Multi-Commodity Flow Problem using Column Generation

Adrien Gausseran

June 2021

1 Linear Programming (LP)

Linear programming, also called linear optimisation, is, as the second name suggests, a method for solving optimisation problems. Since 1947 with the introduction of the *simplex algorithm* by Dantzig [Dan48] (the first practical approach to solving linear programs), linear programming has become one of the most widely used methods for solving optimisation problems. A linear program (LP) consists of three components:

- An **objective function**, which can be a cost to minimise or a profit to maximise. This is the main driver of decision making in optimisation problems.
- A set of **decision variables**. Each variable has a coefficient in the objective (in terms of cost or profit). The value of the objective is therefore dependent on the value of the set of decision variables. The value of the variables determine the output solution.
- A set of **linear constraints** which are in the form of equality and/or inequality. Each constraint restricts the value of one or more variables and thus reduces the set of possible solutions.

Linear programming is widely used to solve optimisation problems in operations research such as scheduling, flow routing, resource allocation, resource management, etc. In a LP, all variables must be fractional, however there are many problems that require integer or binary variables. For example, to model a delivery vehicle scheduling problem, it is impossible to use a fraction of a vehicle, when a truck is sent on delivery it is sent entirely. When all variables are binaries or integers the problem is an ILP (Integer Linear Program). When there is a mix of fractionals and integers/binaries variables the problem is a MILP (Mixed Integer Linear Program) but we will refer to it as ILP during this thesis to keep the nomenclature simple. Solving an ILP instead of an LP changes the complexity of the problem, as explained in subsection 1.2.

A very simple example allows to understand and visualise how it works. A company manufactures two products a and b. Two variables x_a and x_b represent

the output quantity for products a and b. To produce a and b three resources are needed: r_1, r_2, r_3. There are 6 units of resource r_1, 15 units of resource r_2 and 10 units of resource r_3 in stock. To prepare one unit of a, 1 unit of resource r_1, 3 units of resource r_2 and 1 unit of resource r_3 are needed. To prepare one unit of b, 1 unit of resource r_1, 1 unit of resource r_2 and 2 units of resource r_3 are needed. On sale, product a (resp. b) has a profit of \$12 (resp. \$10). The aim of the problem is to determine the number of products a and b to produce in order to maximise the profit with the given number of resources.

This problem can be modelled as follows:

$$\begin{array}{lrcll} \text{Maximise} & 12x_a + 10x_b & & & \\ \text{Subject To:} & x_a + x_b & \leqslant & 6 & (1) \\ & 3x_a + x_b & \leqslant & 15 & (2) \\ & x_a + 2x_b & \leqslant & 10 & (3) \\ & x_a & \geqslant & 0 & \\ & x_b & \geqslant & 0 & \end{array} \tag{1}$$

The constraint (1), (2) and (3) respectively represent the amount of resources r_1, r_2 and r_3 to make products a and b.

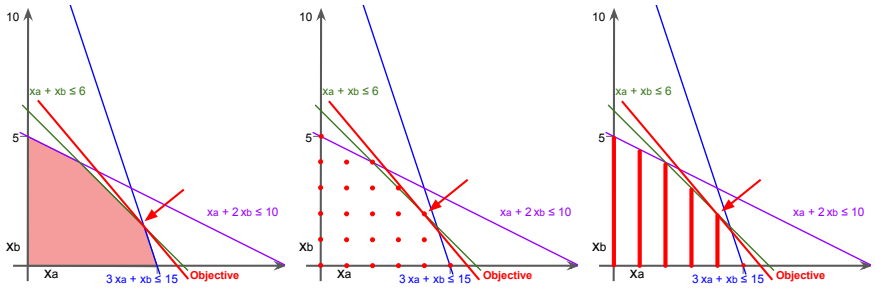
This problem can also be represented by a plot. Figures 1 represents different versions of the problem.

- First, in Figure 1(a), each axis represents a variable, x_a or x_b, and each line corresponds to a constraint (except the red one which corresponds to the objective). The colours of the lines (green, blue and purple) correspond to those of the model 1. The red polytope is the area of all possible solutions and is delimited by all the constraints. To find the optimal solution, the objective line is moved until it reaches an extreme border of the possible solutions area. In this example, the solution found is 69, with a value of 4.5 for x_a and 1.5 for x_b. The variables are therefore part of $\mathbb{R}^+$.

