



Constraint Games for stable and optimal allocation of demands in SDN

Anthony Palmieri, Arnaud Lallouet, Luc Pons

► To cite this version:

Anthony Palmieri, Arnaud Lallouet, Luc Pons. Constraint Games for stable and optimal allocation of demands in SDN. 2019. hal-02114831

HAL Id: hal-02114831

<https://hal.science/hal-02114831>

Preprint submitted on 29 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Constraint Games for stable and optimal allocation of demands in SDN

Anthony Palmieri^{1,2} · Arnaud Lallouet^{1,2} ·
Luc Pons¹

Received: Monday 29th April, 2019

Abstract Software Defined Networking (or SDN) allows to apply a centralized control over a network of commuters in order to provide better global performances. One of the problem to solve is the multicommodity flow routing where a set of demands have to be routed at minimum cost. In contrast with other versions of this problem, we consider here problems with congestion that change the cost of a link according to the capacity used. We propose here to study centralized routing with Constraint Programming and Column Generation approaches. Furthermore, selfish routing is studied through with Constraint Games. Selfish routing is important for the perceived quality of the solution since no user is able to improve his cost by changing only his own path. We present real and synthetic benchmarks that show a good scalability.

Keywords Constraint Programming · Game Theory · Optimization · Integer Linear Programming · Column Generation · SDN · multicommodity Flow

1 Introduction

With the internet of things, all kinds of devices are going to communicate, from washing machines, lightbulbs to autonomous cars. By 2020, the forecasts estimate the number of connected devices to the internet is growing to over 31 billion [34]. The amount of data transfer increases with the rise in the number of connected devices. Recently, Software Defined Networking (or SDN) is replacing traditional network routing because it allows fast and remote network reconfiguration, which enables a plethora of flexible architectures, like the upcoming network slicing [52]. SDN (see Figure 1) allows centralized control over a network of commuters in order to increase the overall performance. A full SDN controller is a nice source for many optimization problems [25] including online ones. Due to this dynamic aspect and the increasing size of the controlled networks, it is very likely that decentralized algorithms will be mandatory to provide both the expected quality of service and short time response.

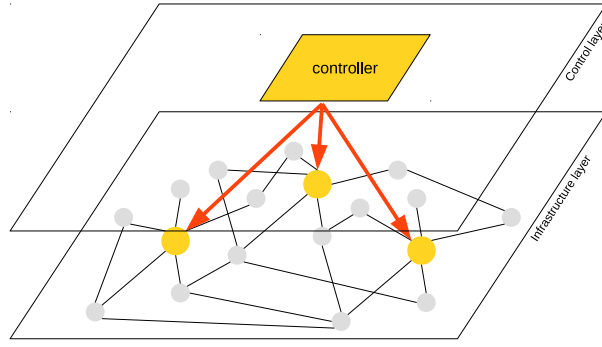


Fig. 1 Software Defined Networking

In this paper, we consider the independent routing of multiple demands across a network, also called *the multicommodity flow routing problem*. Each demand requires to be routed from a source to a destination in a network with limited capacity (i.e. each link has a limited capacity). The overall goal is to assign a route to each demand that minimizes the global cost of routing. This problem and other variants such as robust SDN networks have been studied for a long time [14, 7] with some computational approaches including linear programming [2]. A survey can be found in [28]. Interesting theoretical results have been found, like the one which states that when the problem has a sufficient size and capacity, all flows are actually routed along single-paths [40]. This justifies the modern interest in unsplittable routing of demands. The demands are routed in a network with limited capacity constraining the shortest paths computations, which is known as a NP-complete problem. In our approach, we do not consider other side constraints such as must-pass/cannot-pass or redundant routing, although they can be easily introduced in our constraint model. However, we consider a congestion model increasing the cost of a link according to the traffic routed.

Furthermore, SDN is based on a centralized vision of networking but this does not mean that all algorithms have to be centralized [32]. Indeed, with the growth of the size of the controlled zone and the large increase in the volume of the demands, decentralized algorithms will be necessary to achieve the expected level of performance for future SDN with millions of demands coming online. A common way of modeling agreement between a set of agents is to reach a Nash equilibrium. Also, some instances of the problem correspond to networks of aggregated traffic for which the users (often network providers) are very sensitive to the quality of service. This is why an allocation at Nash equilibrium is desirable as it ensures the user that his quality of service cannot be improved by any selfish move. While a centralized approach is sure to converge to an optimal solution, it is not guaranteed for Nash equilibriums. The equilibriums costs can be far from the global optimum. Braess's paradox [5] is a good illustration. It states that in congested roads network, building a new route creates even more congestion due to the selfishness of agents. This degenerative behavior is one of the motivations to compute Price of Anarchy [15] which allows to evaluate the potential loss of efficiency of decentralized algorithms (i.e. the loss of being at Nash equilibrium).

