Multi-Commodity Flow Problem using Column Generation

Adrien Gausseran

June 2021

1 Linear Programming (LP)

Linear programming, also called linear optimization, is, as the second name suggests, a method for solving optimization 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 consists of 3 components:

- An **objective function**, which can be a cost that we want to minimise or a profit that we want to maximise. This is the main driver of decision making.
- A set of **decision variables**. These variables determine the output solution by having a coefficient in the objective function.
- 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.

The majority of linear programs are written in the form:

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

Where $X = x_1, x_2, ..., x_n$ is the set of variables. $C = c_1, c_2, ..., c_n$ is the set corresponding to the coefficient of the variables in the objective function, it may be a cost we want to minimize or a profit we want to maximize. For every constraint $j \in [1, m]$ we have a set of coefficient $a_{j,i}$ for every variables x_i and

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

we have also b_j know as the *right-hand-side* of equation j. In equation 1, line (1) correspond to the objective, line (2) correspond to the set of constraints and finally line (3) correspond to restriction on variables : some can be unrestricted and some can be negative or non-negative.

There is a **standard form** of writing LPs. The problem must be a maximization and all constraints must be of the form $\leqslant$. To switch to a minimization problem we can easily change the signs of the coefficients of the variables and switch the constraints to $\geqslant$.

Linear programming is widely used to solve optimisation problems in operations research such as scheduling, flow routing, resource allocation, resource management, etc.

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

$$\min \sum_{(u,v)\in E} \sum_{d \in D} x_{uv}^d \cdot bw^d \tag{2}$$

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

$$\sum_{(u,v)\in\omega^+(u)} x^d_{uv} - \sum_{(v,u)\in\omega^-(u)} x^d_{vu} = \begin{cases} 1 \text{ if } u = v^d_s \\ -1 \text{ if } u = v^d_t \\ 0 \text{ else} \end{cases} \tag{3}$$

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

$$\sum_{d\in D} bw^d \cdot x^d_{uv} \leq C_{uv} \tag{4}$$

This simple example provides a general understanding of modelling. 2 is the objective and minimises the total bandwidth used. 3 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, 4 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 still the most used method to solve LPs. To summarise, the simplex method proceeds by going through the polytop 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.

However other algorithms exist, without going into details we can mention the *ellipsoid method* proposed by Khachiyan [Kha80] who proves that the LP can be solved in polynomial time. But his 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 we know that LPs can be solved in polynomial time. But this is only the case when all variables are fractional. Modelling a problem with an ILP therefore does not allow it to be solved in polynomial time, even if a polynomial algorithm exists. While solving an ILP is NP-Hard in general, it can be useful even in large problems, as we will see.

ILPs can also be relaxed by removing the integrality constraint from each variable. This method provides an optimistic bound (upper-bound for maximisation and lower-bound for minimisation) useful for approximation algorithms. Using our example we see that the LP scores 69 against 68 for the ILP, so it gives us an upper bound. By using the LP solution and rounding the value of the variables we can see that in our example we manage to find the optimal solution, however this only works in rare cases and the optimal solution is often far from the relaxation solution.

One of the key property of LP is the **Duality**. For each LP written in standard form, called the *Primal* : there is another LP called the *Dual* which is built following a precise scheme. Every constraints in the primal become variables in the dual. Every variables in the primal become constraints in the dual. And finally the objective function is inversed. The dual problem of the dual is thus the primal.
Two theorems emerge from the duality:

- **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.
- **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) and this objective value is the optimal value for both problems. the bounds given by the weak duality theorem are tigh.

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 ineffective when the problem size increases. Some large problems can still benefit from ILP modelling under certain conditions.

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

- A **Restricted Master Problem** (RMP) which consists of the original version of the problem but with a reduced number of variables and consequently of possible solutions (hence the restricted).
- A set of **Pricing Problems** (PP) which consists of a smaller version of the original problem by focusing on one or a subset of the RMP variables, thus allowing faster execution.

In order to use column generation a problem needs to be able to be divided into several sub-problems.

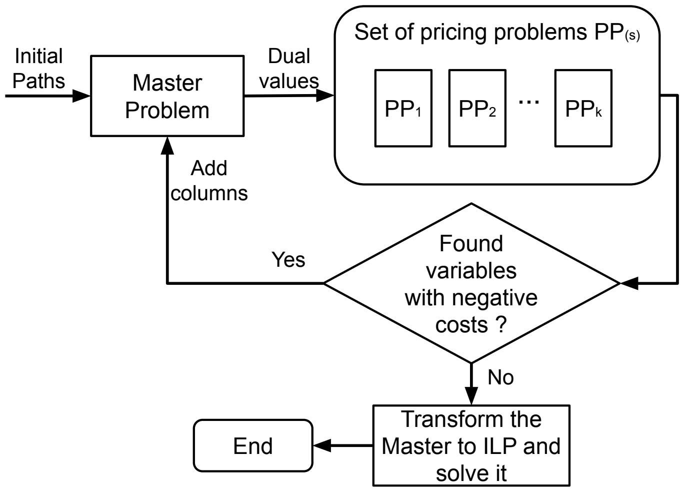
Initial Paths
Master Problem
Dual values
Set of pricing problems PP(s)
PP1
PP2
...
PPk
Add columns
Yes
Found variables with negative costs ?
No
Transform the Master to ILP and solve it
End

Figure 1: How column generation works

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 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 1. 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

Parameters	
D	The set of demand d
P_d	The set of path p from demand $d \in D$
δ^p_{uv}	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

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 $= \pi^*_{CG}$) can be estimated by comparing it to the optimal solution of the relaxed problem (objective $= \pi^*_{LP}$). 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 A adds notations used in the RMP.

Restricted Master Problem

Objective: minimize the bandwith used

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{(u,v) \in E} z_p \cdot \delta^p_{uv} \cdot bw_d \tag{5}$$

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

$$\sum_{p \in P_d} z_p = 1 \tag{6}$$

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

$$\sum_{d \in D} \sum_{p \in P_d} z_d \cdot bw_d \cdot \delta^p_{uv} \leq C_{uv} \tag{7}$$

As we can see the objective 5 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 6 forces the use of one and only one path per request. And constraint 7 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 6 and 7 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:

$$\overline{C_j} = C_j - \sum_i a_{ij} \cdot \mu_i \tag{8}$$

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 8 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 $(u, v) \in E$.

$$\sum_{d\in D} \sum_{p\in P_d} -bw_d \cdot \delta^p_{uv} \geq -C_{uv} \tag{9}$$

The new *objective* is therefore:

$$\min \quad -\mu^6_d + \sum_{(u,v)\in E} x_{uv} \cdot bw_d \cdot (\mu^9_{uv} + 1) \tag{10}$$

As can be seen for constraints 6, 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^6_d. On the contrary for constraints 9, 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^9_{uv}. We can also see that μ^9_{uv} is positive because in 9 the variable δ^p_{uv} 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.

The **advantage** of column generation is its good optimisation acceleration capabilities. To take advantage of the technological advances of the last few years with the increase in the number of cores per CPU, it can be noted that each PP can be solved independently of the others, thus allowing a parallelization potential.

The generation of columns has three **disadvantages**. The final solution is dependent on the initial variables and a bad first set of columns can have a negative 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, and it may need to be implemented to be noticed. Finally, the implementation of a column generation model is more complex than an ILP one.

References

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

[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.