Project AMMM

Miquel Alcón & Roger Pujol

January 2019

1 Problem Statement

We have a Bus company that has a set S of services to do. We also have a set B of buses and a set D of drivers to do the services.

For each service $s \in S$ we have the starting time, the duration of the service in time, the distance of the route and the number of passengers.

For each bus $b \in B$ we have the capacity, the cost per time (€/min) and the cost per distance (€/km).

For each driver $d \in D$ we have maximum time that d can work, the time that d has of base salary (when this time is exceeded an extra cost will be applied), the cost per minute for base salary and the cost per minute for extra time salary.

Our objective is to find a bus $b \in B$ and a $d \in D$ for each $s \in S$ with the minimum possible cost (in €). We can only use limited amount of buses (limited by the variable "maxBuses"). Each $b \in B$ and each $d \in D$ can only do one service at the same time (no overlap in timing).

2 Integer Linear Model

In this section we will describe the Integer Linear Model that we used for the problem described at the previous section.

2.1 Sets and parameters

- **nServices**: the number of services.
- **nDrivers**: the number of drivers.
- **nBuses**: the number of buses.
- **maxBuses**: the maximum number of uses that can be used.
- S: the set of services.

- D: the set of drivers.
- S: the set of buses.
- **startingTime** [$s \in S$]: the set with the starting time of each service s (in minutes).
- **durationTime** [$s \in S$]: the set with the duration time of each service s (in minutes).
- **durationDist** [$s \in S$]: the set with the distance of each service s (in km).
- **passengers** [$s \in S$]: the set with the number of passengers of each service s.
- **capacity** [$b \in B$]: the set with the capacity of passengers of each bus b.
- **costTime** [$b \in B$]: the set with the cost per time of each bus b (in €/min).
- **costDist** [$b \in B$]: the set with the cost per distance of each bus b (in €/km).
- **maxTime** [$d \in D$]: the set with the maximum time that each driver d can work (in minutes).
- **costBM** [$d \in D$]: the set with the cost of each driver d with the salary during base minutes (in €/min).
- **costEM** [$d \in D$]: the set with the cost of each driver d with the salary during extra minutes (in €/min).
- **BM** [$d \in D$]: the set of base minutes of each driver d, when the driver exceeds this time working his cost change from costBM to costEM.

Also with this sets and parameters we precalculate at the preprocessing:

- **finalTime**[$s \in S$]: the set of the ending time of each service s (in minutes).

2.2 Model

2.2.1 Decision Variables

We used the next decision variables:

- **x_d** [$s \in S, d \in D$]: the boolean matrix with "1" if a driver d is assigned to the service s and with "0" otherwise.
- **x_b** [$s \in S, b \in B$]: the boolean matrix with "1" if a bus b is assigned to the service s and with "0" otherwise.
- **x_usedb** [$b \in B$]: the boolean set with "1" if the bus b is used in any service and "0" otherwise.
- **x_timed** [$d \in D$]: the integer set with the time worked by each driver d.

2.2.2 Objective Function

Our objective function is:
minimize:

$$(\sum_{s \in S} \sum_{b \in B} x\_b_{s,b} \times (durationTime_s \times costTime_b + durationDist_s \times costTime_b)) +$$

$$(\sum_{d \in D} (\mathbf{min}(x\_timed_d, BM_d) \times costBM_d + \mathbf{max}(x\_timed_d - BM_d, 0) \times costEM_d))$$

Basically the function is calculates the cost of the assignation. The first part calculate the cost of each assigned bus and then the second part calculates the cost of the worked hours of the drivers.

2.2.3 Constraints

- 1st constraint: a bus that has not enough capacity for the passengers of a service, can't be assigned to that service.

$$x\_b_{s,b} = 0 \qquad \forall s \in S, \forall b \in B \mid passengers_s > capacity_b$$

- 2nd constraint: for all the overlapping services can only have assigned the same bus or driver once or less.

$$x\_b_{s1,b} + \sum_{s2 \in S | overlap(s1,s2)} x\_b_{s2,b} \leq 1 \qquad \forall s1 \in S, \forall b \in B$$

$$x\_d_{s1,d} + \sum_{s2 \in S | overlap(s1,s2)} x\_d_{s2,b} \leq 1 \qquad \forall s1 \in S, \forall d \in D$$

  Where $overlap(s1, s2)$ computes if $s1$ overlaps in time with $s2$.

$$s2 > s1 \wedge$$
$$((startingTime_{s1} \leq startingTime_{s2} \wedge finalTime_{s1} \geq startingTime_{s2}) \vee$$
$$(startingTime_{s1} \leq finalTime_{s2} \wedge finalTime_{s1} \geq finalTime_{s2}) \vee$$
$$(startingTime_{s1} \geq startingTime_{s2} \wedge finalTime_{s1} \leq finalTime_{s2}))$$