We propose two approaches for the centralized approach: a Constraint Programming (or CP) model and an Integer Linear Programming (or ILP) model using Column Generation. Only the Constraint Programming model is able to model closely the congestion problem and can be used to solve the problem to optimally. For this, we use a natural and dedicated heuristic based on increasing paths and a relaxation based on shortest path to prune efficiently the search space. Note that increasing path have been introduced as CP heuristics in [36].

The Constraint Programming model is implemented using the Choco constraint solver [39] and the ILP one with CPLEX [13]. The model computing the Nash equilibriums uses Constraint Games framework: ConGa which is an extension of the Choco solver for Constraint Games [35]. In the benchmarks, we show networks with hundreds or even thousands of commodities solved to optimality including the Nash equilibriums computations. These results show that practical use of game theory is now possible at industrial scale.

The paper is organizing as follow: first we introduce the problem in Section 2, then in Section 3 and 4 we present the CP model and the heuristics used to compute a solution in practice. In section 5 we present the ILP model, in Section 6 the Constraint Games framework used to compute selfish routing, the Section 7 gives a litterature review, in Section 8 the evaluation on a set of benchmarks on real-world and synthetic instances and lastly we present the conclusion.

2 Multicommodity path routing in SDN

2.1 Problem statement

A multicommodity path routing problem (MCPRP) consists of a graph defining a network and a set of commodities (flow demands) to be routed on this graph. We consider in this article the problem in which we compute for each demand a single route from the source to the destination node such that the sum of bandwidth routed by a link does not exceed its capacity. Congestion occurs when a link is taken and is reflected by a congestion cost which helps to ensure a homogeneous distribution of the routes. The overall objective is to minimize the sum of costs of the routed demands.

We assume we have a network $N = (V, E)$, which is a directed graph composed of a set of vertices (or nodes) V and a set of edges (or links) $E \subseteq V^2$. For each edge $e = (x, y) \in E$, we associate a cost $cost(e) \in \mathbb{R}^+$ and a capacity $cap(e) \in \mathbb{R}^+$. Let D be the set of demands to be routed. For a demand $d \in D$, we define $src(d) \in V$ and $dst(d) \in V$ to be respectively the source and destination node, and $bw(d) \in \mathbb{R}^+$ to be the required bandwidth for this demand.

A *path* is a sequence of nodes $p = (v_i)_{i \in [0..n]}$ such that $\forall i \in 0..n-1, (v_i, v_{i+1}) \in E$. We denote by $src(p)$ the node v_0 and by $dst(p)$ the node v_n . We consider here only acyclic paths, i.e. such that $i \neq j \rightarrow v_i \neq v_j$. By a slight abuse of notation, we write $(x, y) \in p$ to denote that the arc (x, y) is taken in the path p .

A *solution* for the MCPRP is the assignment of a path $path(d)$ to each demand d such that we ensure correctness:

$$\forall d \in D, \quad src(path(d)) = src(d)$$

$$\forall d \in D, \quad dst(path(d)) = dst(d)$$

and admissibility with respect to the capacity constraints:

$$\forall e \in E, \quad \left(\sum_{\{d \in D \mid e \in path(d)\}} bw(d) \right) \leq cap(e)$$

2.2 Congestion model

In order to ensure a good balance over the network, we incorporate to the model a model of congestion. Basically, congestion will increase the cost of a link when this link is close to saturation. For this, we define the load of an edge e to be:

$$load(e) = \left(\sum_{\{d \in D \mid e \in path(d)\}} bw(d) \right) / cap(e) \quad (1)$$

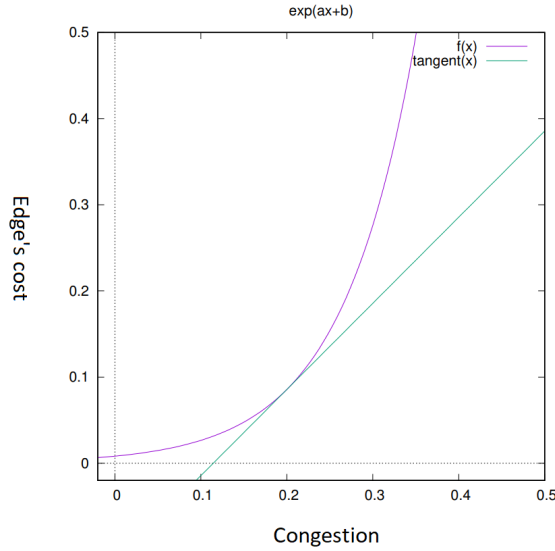


Fig. 2 A plot of the congestion function for $MaxC = 1000$ and $cong'(0.2) = 1$

The congestion model we use for a given arc e has an exponential increase of the form:

