# 3rd Assignment. Frank Wolfe algorithm

*David Cardoner Valbuena*

## Introduction

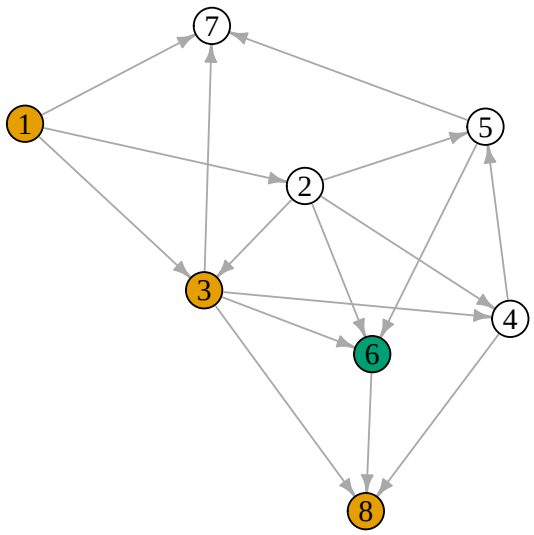
In this work we present a solution for an equilibrium traffic assignment problem. The algorithm is based on the concepts of simplicial decomposition, regularization and partial linearization. In this assignment, we consider an integrable linear delay cost function. For link $(i, j)$, the equation is: $s_{ij} = c_{ij} + d_{ij}x$, where $c_{ij}$,$d_{ij}$ are taken from the corresponding values $c, d$ in the data set.

We have an instance of 8 nodes, and consider the nodes 1-2 as origins and 6-7 as destinations. So we have the next o-d pairs:

```
(1,6)
(1,7)
(2,6)
(2,7)
```

## Graphical representation of the network

In the next graph, we represented the network associated with our instance. Green colors are destinations and orange colors origins:



## Idea of the algorithm

We formulate the problem in terms of paths (routes) between o-d pairs. The simplicial decomposition approach can be viewed as a column generation approach. Consider a subset of k paths , we solve a master problem (**MP**) and evaluate this solution to another problem. If optimal path flows are optimal in MP are also optimal in other problem. The idea is to evaluate the arc flows and the gradient of the objective function. By the Caratheodory theorem, any feasible point of bounded polygon can be expressed as a convex combination of its extreme points, and it's obtained by the MP.

The quadratic function to minimize is: $\sum c_{ij}\ x + \sum \frac{1}{2} d_{ij}\ x^2$ subject to supply the o-d pairs flow in the network.

## Introduction

The class of simplicial decomposition (SD) schemes have shown to provide efficient tools for nonlinear network flows. Shortest subproblems are solved in order to generate extreme points of the polyhedron of feasible flows, and, alternately, master problems are solved over the convex hull of the generated extreme points. We review the development of simplicial decomposition and the closely related column generation methods.

Steps of the algorithm:

1) Solve subproblem (gradient of the objective function): $c_{ij} + d_{ij}\ x$.

First we solve the subproblem with an initial value of $t0 = 1$ to generate a bad initial solution of the Sub-Problem. After that we increase the sets **Ws** and then solve the **Master Problem** (MP). With the values of the **MP** we are going to fix a new value to $t0$ based on the gradient of the **quadratic function** evaluated in **vv** obtained in the resolution of **MP** where **vvv** is the sets **Wx** multiplied by their associated $\alpha_i$, $i = 1..n$.

2) Update working sets: Add a new vertex and remove with information of $\alpha$ (MP variable) the smaller baricentric coordinate (small value of $\alpha$). When we say small is in terms of zero or close to zero.

When we arrive at a point where the dimensions of **Ws** is equal to $\rho$ we have to modify the columns of **Ws** based on the associated values of variable $\alpha_i$. To do that we will choose the smallest value of vector of $\alpha_i$ (associated with the baricentric coordinates of vertexs) and change this column with the value obtained in the new resolution of **SP**.

3) Update best lower bound (BLB) and calculate gap. When the **GAP** will be close to zero or the number of iterations arrives to 500 the algorithm will stop.

In step 3, we will update the **BLB** as: $max(BLB, f(x^v) + \nabla_x f(x^v)^T(\hat{x}^v - x^v))$ and gap as: $\frac{f(x^v - BLB)}{BLB}$. With this criterion we will ensure that our **BLB** and the value of the **f.obj** will converge and the gap will tend to zero.