- 3rd constraint: the drivers can't exceed his maximum hours.

$$x\_timed_d \leq maxTime_d \qquad \forall d \in D$$

- 4th constriant: at most $maxBuses$ can be used.

$$x\_b_{s,b} \leq x\_usedb_b \qquad \forall b \in B, \forall s \in S$$

$$\sum_{b \in B} x\_usedb_b \leq maxBuses$$

- 5th constraint: all the services must have one bus and one driver assigned.

$$\sum_{b \in B} x\_b_{s,b} = 1 \qquad \forall s \in S$$

$$\sum_{d \in D} x\_d_{s,d} = 1 \qquad \forall s \in S$$

- Auxiliary constraint: x_timed has to have the total time worked by each driver.

$$x\_timed_d = \sum_{s \in S} x\_d_{s,d} \times durationTime_s$$

3 Meta-heuristics

3.1 Greedy

Our greedy construction algorithm is:

$Solution \leftarrow \emptyset$
$S \leftarrow$ getServices()
$S_{sorted} \leftarrow$ sort$(S, passengers_s, DESC)$
for each s in S_{sorted} **do**
  $FA \leftarrow$ findFeasibleAssignments(s)
  $chosen \leftarrow$ **argmin**$\{$greedyFunction$(fa) \mid fa \in FA\}$
  if $chosen = \emptyset$ **then**
    makeInfeasible$(Solution)$
    break
  end if
  $Solution \leftarrow Solution \cup$ assign$(s, chosen_{bus}, chosen_{driver})$
end for
return $Solution$

Our greedy function (greedyFunction), where $fa = < s, b, d >$, has 2 parts.

- Bus cost $(q < s, b >)$:

  $q < s, b > = durationTime_s \times costTime_b + durationDist_s \times costTime_b$

- Driver cost $(q < s, d >)$:

  $q < s, d > = \mathbf{min}(x\_timed_d, BM_d) \times costBM_d + \mathbf{max}(x\_timed_d - BM_d, 0) \times costEM_d$

  Where x_timed_d is the worked time by d with all the assignments done to this point (including the one that the function is testing).

Greedy Function $(q < s, b, d >)$:

$$q < s, b, d > = q < s, b > + q < s, d >$$

This function basically calculates the cost of the current assignment.

3.2 Local Search

The main LocalSearch algorithm is:

```
bestSolution ← InputSolution
bestCost ← calculateCost(bestSolution)
newBestSolution ← true
while newBestSolution do
  newBestSolution ← false
  neighbor ← exploreNeighborhood(bestSolution)
  neighborCost ← calculateCost(neighbor)
  if neighborCost < bestCost then
    bestSolution ← neighbor
    bestCost ← neighborCost
    newBestSolution ← true
  end if
end while
return bestSolution
```

Going into more details, the **exploreNeighborhood** function has two different approaches: first and best improvement. It also follows a reassignment strategy (reassignment of a service from a bus and a driver to another ones). Next, you have the pseudo-code of this function:

```
bestSolution ← InputSolution
A ← getAssignments(bestSolution)
A_sorted ← sort(A, passengers_a, DESC)
for each < s, b, d > in A_sorted do
  for each b' in bestSolution_buses do
    for each d' in bestSolution_drivers do
      if b' ≠ b and d' ≠ d then
        Solution' ← bestSolution \ {< s, b, d >} ∪ {< s, b', d' >}
        if Solution' is feasible and Solution'_cost < bestSolution_cost then
          bestSolution ← Solution'
          if policy is FirstImprovement then
            break
          end if
        end if
      end if
    end for
  end for
end for
return bestSolution
```

3.3 GRASP

Our GRASP algorithm is:

$Solution \leftarrow \emptyset$
$S \leftarrow getServices()$
$S_{sorted} \leftarrow \text{sort}(S, passengers_s, DESC)$
for each s in S_{sorted} **do**
  $CL \leftarrow \text{findFeasibleAssignments}(s)$
  if $CL = \emptyset$ **then**
    return *INFEASIBLE*
  end if
  $totalCost_{min} \leftarrow \textbf{min}\{\text{greedyFunction}(c) \mid c \in CL\}$
  $totalCost_{max} \leftarrow \textbf{max}\{\text{greedyFunction}(c) \mid c \in CL\}$
  $RCL \leftarrow \{c \in CL \mid \text{greedyFunction}(c) \leq totalCost_{min} + (totalCost_{max} - totalCost_{min}) \times \alpha\}$
  $candidate \leftarrow \text{Select } c \in RCL \text{ at random}$
  $Solution \leftarrow Solution \cup \text{assign}(s, candidate_{bus}, candidate_{driver})$
end for
return *Solution*