$$cong(e) = \exp(a \times load(e) + b) \quad (2)$$

In order to choose the parameters a and b , we pose some conditions on the function. First we should have a sufficiently high value of $cong(e)$ when the load is 1. By sufficiently high we mean that a demand should not prefer to take a heavily congested link while there are some (maybe longer) available paths. It can be done by fixing this limit to the highest link cost of the network $MaxC$. We then have the equation $e^{a+b} = MaxC$. Then, in order to set when the exponential starts to overtake on a linear increase, we impose a condition on the derivative to be 1 at a given point α . The derivative of the congestion function is given by $cong'(x) = ae^{ax+b}$. If we impose that the derivative should be 1 for $x = \alpha$, we get the equation $ae^{a\alpha+b} = 1$. By solving numerically these equations we get the values of a and b for a given problem. For example, in Figure 2 is a plot of the congestion function for $MaxC = 1000$ and $cong'(0.2) = 1$. We assume that the same values of a and b are set for all the links of the network, although this can be easily changed.

2.3 Optimization

Solving a MCPRP P to optimality means finding a solution minimizing the global cost of the demands. For this, we first define the cost to route a demand. It is obtained by aggregating the cost of each traversed arc with the cost coming from congestion:

$$cost(d) = bw(d) \times \sum_{e \in path(d)} (cost(e) + cong(e)) \quad (3)$$

Then the cost of the whole problem is given by:

$$cost(P) = \sum_{d \in D} cost(d) \quad (4)$$

Note that this function is strictly monotonic, resulting in that each addition of demand increases the edge cost.

3 Constraint model

In order to implement this problem as a constraint program, we need to first represent paths, which will be the solutions of our problem. Then we need to link the computed paths to the network data: costs, capacity and provide a support to compute congestion.

3.1 Path modelling

A path is represented by an array $path$ of $|V|$ variables which correspond to the set of arcs in the path. Each variable's value corresponds to the node's successor (i.e. the next node along the path). The initial domain of a variable associated with a node v is given by the set of neighbors of v in the graph. In order to ensure the correct representation of a path, we use the global constraint $subPath(path, src, dst)$ which ensures

that the node from *src* to *dst* form a valid subpath of the graph. This constraint is a variant of *subCircuit*. Unused nodes of the path point to themselves and an extra variable is appended to the array to indicate which vertex starts the path.

Example 1 (Path model)

The figure 3 describes the model for finding a path having the node 2 as a source and the node 5 as the destination. In the beginning, the variables domains are filled with all the possible neighbors including itself. For example, the node 0 can have as successor the nodes: 0, 1 or 2. Only the source and the destination are treated differently. Since a path can be seen as a circuit between the source and the destination nodes. That is why their domains are adapted: no self-loop for the source node (i.e. a successor is required) and the destination's successor is the source. A solution to the problem instance is depicted in the array line labeled by *path*. This array encodes the path (2, 1, 4, 3, 5). It has to be read as follow: the node 0 has 0 as successor (not in the path), the successor's of 1 is 4, the successor's node of 2 is 1 ...

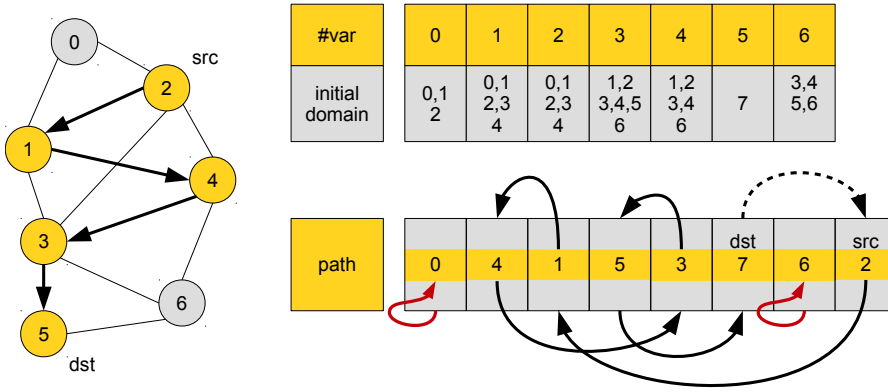


Fig. 3 Encoding of a path

For each demand $d \in D$ we associate an array $path^d = (v_i^d)_{i \in V}$ constrained by:

$$subPath([v_1^d, \dots, v_n^d], src(d), dst(d))$$

3.2 Graph model

In order to ensure that no link is overloaded and in order to compute congestion, we need to know which demands are routed by a given arc. In this model, we use a Boolean variable $EdgeIsUsed_{(i,j)}^d$ which is true if the path $[v_1^d, \dots, v_n^d]$ assigned to demand d uses the arc (i, j) . This connection is made with the following channeling constraints:

$$\forall (i, j) \in E, \forall d \in D, \quad EdgeIsUsed_{(i,j)}^d \leftrightarrow v_i^d = j$$

