Algorithmic Methods for Mathematical Models
COURSE PROJECT

Eloy Gil Guerrero
eloy.gil@est.fib.upc.edu

1. Problem statement

Route scheduling is one of the most important tasks of logistic companies.
Given a set of locations l including a distinguished location l_s, every route starts at the same time from l_s and also end at l_s, maximum 12 hours later. All other locations should be visited once among all routes.
There is a known distance $dist_{l_1,l_2}$, in minutes, between each pair of locations l_1 and l_2. Also, except for l_s at each location the vehicle must perform a task during t_t minutes, that must start in the range of a $[min_l, max_l]$ time window, therefore vehicles may arrive too early and wait until the task can start. Having an unlimited number of vehicles, the goal is to find the minimum number needed to visit all locations and their routes. If there are two solutions with the same number the best will be the one in which the latest vehicle arrives at its final destination sooner.

2. OPL model[1]

2.1 Data structures

```
int nP = ...;
int S = nP+1;
int nT = nP;
int workTime = ...;
int bigM = workTime * 2;

range T = 1..nT;
range P_s = 1..nP;
range P = 1..S;
range window = 1..2;

int dist[p in P][q in P] = ...;
int task[p in P] = ...;
int task_window[p in P_s][w in window] = ...;

dvar int+ arrive_pt[p in P_s][t in T];
dvar int+ end_time[t in T];
dvar boolean pt[p in P][t in T];
dvar boolean from_p_to_q[t in T][p in P][q in P];
```

2.2 Objetive function

The S row of *pt* tells us the trucks which are in use. Multiplying it by *bigM* gives importance and if two solutions draw, it will break the tie adding the maximum arrival time.

```
minimize (sum(t in T) pt[S][t] * bigM) + (max(t in T) end_time[t]);
```

[1] Full model available in Project/Project.mod

2.3 Constraint definitions

2.3.1 Constraint set 1 (no time travelling allowed between two points)

The time difference between the arriving time at the 2nd point and the arriving time at the 1st is equal or bigger than the distance between the 1st and the 2nd point plus the time spent in the task, if such a track actually exists. Otherwise this comparison is avoided using the *bigM* value that is a way bigger negative value.

```
forall (p in P_s, q in P_s)
        forall (t in T)
                arrive_pt[q][t] - arrive_pt[p][t] >= (from_p_to_q[t][p][q] *
(dist[p][q] + task[p])) - (bigM * (1-from_p_to_q[t][p][q]));
```

2.3.2 Constraint set 2 (same as Constraint 1 but for the special cases of Source/Destination point)

```
forall (p in P_s, t in T) {
        arrive_pt[p][t] >= from_p_to_q[t,S,p] * dist[S][p];
        end_time[t] >= arrive_pt[p][t] + (from_p_to_q[t][p][S] * (dist[p][S] +
task[p]));
}
```

2.3.3 Constraint set 3 (task started during the specified time window)

```
forall (p in P_s, t in T) {
        arrive_pt[p][t] >= task_window[p][1] * pt[p][t];
        arrive_pt[p][t] <= task_window[p][2] * pt[p][t];
   }
```

2.3.4 Constraint set 4 (worktime behind maximum)

```
forall (t in T)
        end_time[t] <= workTime;
```

2.3.5 Constraint set 5 (each point is visited exactly once, except the Source/Destination point)

```
forall (p in P_s)
        sum(t in T) pt[p,t] == 1;
```

2.3.6 Constraint set 6 (defines the order of the visits)

If truck *t* visits any point it also visits the Source/Destination point, avoids to travel from a point to itself and If truck *t* visits a point *p* it must have a previous and next point.

```
forall (t in T, p in P) {
        pt[S][t] >= pt[p][t];
        from_p_to_q[t,p,p] == 0;
        sum(q in P) from_p_to_q[t,p,q] == pt[p][t];
        sum(q in P) from_p_to_q[t,q,p] == pt[p][t];
}
```

3. Metaheuristics

Due to the complexity of the optimization problem, heuristic algorithms are needed. I've implemented in Python a custom GRASP solver (with both Best and First Improvement) and also a BRKGA solver[2].

3.1 GRASP

3.1.1 Constructive phase pseudo-code

```
procedure construct(quality(·), alpha, problem)
       x = initialize empty solution with problem
       initialize candidate set (C)
       while C is not empty do:
              C = sort(C) by quality, asc (lower value is better)
              minC = worst quality value of candidates in C (last candidate)
              maxC = best quality value of candidates in C (first candidate)
              RCL = { s ∈ C | quality(s) <= minC + alpha * (maxC – minC) }
              select s randomly from the RCL
              add s to x
              update candidate set C
       end while
       return x
end construct
```

3.1.2 Local search pseudo-code

```
procedure localSearch(quality(·), N(·), x)
       initialize bestNeighbour with solution x
       H = { n ∈ N(x) | quality(n) < quality(bestNeighbour) } (lower value is better)
       while H is not empty do:
              q = +infinity
              foreach neighbour in H do:
                     if quality(neighbour) < q do:
                            q = quality(neighbour)
                            bestNeighbour = neighbour
                            if algorithm is GRASP-FI: break
              H = { n ∈ N(bestNeighbour) | quality(n) < quality(x) }
       end while
       return bestNeighbour
end localSearch
```

3.1.3 Greedy function description

[content goes here]

[2] All the source code of the solvers is available in the GRASP directory.