As you can see above, the **RCL** function is:

$RCL \leftarrow \{c \in CL \mid \text{greedyFunction}(c) \leq totalCost_{min} + \alpha(totalCost_{max} - totalCost_{min})\}$

4 Instance Generator

We have done an instance generator in order to simplify and automatize the way to do large instances. This generator takes as input a "config" file with the values for:

- **instancesDirectory**: the directory where the instances will be saved.
- **fileNamePrefix**: the name of all the instances (will be followed by the number of instance).
- **fileNameExtension**: the file extension (in our case always will be "dat").
- **numInstances**: the number of instances that will be generated.
- **numServices**: the number of services of the instance.
- **numDrivers**: the number of drivers of the instance.
- **numBuses**: the number of buses of the instance.
- **maxBuses**: the maximum number of buses that will be used.
- **minStartTime & maxStartTime**: the minimum and maximum value for the starting time of each service (in minutes).

- **minDurationTime & maxDurationTime**: the minimum and maximum value for the duration time of each service (in minutes).
- **minDurationDistance & maxDurationDistance**: the minimum and maximum value for the distance of each service (in km).
- **minPassengers & maxPassengers**: the minimum and maximum number of passengers of each service.
- **minCapacity & maxCapacity**: the minimum and maximum capacity of passengers for each bus.
- **minCostTime & maxCostTime**: the minimum and maximum value of the cost per time (€/min) of each bus.
- **minCostDist & maxCostDist**: the minimum and maximum value of the cost per distance (€/km) of each bus.
- **minMaxTime & maxMaxTime**: the minimum and maximum value of the maximum time that each driver can work (in minutes).
- **minCostBM & maxCostBM**: the minimum and maximum value of the base cost per time (€/min) of each driver.
- **minCostEM & maxCostEM**: the minimum and the maximum value of the extra cost per time (€/min) of each driver.
- **minBM & maxBM**: the minimum and the maximum time of base salary of each driver (in minutes).

5 Comparison of the results

In order to compare the different implementations we have created five different configurations increasing sizes: small, medium, big, huge and giant (named "chungus" at the files). Each configuration will generate 5 instances. Then using the "IBM ILOG CPLEX Optimization Studio" we solved all the instances with no time limit and $GAP \leq 0.01\%$, with this results, we selected the ones that required more time to solve of each size (see table 1 for the results). We selected this ones because the time is more according to the size of the instance, whereas the fastest ones are just lucky executions of CPLEX that found the best solution without exploring most of the nodes.

	Small	Medium	Big	Huge	Giant
File	small_1	medium_0	big_1	huge_1	chungus_1
Constraints	1061	2356	4049	6177	8988
Variables	649	1393	2417	3721	5305
Time(s)	28.06	461.27	1572.88	2925.16	4133.00
Solution	397.25	524.22	805.15	889.61	1195.25

Table 1: The results of the slowest instances of each size when using CPLEX.

To test the heuristics, we first tried different values of α and policy (first and best improvement) for GRASP, with a medium instance for 5 min. We did this in order to find their optimal values. You can find the results in (see table 2). With this results we chose $\alpha = 0.1$ because it seems to be the most optimal value.

α	Solution with FI	Solution with BI
0.1	639.79	640.51
0.2	642.95	640.22
0.3	641.70	641.06
0.4	644.43	643.39
0.5	645.61	641.46
0.6	644.24	640.73
0.7	640.69	643.04
0.8	642.05	641.59
0.9	646.40	642.06

Table 2: Solutions for different values of α and policies for medium instance.

Having the optimal α we tried to solve the selected instances with:

- Greedy
- Greedy + LocalSearch (LS) with FirstImprovement (FI)
- Greedy + LocalSearch (LS) with BestImprovement (BI)
- GRASP + LocalSearch (LS) with FirstImprovement (FI)
- GRASP + LocalSearch (LS) with BestImprovement (BI)

The results of this executions are at the tables: 3, 4, 5, 6 and 7. Also there is a comparison of the Quality/Time of the solutions at the figures: 1, 2, 3, 4 and 5.

Method	GAP(%)	Time(s)	Solution
Greedy	21.98	0.0549	484.56
Greedy LS FI	21.98	0.4401	484.56
Greedy LS BI	21.98	0.4465	484.56
GRASP LS FI	20.47	41.394	478.57
GRASP LS BI	19.61	18.926	475.17
CPLEX	0	28.060	397.25

Table 3: Comparative of the different solving methods for the small instance. The maximum execution time for GRASP is 1 minute.

Method	GAP(%)	Time(s)	Solution
Greedy	26.58	0.2635	663.55
Greedy LS FI	26.58	1.2593	663.55
Greedy LS BI	26.58	1.2288	663.55
GRASP LS FI	25.17	96.918	656.18
GRASP LS BI	25.44	234.59	657.56
CPLEX	0	461.27	524.22

Table 4: Comparative of the different solving methods for the medium instance. The maximum execution time for GRASP is 5 minutes.