**We stop iterating if** $gap < 0.005$ **or iteration number is equal to 500.**

4) Solve MP: Minimize quadratic function subject to actual working set, or in other words, solve the problem with a convex combination of extreme points obtained in the previous steps. *Go to step 1.*

At this point, we will use the set of combinations, called **W** (solutions obtained solving at each step the SP) to solve the MP and obtain the optimal solution of the algorithm or new points (combinations of origin destination flows) to use in the SP as a lineal combination of origin destination flows multiplied by $\alpha_i$, $i = 1..n$.

We start algorithm with **BLB** $= -\infty$. The goal is to observe how the **MP** and **BLB** will become closer at every iteration.

**Procedure information**

In the next table, we put information about the procedure at some iterations.

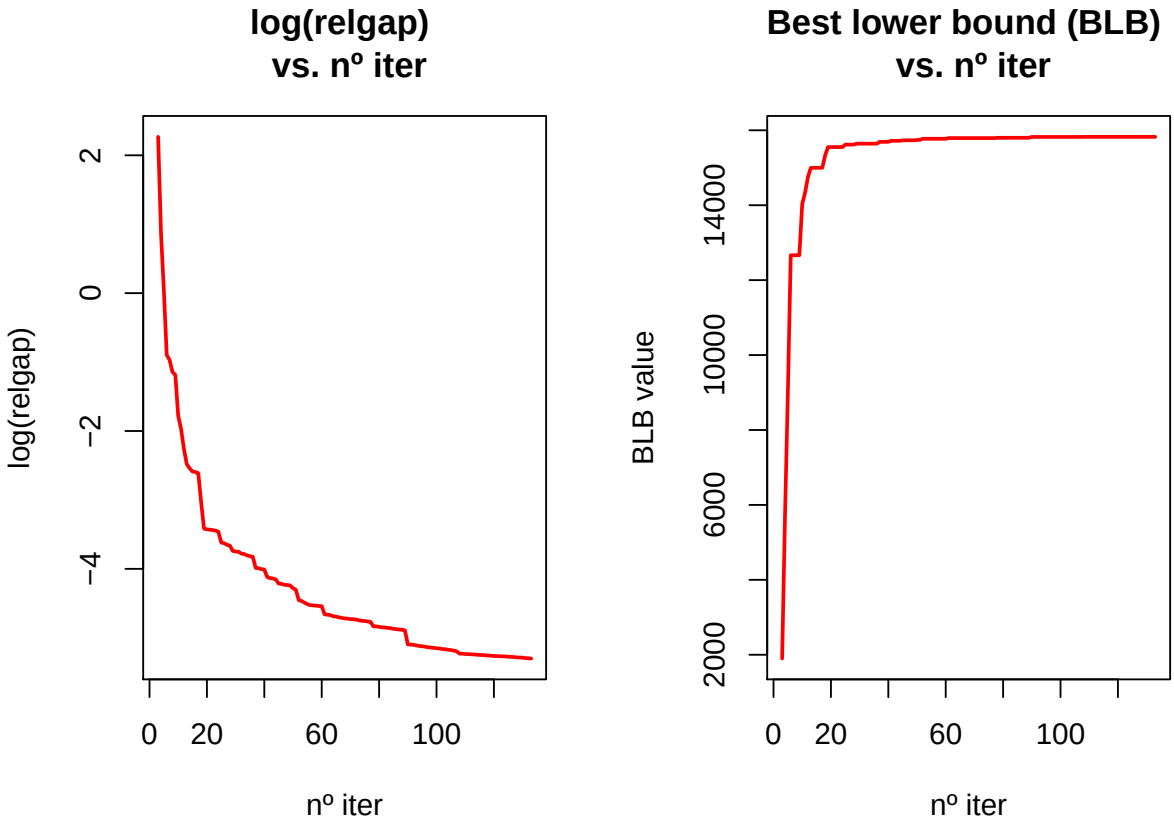
- iter = # iteration until $\rho = 2$, so we need to update extreme points with $\alpha_i$ variables.
- BLB = Best Lower Bound
- log_relgap = logarithm of relative gap value
- relgap = relative gap value
- Vnl = Quadratic objective function solution (associated to MP)
- alpha = Value of $\alpha_i$ variables to obtain information about baricentric coordinates.