3.2 BRKGA

3.2.1 Chromosome structure

[content goes here]

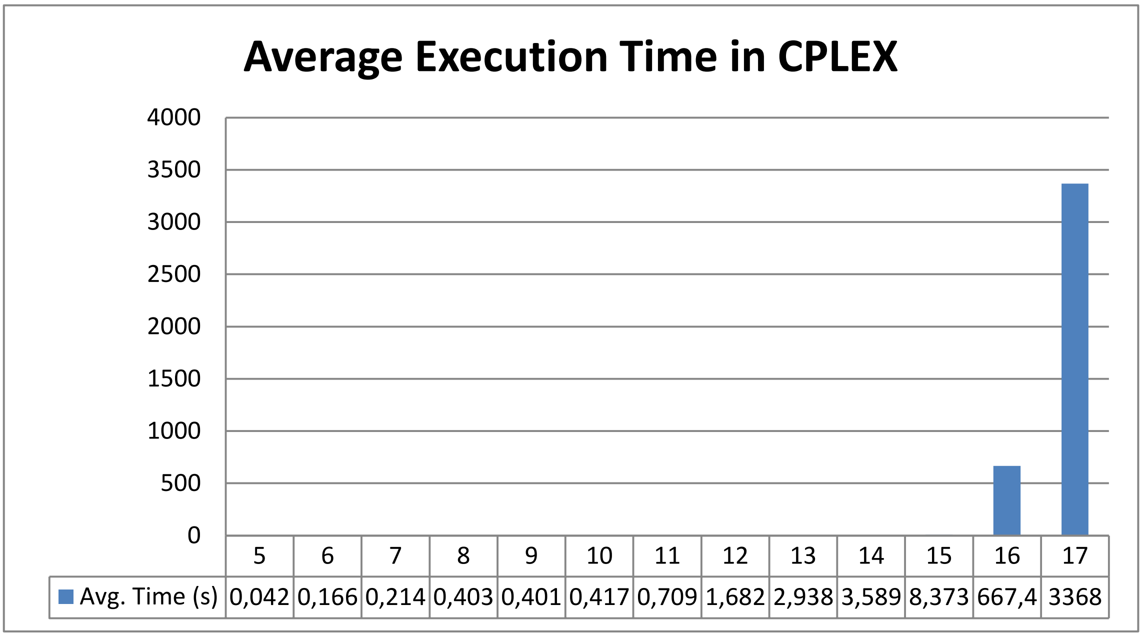
3.2.2 Decoder algorithm pseudo-code

[content goes here]

4. Performance comparison

4.1 CPLEX Execution Time evolution

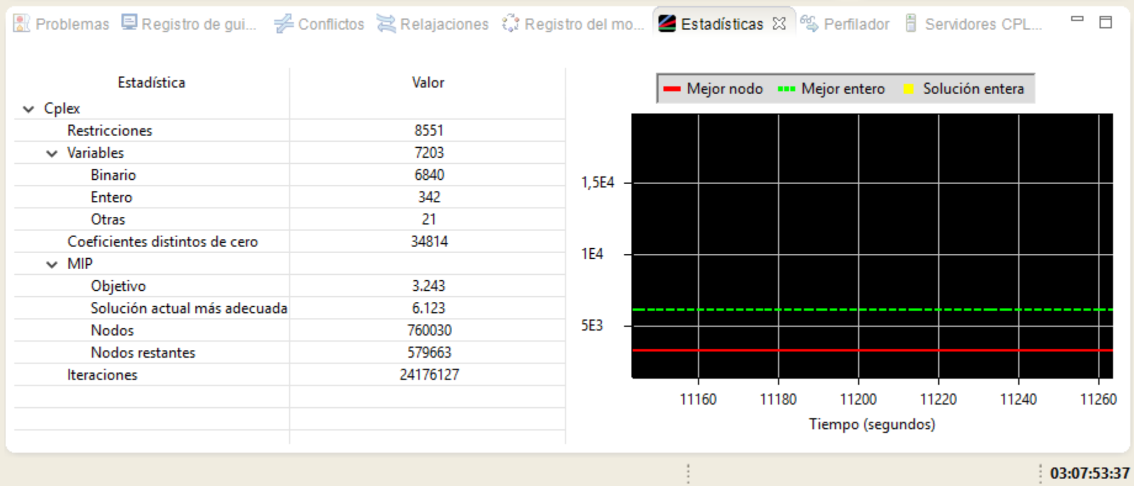
Using CPLEX we can obtain optimal solutions until the size of the instance makes the calculation impossible in an acceptable period of time. During the experimental phase, the biggest instance that is solved in less than 2 hours, in average, is instance 17, with 17 destination points.



	5	6	7	8	9	10	11	12	13	14	15	16	17
■ Avg. Time (s)	0,042	0,166	0,214	0,403	0,401	0,417	0,709	1,682	2,938	3,589	8,373	667,4	3368

Figure 1

As it is shown in figure 1, it is clear that the average execution time evolution follows an exponential distribution, which makes the execution of the instance 18 (it already has 19 points, including the starting one) unfeasible. After many hours of execution it is not even close, as it is shown in figure 2.



Estadística	Valor
Cplex	
Restricciones	8551
Variables	7203
Binario	6840
Entero	342
Otras	21
Coeficientes distintos de cero	34814
MIP	
Objetivo	3.243
Solución actual más adecuada	6.123
Nodos	760030
Nodos restantes	579663
Iteraciones	24176127

Figure 2

4.2 Objective function value comparison

4.2.1 CPLEX optimal vs GRASP-BI, FI & BRKGA in 30s executions

[content goes here]

4.2.2 CPLEX optimal vs GRASP-BI, FI & BRKGA in 300s executions

[content goes here]