Method	GAP(%)	Time(s)	Solution
Greedy	14.51	0.6846	921.96
Greedy LS FI	14.51	3.3543	921.96
Greedy LS BI	14.51	3.5170	921.96
GRASP LS FI	13.80	40.161	916.33
GRASP LS BI	14.31	160.78	920.39
CPLEX	0	1572.8	805.15

Table 5: Comparative of the different solving methods for the big instance. The maximum execution time for GRASP is 10 minutes.

Method	GAP(%)	Time(s)	Solution
Greedy	23.14	1.8729	1095.43
Greedy LS FI	23.14	8.3487	1095.43
Greedy LS BI	23.14	8.3637	1095.43
GRASP LS FI	23.12	8.2651	1095.31
GRASP LS BI	23.19	8.2874	1095.94
CPLEX	0	2925.2	889.61

Table 6: Comparative of the different solving methods for the huge instance. The maximum execution time for GRASP is 20 minutes.

Method	GAP(%)	Time(s)	Solution
Greedy	17.22	3.1425	1401.07
Greedy LS FI	17.22	16.723	1401.07
Greedy LS BI	17.22	16.748	1401.07
GRASP LS FI	17.17	731.76	1400.42
GRASP LS BI	16.70	326.20	1394.85
CPLEX	0	4133.0	1195.25

Table 7: Comparative of the different solving methods for the giant instance. The maximum execution time for GRASP is 30 minutes.

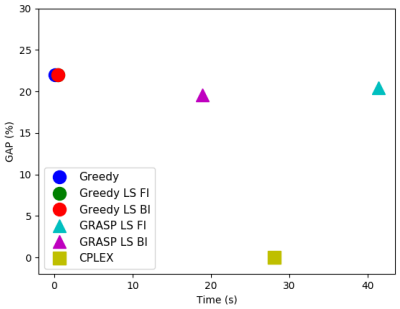

Figure 1: Small

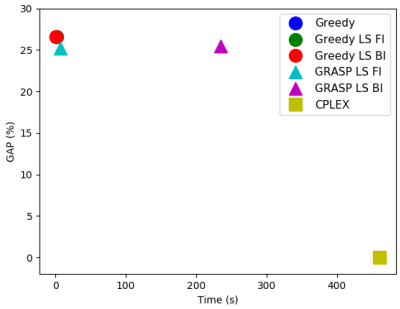

Figure 2: Medium

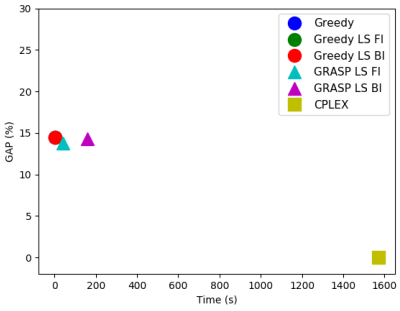

Figure 3: Big

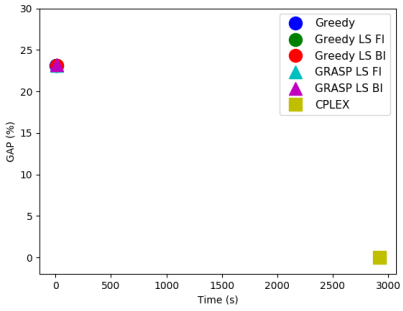

Figure 4: Huge

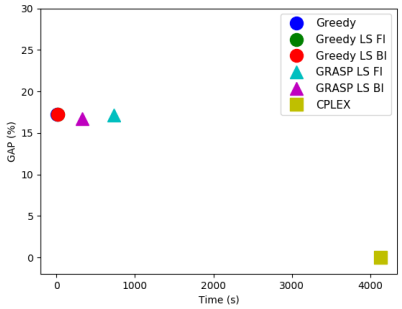

Figure 5: Giant

6 Conclusions

After the experiments, we have seen that in our case the CPLEX solution is much better than the ones obtained with the meta-heuristics (between 10 and 25 % of GAP better), but it's worth to be noted that heuristics give the solution in less time. The best results that we obtained with heuristics are using GRASP, which have little differences in the results for different sizes. In smaller instances, GRASP works better with LocalSearch + BestImprovement, since it can check all the possibilities easily. In bigger instances, GRASP works better with LocalSearch + FirstImprovement because this way will explore different results and won't get stuck. Finally, the simple Greedy returns almost always the same solutions that Greedy with LocalSearch, but in less time. This is probably because our Greedy construction falls most of the times at a local minimum and LocalSearch can't find a better solution next to it.