matrix

minimum spanning tree. Each w_l is given by the Dijkstra's algorithm which receives the time matrix, $org(l)$ and $dest(l)$.

In the fifth step is calculated the penalty matrix h_{lj}. This operation follows three phases. In the first one, penalties $h0_{lj}$ are obtained through the Dijkstra's algorithm which receives the cost matrix, $org(l)$ and the destination j. In the second phase, a percentage z_{lj} is randomized between 90% to 150% if the instance is in class A or B. If the instance is in class C, this percentage falls to the range of 10% to 30%. In the last phase, the penalty matrix is calculated by formula $h_{lj} = h0_{lj}\dot{z}_{ij}$. We record that if $j = orig(l)$ or $j = dest(l)$, then $h_{lj} = 0$.

The quota b_i of each vertex is defined at sixth step. First, we calculate $z_i = \sum_{j=1}^{n} c_{ij}, \forall i \in N$, where N is the set of vertices of the current instance. Next, we compute $z_\alpha = \frac{\sum_{i=1}^{n} z_i}{n}$. If $z_i \leq z_\alpha$, then b_i will be randomized between 100 and 299. If $z_i > z_\alpha$, then b_i will be randomized between 300 to 700. In this way, the costly vertices will have the biggest quotas making the salesperson's choices more difficult.

Finally, in step 7, the collection time g_i associated to each quota b_i and the minimum quota K are defined. Each element g_i is randomized between 1 and 39 if $z_i \leq z_\alpha$. In the case of $z_i > z_\alpha$, g_i will be randomized between 40 and 79. After setting a g_i for each b_i, we calculate $K = \epsilon \cdot (\sum_{i=1}^{n} b_i)$, where ϵ is a regulate with value randomly picked between 0.4 and 0.6.

3 Instance file structure

Figure 1 - Instance file example

```
5 6 3                          /* The values of |N|, |L| and R */

0 397 530 452 120              /* In this matrix, every c_ij is set */
397 0 151 387 229
530 151 0 496 348
452 387 496 0 502
120 229 348 502 0

0 54 45 50 92                  /* In this matrix, every y_ij is set */
54 0 65 34 70
45 65 0 53 58
50 34 53 0 39
92 70 58 39 0

125.0 162.0 1 2                /* This line sequence comprehends */
178.0 162.0 2 1                /* the w_l, t_l, org(l) and dest(l) of */
107.0 135.0 2 3                /* each passenger l */
142.0 267.0 3 1
125.0 267.0 4 2
160.0 135.0 5 3

0 0 468 452 120                /* In this matrix, every h_lj is set */
0 0 468 452 120
433 0 0 452 120
0 397 0 452 468
397 0 468 0 229
468 397 0 452 0

922.35                         /* The value of minimum quota K*/
1 0.0 0.0                      /* A list of every b_i and the associated g_i */
2 290.0 7.0
3 586.0 20.0
4 451.0 9.0
5 350.0 5.0
```

In Figure 1, we present the structure of an instance with 5 vertices, 6 passengers and 3 seats available to board passengers. The bank comprises instances of 10 vertices onwards. Therefore, the instance presented here serves only as an example. The bank has 162 instances, divided between simetric and assimetric types and distributed equally in classes A, B and C.