

The flow chart of column-pool-based approximation

**1. Dynamic vehicle flow assignment based on Model 2**

Once the columns are generated for each vehicle group, the remaining task is to assign vehicles to satisfy passengers’ trip requests and network capacities. Assume that the total number of vehicles from each vehicle group is also unknown, then we can apply ADMM to convert the flow-based linear programming model as a quadratic programming model as follows.

Objective function:

(24)

where is the path flow of path , and are parameters. For simplicity, for each OD pair is replaced by by resorting all path numbers.

Its Hessian Matrix in can be derived as,

Since it is difficult to directly obtain its inverse matrix , especially in large-scale networks, we apply ADMM to decompose the primal problem to sequentially solve the subproblem for each column as

The subproblem for is a quadratic programming model which could be solved by the projected gradient method (Rosen, 1960) as follows:

(25)

where, and . In addition, projected gradient method also has been used in solving the path-based nonlinear programming models in equilibrium traffic assignment (Larsson and Patriksson, 1992; Jayakrishnan et al, 1994; Florian et al., 2009), and it is more efficient, compared with arc-based nonlinear programming models, but it needs more memory use.

At each iteration of ADMM, the Lagrangian multipliers are updated as follows,

Passenger group trip requests:

Arc capacity constraints:

As a discussion, it is possible to assign different vehicles within different blocks, and each block can be sequentially solved in ADMM and vehicles within a same block can find the best solution with parallel computing techniques.

We need to note that the solution from ADMM cannot always guarantee its feasibility. In order to find a feasible solution and upper bound value, we can sequentially load each column flow from ADMM. Path flows that exceed the required passenger trip requests or arc capacity will be removed during the sequential loading process. Finally, if some passenger requests cannot be satisfied, virtual vehicles will be used to find a feasible solution as the upper bound.

**2. Experiments with flow-based ADMM: Trip with pickup only request in the Chicago sketch test network**

**Stage A: A priori generation of column pool in Model 1**

The Chicago sketch network has 1320 nodes and 5431 links in Fig. 13. We assume that all passengers have the pick-up only trip requests as the first mile problem, which indicates that each passenger group with a number of passengers will have the same destination with a vehicle group. We treat them as one pair of vehicle group and passenger group. To generate the column pool, two scenarios are designed:

**Scenario 1:** 10 pairs of vehicle groups and passenger groups. In each pair, as a sample set, we assume that (i) 243 vehicles departs from different origins to one destination with different working time windows and (ii) 387 passengers submit trip requests with different pickup locations and time windows. The time horizon is 60 min (time unit). Since this is a simple set, the space-time arc capacity in each minute is assumed to be 5 vehicles. Vehicle carrying capacity is given as 1, so each vehicle aims to pick up one passenger from the origin depot to their same destination. It has 2430 binary variables and 332,160 constraints.

**Scenario 2:** 20 pair of vehicle groups and passenger groups. For each pair, similar to scenario 1, we also assume the same number of vehicles and passenger trip request but with different vehicle inputs (origin, destination, working time windows) and passenger inputs (pickup locations and time windows). It has 4860 binary variables and 338,460 constraints

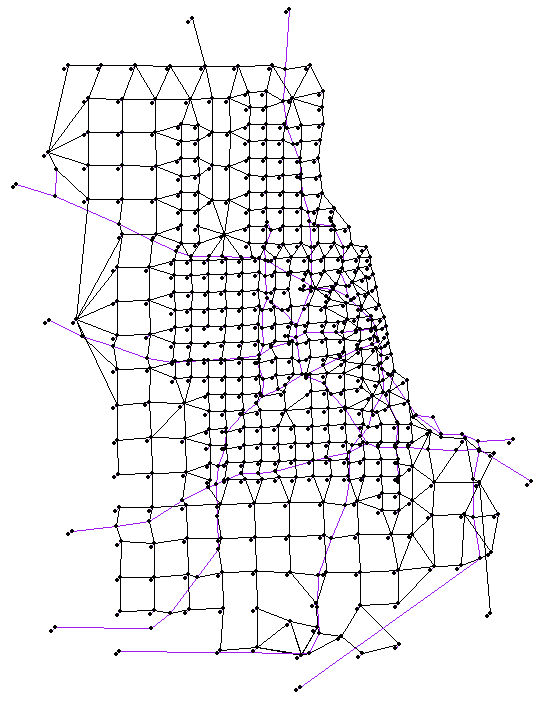


Fig. 13 Chicago sketch network for passenger pickup only

Then the vehicle-based ADMM in section 4.2.1 is used to find the vehicle-to-passenger and vehicle-to-arc assignment. Since the input data is randomly generated, some passengers may not be served and some vehicles may not serve any passengers. The final results are that (i) in scenario 1, 1789 vehicles find their paths/columns to serve 1084 passengers, and 23357 space-time arcs are generated based on vehicles’ paths, and (ii) in scenario 2, 3686 vehicles find their paths/columns to serve 2226 passengers, and 36454 space-time arcs are generated based on vehicles’ paths. The computation times for scenarios 1 and 2 are about 70 seconds and 140 seconds, respectively, from the laptop with 2.80GHz.