To explore the results of the algorithm, a subset of the iterations is performed in the next table. The first 18 and the last 14 iterations is shown. With this table, we can see how our algorithm start with very bad results in terms of **GAP** and how in each iteration the value reduces. The scale logarithmic is also assumed at this point (like in the graphical representation).

| iter | BLB | log_relgap | relgap | Vnl | alpha |
|---|---|---|---|---|---|
| 3 | 1900 | 2.268 | 9.658 | 19364.4 | alpha 0 0.899372 alpha 1 0.0178041 alpha 2 0.0828237 |
| 4 | 5707.65 | 0.872 | 2.393 | 18392.2 | alpha 0 0.853083 alpha 1 0.142375 alpha 2 0.00454142 |
| 5 | 8979.52 | 0.047 | 1.048 | 17832.8 | alpha 0 0.873915 alpha 1 0.00720853 alpha 2 0.118876 |
| 6 | 12663.9 | -0.896 | 0.408 | 17470.7 | alpha 0 0.855696 alpha 1 0.14009 alpha 2 0.00421331 |
| 7 | 12663.9 | -0.969 | 0.380 | 16696.2 | alpha 0 0.585795 alpha 1 0.141014 alpha 2 0.273192 |
| 8 | 12663.9 | -1.144 | 0.318 | 16535.5 | alpha 0 0.870893 alpha 1 0.072205 alpha 2 0.0569024 |
| 9 | 12663.9 | -1.185 | 0.306 | 16402.3 | alpha 0 0.930366 alpha 1 0.0152459 alpha 2 0.0543882 |
| 10 | 14035 | -1.780 | 0.169 | 16330.7 | alpha 0 0.939439 alpha 1 0.060561 alpha 2 0 |
| 11 | 14336.1 | -1.972 | 0.139 | 16289.6 | alpha 0 0.930208 alpha 1 0.0286239 alpha 2 0.041168 |
| 12 | 14754.4 | -2.263 | 0.104 | 16251.4 | alpha 0 0.943345 alpha 1 0.0498166 alpha 2 0.00683847 |
| 13 | 14998.7 | -2.483 | 0.084 | 16178.1 | alpha 0 0.832221 alpha 1 0.0508738 alpha 2 0.116906 |
| 14 | 14998.7 | -2.543 | 0.079 | 16125.6 | alpha 0 0.846169 alpha 1 0.0638811 alpha 2 0.0899498 |
| 15 | 14998.7 | -2.588 | 0.075 | 16118.6 | alpha 0 0.989506 alpha 1 0.00712813 alpha 2 0.00336541 |
| 16 | 14998.7 | -2.595 | 0.075 | 16099.3 | alpha 0 0.97232 alpha 1 8.77807e-05 alpha 2 0.0275923 |
| 17 | 14998.7 | -2.612 | 0.073 | 16070.5 | alpha 0 0.954694 alpha 1 0.0390029 alpha 2 0.00630297 |
| 18 | 15334.8 | -3.037 | 0.048 | 16061.7 | alpha 0 0.976123 alpha 1 0 alpha 2 0.0238774 |
| 19 | 15551.6 | -3.417 | 0.033 | 16057.0 | alpha 0 0.974957 alpha 1 0.0185232 alpha 2 0.00652006 |
| 20 | 15551.6 | -3.427 | 0.032 | 16054.5 | alpha 0 0.990485 alpha 1 0.000834615 alpha 2 0.00868044 |
| 21 | 15551.6 | -3.432 | 0.032 | 16051.6 | alpha 0 0.988832 alpha 1 0.0111681 alpha 2 0 |
| 121 | 15825.1 | -5.265 | 0.005 | 15906.7 | alpha 0 0.995782 alpha 1 0.000812391 alpha 2 0.00340549 |
| 122 | 15825.1 | -5.268 | 0.005 | 15906.5 | alpha 0 0.996769 alpha 1 0.00210489 alpha 2 0.00112657 |
| 123 | 15825.1 | -5.269 | 0.005 | 15906.4 | alpha 0 0.997319 alpha 1 0 alpha 2 0.00268121 |
| 124 | 15825.1 | -5.271 | 0.005 | 15906.1 | alpha 0 0.98745 alpha 1 0.00812994 alpha 2 0.00442013 |
| 125 | 15825.1 | -5.275 | 0.005 | 15906.0 | alpha 0 0.998051 alpha 1 0.00110597 alpha 2 0.000842947 |
| 126 | 15825.1 | -5.276 | 0.005 | 15905.7 | alpha 0 0.994986 alpha 1 0.00185642 alpha 2 0.00315771 |
| 127 | 15825.1 | -5.279 | 0.005 | 15905.2 | alpha 0 0.992083 alpha 1 0.0058982 alpha 2 0.00201907 |
| 128 | 15825.1 | -5.286 | 0.005 | 15905.2 | alpha 0 0.999242 alpha 1 0.000145276 alpha 2 0.000612973 |
| 129 | 15825.1 | -5.286 | 0.005 | 15905.0 | alpha 0 0.996814 alpha 1 0.00287837 alpha 2 0.000307206 |
| 130 | 15825.1 | -5.288 | 0.005 | 15904.5 | alpha 0 0.98179 alpha 1 0.00574847 alpha 2 0.012462 |
| 131 | 15825.1 | -5.294 | 0.005 | 15904.2 | alpha 0 0.993682 alpha 1 0.00241463 alpha 2 0.00390351 |
| 132 | 15825.1 | -5.298 | 0.005 | 15904.0 | alpha 0 0.996888 alpha 1 0.00266113 alpha 2 0.000450473 |
| 133 | 15825.1 | -5.301 | 0.005 | 15903.8 | alpha 0 0.994964 alpha 1 0.00169485 alpha 2 0.00334075 |