- Depending on the problem, the output values should be integer, and not fractional. Indeed, if for example x_a and x_b represent a number of phones to be sold, it's impossible to sell half a phone. In Figure 1(b), x_a and x_b are part of $\mathbb{N}$ and therefore the sum of each can only be part of $\mathbb{N}$. The area of possible solutions is thus made up of a set of points and the best solution must be chosen using the objective line. The program is no longer an LP but an ILP. The result is 68 with 4 for x_a and 2 for x_b.

- Finally, a third version, if x_a is part of $\mathbb{N}$ and x_bb is part of $\mathbb{R}^+$. As in Figure 1(c), the set of solutions can be represented by lines and the solution is found by the objective line. In this case, the program is a MILP.

Linear programs are written in the form:



(a) LP (b) ILP (c) MIP

Figure 1: Representation of the graphical solution of a linear optimisation problem

$$
\begin{aligned}
&\text{Minimise/Maximise} && \sum_{i=1}^{n} c_i x_i && && (1) \\
&\text{Subject To:} && \sum_{i=1}^{n} a_{j,i} x_i && \begin{cases} \leqslant \\ = \\ \geqslant \end{cases} b_j, \quad j \in [1, m] && (2) \\
& && x_i && \begin{cases} \leqslant \\ \geqslant \end{cases} 0, \quad i \in [1, n] && (3)
\end{aligned}
\tag{2}
$$

Where n is the number of variables and m the number of constraints. $X = \{x_1, x_2, ..., x_n\}$ is the set of decision variables. $C = \{c_1, c_2, ..., c_n\}$ is the set corresponding to the coefficients of the variables in the objective function, it may be a cost to minimise or a profit to maximise. For every linear constraint $j \in [1, m]$ there is a set of coefficient $a_{j,i}$ for every variables x_i and b_j is the *right-hand-side* of equation j.

In Equation 2, line (1) corresponds to the objective, line (2) corresponds to the set of constraints and finally line (3) corresponds to the restriction on variables: either unrestricted or negative or positive.

There is a **standard form** of writing LPs. The problem must be a maximisation and all constraints must be of the form lesser or equal. To switch to a minimisation problem the signs of the coefficients of the variables are changed and the constraints are switched to greater or equal.

Any linear program can be written in matrix form and in standard form:

$$
\begin{aligned}
&\text{Maximise} && z = c^T x \\
&\text{Subject To:} && Ax \leqslant b \\
& && x \geqslant 0
\end{aligned}
\tag{3}
$$

- $c = \begin{pmatrix} c_1 \\ ... \\ c_n \end{pmatrix}, x = \begin{pmatrix} x_1 \\ ... \\ x_n \end{pmatrix}$ are column vectors of size n.

Parameters	
$G = (V, E)$	The network where V represents the set of nodes and E the set of links.
C_{uv}	Capacity of a link $(u, v) \in E$ expressed as its total bandwidth available.
D	The set of all demands
(v_s^d, v_t^d, bw^d)	Each demand $d \in D$ is modeled by a triplet with v_s^d the source, v_t^d the destination, and bw^d the required units of bandwidth.
Variables	
x_{uv}^d	Utilisation of link $(u, v) \in E$ by the demand $d \in D$. $x_{uv}^d \in \{0, 1\}$ and its equal to 1 if the link (u, v) is used by d, 0 otherwise.

Table 1: Notation for the multi-commodity flow problem

- $b = \begin{pmatrix} b_1 \\ ... \\ b_m \end{pmatrix}$ is a column vector of size m.
- $A = \begin{pmatrix} a_{1,1} & ... & a_{1,n} \\ ... & ... & ... \\ a_{m,1} & ... & a_{m,n} \end{pmatrix}$ is a matrix of size $m \times n$.
- c^T represents the transpose of vector c.

1.1 Multi-Commodity Flow Modelling

We consider the ILP modelling of the Multi-Commodity Flow problem.

On a network modeled by a graph $G = (V, E)$, a set of requests D must be routed, each request $d \in D$ is a triplet (v_s^d, v_t^d, bw^d) composed of a source node v_s^d, a destination node v_t^d and a bandwidth demand bw_d. An allocated request uses one or more links between the source and the destination and its bandwidth is applied to the capacity of these links. Several objectives are possible, the one proposed here is to minimize the sum of the bandwidth used on the network.

This problem is simple and using an ILP is not the most efficient way to solve it but it is a simple way to understand the modelling of constraints. The list of parameters and variables is noted in table 1.