**Stage B: customized algorithm for flow-based ADMM**

Note that each passenger has a specific pickup location, time window and destination, and vehicle can only pick up passengers within a group with the same destination location. Therefore, we can build a column pool where each path of vehicles represents one column and each passenger represents one passenger group from Model 1. The question that arises is how many vehicles from different vehicle groups are required to satisfy those trip request from different passenger groups under tight road capacity constraints. Based on the last two scenarios, we design two experiments:

**Experiment 1**: there are 1084 passenger groups and each passenger group has 4 passenger trip requests. **Experiment 2**: there are 2226 passenger groups and each passenger group has 2 passenger trip requests. In this physical network, we assume that all space-time arc capacity in each minute is 35 vehicles, equal to 2100 vehicles per hour. To solve this problem, we try three approaches: (i) flow-based ADMM, (ii) upper bound generation by sequentially loading the column flow solution from ADMM, (iii) optimal solution from standard solver CPLEX in GAMS.

In **experiment 1**: three cases with different penalty parameters of for passenger trip constraints and for arc capacity constraints in ADMM are performed. Case 1: and ; Case 2: and ; Case 3: and . The results from ADMM by running 250 iterations and the optimal solution from CPLEX are shown in Fig. 14. The ADMM can converge to different objective values in three cases with different penalty parameters. Since capacity values and the number of capacity constraints are much higher than that of passenger trip requests, it is better to assign a smaller penalty value for in the objective function of ADMM.



Fig. 14 Solution of each iteration of ADMM in three cases and CPLEX in experiment 1

Then the upper bound generation algorithm is also implemented to find a feasible solution based on the results of ADMM. Fig. 15 shows the objective values of upper bound in three cases and the optimal solution. The Gap values of three cases compared with the optimal solution are 4.3%, 3.4% and 3.1%, respectively. It is observed that the three cases can finally reach good solutions with very small gap values. Further, the ADMM result of case 3 has the biggest gap value, but its upper bound solution can still have a small gap value. The possible reason is that the total path flow is a variable, so the upper bound generation can reduce the total path flow from ADMM to have a better feasible solution to satisfy those passenger trip demands and not violate the arc capacity constraints.



Fig. 15 Upper bound in three cases and the optimal value in CPLEX in Experiment 1

From the upper bound solution, 4737 space-time waiting arcs at 405 nodes have assigned vehicle flows, which indicates that the waiting happens at those nodes. By calculating the total waiting flow at those congested nodes during 60 mins, its heat map and the top10 of the most congested nodes are shown in Fig. 16(a) and (b), respectively. It can be observed that the destination areas of different passengers with pickup request only become congested, so it also raises one question about how to design the drop-off location in the future when a large number of passengers have the same destination with similar arrival time.



Fig. 16 Visualization of congested nodes in experiment 1

In order to compare the computation efficiency and memory use of the ADMM algorithm and CPLEX in GAMS (version 24.7.4), we also implement three other cases with different numbers of columns based on experiment 1. All tests are performed in the laptop with 8G memory and i5-4210U CPU @1.7GHz. The result is listed in Table 5. It can be observed that that CPLEX is much more efficient, but our customized algorithm can better utilize the memory. Specifically, our flow-based model is a linear programming (LP) model. How to efficiently solve LP model has been studied for almost 80 years, and most commercial solvers are powerful in solving LP models. Actually, most LP solvers apply some preconditioning techniques, such as, a simple geometric mean scaling in combination with equilibration, to reduce the condition number of the constraint matrix in order to decrease the computational efforts. As we know, CPLEX also has its “scaling routine” to preprocess LP models. In most of the literature in transportation domain, we haven’t found any papers which proposed an algorithm for LP and ultimately beat CPLEX in computation efficiency. Most comparisons happened when solving mixed integer programming, because CPLEX usually applied branch and cut to deal with those integer variables and the process is relatively time-consuming. In addition, it is still a research topic about how to improve the computation efficiency of ADMM, such as, by parallel computing (Boyd et al., 2010), which will be our future research. Also, we should emphasize that our ADMM is a general algorithm which can deal with nonlinear programming rather than just linear programming by CPLEX. Therefore, when the column cost is a nonlinear function of column flows, such as, for paths with reliability and variance (Xing and Zhou, 2011), our model could still be appliable and CPLEX will not deal with it. Another advantage of our column pool-based model is that a large number of columns has been stored in advance, so it will be beneficial for re-optimization and real-time optimization in the future and doesn’t need to read all the input data every time, which needs to be done in the case of CPLEX.

Table 5. Computation efficiency comparison between ADMM and CPLEX in GAMS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num of columns | Flow-based ADMM\_C++ (250 iterations) | | GAMS (solver: CPLEX) | |
| computation\_time | memory\_use | computation\_time | memory\_use |
| 1789 | 12s | 20m | 0.8s | 16m |
| 17890 | 81.9s | 43.4m | 3.2s | 77m |
| 89450 | 344s | 134.7m | 15.2s | 345m |
| 178900 | 514s | 264.1m | 31.5s | 686m |

Then the same procedure is also applied to **experiment 2**. The solution process and comparison among different cases are shown in Fig. 17 and Fig. 18, respectively. The Gap values of the three cases compared with the optimal solution are 3.9%, 2.9% and 2.5%, respectively. From the upper bound solution, 7173 space-time waiting arcs at 448 nodes have assigned vehicle flows. Its heat map and the top10 of the most congested nodes are shown in Fig. 19(a) and (b), respectively.



Fig. 17 Solution of each iteration of ADMM in three cases and CPLEX in experiment 2



Fig. 18 Upper bound in three cases and the optimal value in CPLEX in Experiment 2



Fig. 19 Visualization of congested nodes in two experiments