### Graphical representation of procedure

In the nexts graphs are the detailed evolution of relative gap (in logarithmic terms) and the evoluton of **BLB**. After 134 iterations the value of the **relgap** reaches the minimum value. The values are plotted after the **rho** is full, the reason is that we want to show how the algorithm reduces the value of the objective function at each iteration.




## Direct solution

Let's start solving the problem directly. The objective function found is 15.865. With this value we will now the aproximate value of the solution of **RSD** algorithm (i say aprox. because our GAP is not zero). We display the values of the amount that is moved between the diferent **O-D** pairs and the value of the objective function returned by the algorithm. Is important to remember that in easy problems the algorithm solved by this method (let's call them the simplest method) could be obtained, while in long problems will be very expensive (in terms of computation) to know this value.

```
cat('
variable v
v :=
1 2   45.0643
1 3   41.7813
1 7   33.1544
2 3   14.8559
2 4    9.05642
2 5   60.3385
```

```
2 6   40.8135
3 4    0
3 6   44.161
3 7   12.4761
3 8    0
4 5    9.05642
4 8    0
5 6   15.0255
5 7   54.3694
6 8    0
;

variable v_k
v_k [*,*,1] (tr)
:      1         2          3          4         5      6    :=
2   45.0643     .          .          .         .      .
3   41.7813    0           .          .         .      .
4     .        9.05642    0          .         .      .
5     .       36.0079      .        9.05642    .      .
6     .        0         40          .         0      .
7   33.1544     .         1.78126    .        45.0643  .
8     .         .         0         0          .      0

 [*,*,2] (tr)
:   1      2         3       4      5       6    :=
2   0     .         .        .     .       .
3   0   14.8559     .        .     .       .
4   .    0         0         .     .       .
5   .   24.3306     .        0     .       .
6   .   40.8135    4.16101   .   15.0255    .
7   0     .       10.6949    .    9.30511   .
8   .     .        0         0     .       0
;

objective function
Vnl = 15865.7

Subject to:
flux_total :=
1 2    49.0643
1 3   128.344
1 7   168.772
2 3    79.2795
2 4    49.2821
2 5    61.3385
2 6   126.44
3 4     5
3 6    47.161
3 7    40.4284
3 8     3
4 5    12.0564
4 8     1
5 6    65.102
```

```
5 7    58.3694
6 8     2
;

sum of flow by origin
sum{(i,j) in links, k in origens} v_k[i,j,1] = 521.933

sum{(i,j) in links, k in origens} v_k[i,j,2] = 238.373

cost of flow by origin
v_k[i,j,1]*c[i,j] + v_k[i,j,1]^2/2*d[i,j] :=
1 2   1195.65
1 3   2743.85
1 7   2847.51
2 3      0
2 4    241.272
2 5    684.292
2 6      0
3 4      0
3 6    920
3 7     10.1031
3 8      0
4 5     68.1786
4 8      0
5 6      0
5 7   1195.65
6 8      0
;

v_k[i,j,2]*c[i,j] + v_k[i,j,2]^2/2*d[i,j] :=
1 2      0
1 3      0
1 7      0
2 3    626.024
2 4      0
2 5    320.32
2 6   2661.87
3 4      0
3 6     21.14
3 7    203.656
3 8      0
4 5      0
4 8      0
5 6    526.659
5 7     80.513
6 8      0
;

'
)
```

## Conclusions

The value of v obtained with the direct solution and the value obtained with the aplication of SD are quite similar but not equal. The reason is because the **relgap** is not exactly zero. This is the reason why some values of **v** and **vv** are not at all similar.

## Printout of the iterations

- We can look information about working sets (matrices Wx, Ws and W (which includes the other )) in document *annex.txt*. This matrices show the evoluton of the convex sets.
- The AMPL files for solve directly traffic problem are: *MinCM2_def.mod* and *MinCM222_def.run*.
- The AMPL files for solve traffic problem using procedure are: *entrega3_def.run* and *entrega3_def.mod*.