Objective: Minimize the sum of the bandwidth used

$$\min \sum_{(u,v) \in E} \sum_{d \in D} x_{uv}^d \cdot bw^d \tag{4}$$

Flow conservation constraints: Create a path from v_s to v_t. $\forall u \in V$, $\forall d \in D$.

$$\sum_{(u,v) \in \omega^+(u)} x_{uv}^d - \sum_{(v,u) \in \omega^-(u)} x_{vu}^d = \begin{cases} 1 \text{ if } u = v_s^d \\ -1 \text{ if } u = v_t^d \\ 0 \text{ else} \end{cases} \tag{5}$$

Link capacity constraints: $\forall (u, v) \in E$

$$\sum_{d \in D} bw^d \cdot x_{uv}^d \leq C_{uv} \tag{6}$$

This simple example provides a general understanding of modelling. 4 is the objective and minimises the total bandwidth used. 5 is a set of 3 constraints for each node and each demand, it forces one and only one link to leave if the node is v_s, one and only one link to reach the node if it's v_t, and for each intermediate node every incoming link implies an outgoing link. There is no need to constrain the fact that there is no cycle thanks to the minimization objective, If a flow uses a cycle it implies that there is a better allocation. Finally, 6 enforces the respect of the links' capacities. There is no non-negativity constraint because we specify in Table 1 that the x variables can only take the value 0 or 1.

1.2 Linear Programming properties

The *simplex algorithm* is the most common method to solve LPs. To summarise, the simplex method proceeds by going through the polytop (a compact convex set with a finite number of extreme points) from one vertex to another. At each step, it chooses the best vertex relative to the objective function. Either the algorithm will determine that the constraints are unsatisfiable, or it will determine that the objective function is unbounded, or finally it will reach a vertex from which it cannot progress, which optimises the objective function.

Other algorithms exist, without going into details we can mention the *ellipsoid method* proposed by Khachiyan [Kha80] who proves that an LP can be solved in polynomial time. But this algorithm was only effective in theory. Finally, in 1984, Karmarkar [Kar84] developed a new polynomial algorithm for practical use, based on the *interior-point method*. Thanks to these advances, LPs can be solved in polynomial time. While solving an ILP is NP-hard in general, it can be useful even in large problems. As explained in Section 2, some problems can be broken down into several smaller problems and solved quickly by an ILP. However, these solutions may not always be optimal. ILPs can also be relaxed by removing the integrality constraint from each variable. This method provides an upper-bound for maximisation and a lower-bound for minimisation, which is useful for approximation algorithms. Using the previous example, the LP scores 69 against 68 for the ILP. By using the LP solution and rounding the value of the variables, the optimal solution is found. This does not work in every case and the optimal solution may be far from the relaxation solution (there may be no solution to the ILP while there is one for the LP).

One of the key properties of LP used in the following section is the **Duality**. For each LP written in standard form, called the *Primal*, there is another LP called the *Dual*. The objective of the dual is the opposite of the primal one, i.e. if the primal is a minimisation problem, then the dual is a maximisation one, and vice-versa. For every variable in the primal there is a constraint in the

dual. And for every constraint in the dual there is a variable in the dual. The dual problem of the dual is the primal.

The dual problem (left) of the precedent primal 3 (right) is modelled as follows:

$$\begin{array}{lll|lll} \text{Minimise} & b^T y & & \text{Maximise} & c^T x & \\ \text{Subject To:} & A^T y \geqslant c & & \text{Subject To:} & Ax \leqslant b & (7) \\ & y \geqslant 0 & & & x \geqslant 0 & \end{array}$$

Where y is the vector variables of size m of the dual problem.

There are several rules for transforming the constraints and bounds of a primal maximisation problem to its dual:

- For an inferiority constraint and a non-negativity bound, the dual has a superiority constraint and a non-negativity bound.
- For an equality constraint and a non-negativity bound, the dual has a superiority constraint and no bound.
- For an inferiority constraint with no bound, the dual has an equality constraint and a non-negativity bound.

For example, the dual problem (left) of the precedent primal 1 (right) is modelled as follows:

$$\begin{array}{lll|lll} \text{Minimize} & 6y_a + 15y_b + 10y_c & & \text{Maximise} & 12x_a + 10x_b & \\ \text{Subject To:} & y_a + 3y_b + y_c \geqslant 12 & & \text{Subject To:} & x_a + x_b \leqslant 6 & \\ & y_a + y_b + 2y_c \geqslant 10 & & & 3x_a + x_b \leqslant 15 & (8) \\ & y_a \geqslant 0 & & & x_a + 2x_b \leqslant 10 & \\ & y_b \geqslant 0 & & & x_a \geqslant 0 & \\ & y_c \geqslant 0 & & & x_b \geqslant 0 & \end{array}$$

Two theorems emerge from the duality [Dan63]:

- **The weak duality theorem:** The objective value of any feasible solution of the dual is an upper bound on the optimal objective value of the primal solution. And reciprocally the objective value of any feasible solution of the primal is a lower bound on the optimal objective value of the dual solution. This implies that if the dual is unbounded, then the primal has no feasible solution, and reciprocally if the primal is unbounded, then the dual has no feasible solution.
  If x^* is a feasible solution of the primal maximisation and y^* is a feasible solution of the dual minimisation then the weak duality theorem can be stated as :
  $$\text{Maximise} \quad c^T x^* \quad \leqslant \quad \text{Minimise} \quad b^T y^*$$
- **The strong duality theorem:** If the primal problem has an optimal solution with a finite objective value then so does the dual (and vice

versa). This objective value is the optimal value for both problems. The bounds given by the weak duality theorem are tigh. If x^* is an optimal solution of the primal maximisation, there exists y^* an optimal solution of the dual minimisation and:

$$\text{Maximise} \quad c^T x^* \quad = \quad \text{Minimise} \quad b^T y^*$$

The comprehension of the duality in LP is important to understand the functioning of the column Generation (CG).

2 Column Generation

ILPs can be an interesting method for solving small problems but they quickly become inefficient when the size of the problem increases. Some large problems can still benefit from ILP modelling under certain conditions.

Column Generation (CG) [DDS05] is a decomposition method coming from the field of operational research and which divides an optimisation model into two parts:

- A **Restricted Master Problem** (RMP) which consists of the original problem but with a restricted number of variables and consequently of possible solutions.
- A set of **Pricing Problems** (PP) which consists of a smaller problem by focusing on one (or a subset) of the RMP variables, thus allowing a faster execution. Each pricing will create columns (decision variables) to feed the RMP. Using an ILP is not mandatory for modelling PPs. Any exact optimisation algorithm will suffice as long as the objective can be modified. Using an approximation algorithm or a heuristic can improve the execution time of the PPs but can potentially reduce the quality of the RMP solution.

In order to use column generation, a problem needs to be divided into several pricing problems.

To recall, an LP uses a matrix in which the rows are the constraints and the columns are the decision variables. The CG works by gradually adding columns (decision variables) to find a good solution.

For example Three PPs, red, blue and green are the sub-problems of a RMP. At the start of the algorithm the constraint matrix of the RMP is:

$$A = \begin{pmatrix} a_{1,(1,1)} & a_{1,(2,1)} & a_{1,(3,1)} \\ \dots & \dots & \dots \\ a_{m,(1,1)} & a_{m,(2,1)} & a_{m,(3,1)} \end{pmatrix}$$

Where $a_{x,(y,z)}$ represents the coefficients of the column z of the pricing y in the x^{th} constraint. After each pricing has produced a valid column, the matrix

evolves.

$$A = \begin{pmatrix} a_{1,(1,1)} & a_{1,(1,2)} & a_{1,(2,1)} & a_{1,(2,2)} & a_{1,(3,1)} & a_{1,(3,2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m,(1,1)} & a_{m,(1,2)} & a_{m,(2,1)} & a_{m,(2,2)} & a_{m,(3,1)} & a_{m,(3,2)} \end{pmatrix}$$

Figure **??** is a representation of how CG works. In order to be solved, CG needs first a valid solution (even a very bad one). Usually the algorithm starts with the smallest possible number of variables (down to one per pricing problem). This first set of variables may be found either through using a heuristic or by using the PPs a first time if it works. The quality of this first solution can have an impact on the quality of the final solution found by the RMP. Variables will then be added at each iteration. In order to add variables that can potentially improve the solution of the master, the relaxation of the master problem is solved. The use of an LP allows, thanks to the simplex algorithm, to obtain the dual values of the constraints. These dual values indicate for each row if it constrains the solution of the problem. The dual values are then injected in the pricing problems. For each pricing problem, when a column related to it is present in a constraint, the dual of this constraint influences the pricing objective.

The objective $\overline{C_j}$ of the PP for the variable j is called the reduced cost and is expressed as follows:

$$\overline{C_j} = C_j - \sum_i a_{ij} \cdot \mu_i \tag{9}$$

Where C_j corresponds to the original cost of the variable j in the RMP objective. The sum on i is the sum of all constraints i where the variable j appears. a_{ij} is the coefficient of the variable j in the constraint i. μ_i is the dual value of the constraint i.

Each PP is then solved. If the reduced cost of a PP is less than 0, the resulting column can potentially improve the solution of the RMP relaxation, so it is added to the set of columns of the RMP. The algorithm then iterates between executing the RMP relaxation and executing the PPs. When no more columns are created the integral RMP is executed. This execution gives an optimal solution for the restricted set of columns of the RMP. The end of the iteration can also be triggered using a timer or when the improvement of the RMP relaxation objective is lower than a parameterised threshold. Finally, even if the RMP has a restricted number of variables, in some problems this number may still be so large that it cannot be executed in a reasonable time. In this case, a column filtering method can be added to remove the least useful columns. As an example, the least used columns in relation to their number of iterations in the RMP may be removed. However, removing columns may decrease the quality of the RMP solution, as variables not used when running the LP may become useful when running the ILP.

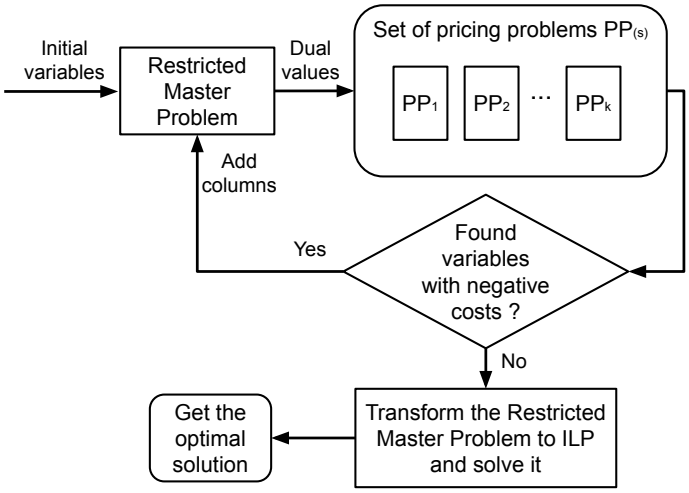


Figure 2: How column generation works

The solution given by the RMP may be close to or far from the actual optimal solution, depending on the problem. But, the CG at least gives a solution, while it is not possible for an ILP due to time execution constraints. The quality of the CG solution (objective = π^*_{CG}) can be estimated by comparing it to the optimal solution of the relaxed problem (objective = π^*_{LP}). By calculating $(\pi^*_{CG} - \pi^*_{LP})/\pi^*_{LP}$ we obtain a ratio. If this ratio is 0, then the solution is optimal, and the closer it is to 0, the better the solution.

The **advantage** of column generation is its good optimisation acceleration capabilities. In the last few years, the number of cores per CPU has increased drastically. Since the PPs are independent, they can be solved in parallel and take advantage of this technological advance.

The **drawbacks** of column generation are as follows. Finding a first set of variables to start the algorithm may be difficult the quality of this first set may have an impact on the quality of the solution. In some problems, the speed up may be negligible or the execution time may be longer than ILP. Pricing problems can take too long to solve if the problem they address is too complex. Finally, the implementation of a column generation model is more complex than an ILP one.

2.1 Multi-Commodity Flow

To understand how CG works we take the previous example of the shortest path and generalise it. This problem is called the Multi-commodity flow problem, it is extremely similar to the previous one except that this time we have several requests that must be satisfied on the same network. The objective will change slightly, instead of simply minimising the number of links used, we will minimise the bandwidth (which implies that between 2 requests with a different

Parameters	
D	The set of demand d
P_d	The set of path p from demand $d \in D$
δ_{uv}^p	Is equal to 1 if the link (u, v) is in path p, 0 otherwise
Variables	
z_p	Utilisation of path p, its equal to 1 if path p is used, 0 otherwise.

Table 2: Additional notation for the RMP

throughput it is better to favour the biggest one). The variables and parameters are almost the same, each triplet identifying the needs of a request (source, destination, bandwidth) is now linked to a demand d. The same is true for the x variables which are linked to a demand d. This ILP can therefore be used to solve our new problem, but as we said before using an ILP (especially if the number of requests is very large) can quickly become ineffective. To solve this problem with a large number of requests we therefore use the CG technique.

To understand the design we will first look at Figure 2. Instead of having an ILP finding the solution for all requests we have a set of sub-problems (PPs) that will each find a possible path for a request. These paths are then given to the RMP which instead of having as many variables as there are links and requests, will only have a reduced number of paths. The RMP is then relaxed (with fractional variables) and the dual values of the active constraints on the path variables are collected. These dual variables are passed to the PPs to update their objective and they are executed as well. If a PP finds a path with a negative cost it implies that this path can potentially improve the solution of the RMP and so it (this column) is added to the RMP. We iterate until no PP finds a path with a negative cost. We then solve the RMP with integer variables and it gives us the optimal solution for the paths it has. This solution may be close to or far from the actual optimal solution, depending on the problem, but the use of the CG may allow us to have a good solution where an ILP would not have been able to give us one due to time constraints.

The quality of the CG solution (objective = π_{CG}^*) can be estimated by comparing it to the optimal solution of the relaxed problem (objective = π_{LP}^*). By calculating $(\pi_{CG}^* - \pi_{LP}^*)/\pi_{LP}^*$ we obtain a ratio which the closer it is to 0 the better is the solution found. If the ratio is 0 the solution is optimal.

Note that to start the algorithm we need a first set of paths that give us a valid solution (even a very bad one), either found through a heuristic or by using the PPs a first time if it works. The quality of this first solution can have an impact on the quality of the final solution found by the RMP.

Table 2 adds notations used in the RMP.

Restricted Master Problem

Objective: minimize the bandwith used

$$\min \quad \sum_{d \in D} \sum_{p \in P_d} \sum_{(u,v) \in E} z_p \cdot \delta_{uv}^p \cdot bw_d \tag{10}$$

One path used per demand. For each Demand $d \in D$.

$$\sum_{p \in P_d} z_p = 1 \tag{11}$$

Link capacity constraints. For each Link $(u, v) \in E$.

$$\sum_{d \in D} \sum_{p \in P_d} z_d \cdot bw_d \cdot \delta_{uv}^p \leq C_{uv} \tag{12}$$

As we can see the objective 10 is a little different, we want to minimize the bandwidth used and for that we minimize the bandwidth for the path used by each demand. There are no longer any flow conservation constraints which are managed by the PPs. Constraint 11 forces the use of one and only one path per request. And constraint 12 prohibits the overloading of links.

For the **Pricing Problems**, the model will be the same as for the shortest path except that the objective must take into account the dual values of 11 and 12 constraints, represented by μ. The objective $\overline{C_j}$ of the PP for the variable j is called the reduced cost and is expressed as follows:

$$\overline{C_j} = C_j - \sum_i a_{ij} \cdot \mu_i \tag{13}$$

Where C_j corresponds to the original cost of the objective function.
The sum on i is the sum of all constraints i where the variable j appears.
a_{ij} is the coefficient of the variable j in the constraint i.
μ_i is the dual value of the constraint i.
In this problem the original cost C_j of a column j is the bandwidth used by the demand :

$$\sum_{(u,v) \in E} x_{uv} \cdot bw_d$$

Before writing the objective function, it is necessary to verify the form of the constraints in the RMP. Indeed, equation 13 is valid when the RMP is written in standard form, so for a minimization problem all constraints must be in the form *geq* or the dual of the constraints will have negative values.
Link capacity constraints in standard form. For each Link $(u, v) \in E$.

$$\sum_{d \in D} \sum_{p \in P_d} -bw_d \cdot \delta_{uv}^p \geq -C_{uv} \tag{14}$$

The new *objective* is therefore:

$$\min \quad -\mu_d^{11} + \sum_{(u,v) \in E} x_{uv} \cdot bw_d \cdot (\mu_{uv}^{14} + 1) \tag{15}$$

As can be seen for constraints 11, there is one constraint per demand d, so in the PP objective the dual value taken into account is therefore the one associated with the PP demand, resulting in the notation μ_d^{11}. On the contrary for constraints 14, there is one constraint per link (u, v), which means that the dual value will be used by all PPs resulting in the notation μ_{uv}^{14}. We can also see that μ_{uv}^{14} is positive because in 14 the variable δ_{uv}^{p} has a negative coefficient.

Using an ILP is not mandatory for modelling PPs. Any exact optimisation algorithm will do as long as the objective can be modified.

For a more complete understanding of column generation, the reader may refer to [DDS05].

References

[Dan48] George B Dantzig. Programming in a linear structure. *Washington, DC*, 1948.

[Dan63] George Bernard Dantzig. *Linear Programming and Extensions*. RAND Corporation, Santa Monica, CA, 1963.

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column Generation*. Number 978-0-387-25486-9 in Springer Books. Springer, May 2005.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, Dec 1984.

[Kha80] L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.