We compute the amount of bandwidth routed by an arc in a variable $f(e)$ with the constraint:

$$\forall e \in E, f(e) = \sum_{d \in D} EdgeIsUsed_e^d \times bw(d)$$

We ensure that the capacity of each arc is not exceeded:

$$\forall e \in E, f(e) \leq cap(e)$$

Then we can compute the congestion of a given edge in a variable $cong(e)$:

$$\forall e \in E, cong(e) = e^{a \times \frac{f(e)}{cap(e)} + b}$$

The cost $cost(d)$ of routing a demand by a given path is given by the constraint:

$$\forall d \in D, cost(d) = \sum_{e \in E} EdgeIsUsed_e^d \times bw(d) \times (cost(e) + cong(e)) \quad (5)$$

A variable *ProblemCost* sums the costs to route all demands:

$$ProblemCost = \sum_{d \in D} cost(d) \quad (6)$$

We shall minimize this variable.

This model is quite standard and intuitive. It defines one Boolean variable by edge and by demand. Since the number of edges is quadratic in the number of vertices, this number may grow a lot for some large networks.

4 Heuristics and problem's relaxation

We have tried a variety of combinations of search strategy and problem's relaxation to improve the resolution of this problem. In this paper, we will refer to a particular combination by A/B/C where A is the variable selection strategy, B the value selection strategy and C the type of relaxation to compute the problem's bound, as explained below. At a given node of the search tree, some demands or some partial paths may already be assigned. Apart from classical CP heuristics, all heuristics and lower bound computations use the *residual graph* obtained by considering this part already fixed.

4.1 Residual graph

For each demand, a residual graph is maintained all along the search. This graph is the cornerstone to solve efficiently this problem. It is used by the search heuristics and the relaxation technique. A residual graph is modified incrementally at each search tree node. We refer as *path* the edges belonging to the path as it is in the current search tree (i.e. the instantiated variables) and as *future path* the path's part which is not instantiated in the search tree but computed by the Dijkstra algorithm. A residual graph is built such that:

- It exists a directed edge from the node i to $j \leftrightarrow j \in D(V_i)$
- The cost of an edge is dynamically set and updated by the variables' values. When a variable is instantiated (i.e. an edge is added to the path), the edge's minimal cost is then updated. For instance, when a demand goes through an edge, its congestion cost is automatically updated with the current demand's bandwidth and that for all residual graphs. However, it is not possible to take into account the congestion in future path. It means that two demands which can take the same link act like if they do not create congestion in their future paths.

The residual graphs are constructed with the CP variables, therefore the graphs are modified at each decision or propagation automatically.

Example 2 (residual graph) Two demands d_1 and d_2 having each a bandwidth of 2, have to be routed in the 4 nodes networks shown in Figure 4. In this network, the cost of each edge is 0 and the congestion parameters are respectively $a = 1$ and $b = -0.5$. At first, the residual graphs are constructed at the root of the search tree (the edge sets representing the paths are empty). The costs are initialized only with the bandwidth induced by the demand. For instance in Figure 4a, the residual graph costs of d_1 are computed only knowing the bandwidth of d_1 , no assumption can be done about d_2 . The costs are thus 2 obtained by $:2 \times e^{\frac{2}{4}-0.5}$. In the next step shown in Figure 4b, d_1 's path is expanded with the edge between nodes 0 and 1. This decision updates the cost of the residual graph of d_2 . The new cost is $2 \times e^{0.5}$, it corresponds to the cost when the two demands take the same edge. In Figure 4c, d_2 is going through the same edge as d_1 , thus the residual graph of d_1 is updated.

4.2 Search strategies

Path-oriented problems are particularly sensitive to search strategy, and not surprisingly, a standard dynamic CP search strategy (denoted by CP in this paper) like impact or activity would be of weak efficiency for this type of problem. Indeed, it is likely that this search strategy will label any node in the path without knowing if it could be linked to the source or destination. Therefore, we propose a variable's value selection strategy as well as three variable selection strategy, all dedicated to this SDN problem.

4.2.1 Value selection

For each variable, the value search strategy determines the path's direction. Since the goal is to find the best path for each demand, it would be inefficient to start the path in a wrong direction. We have chosen to label path variables in order of *increasing* path cost. In order to start with the most promising path, we maintain at each node of the search tree the shortest path to the destination in the residual network for each demand in isolation. In other words, given a variable v_i^d and the shortest path SP_d for the demand d the variable is going to be instantiated as follow:

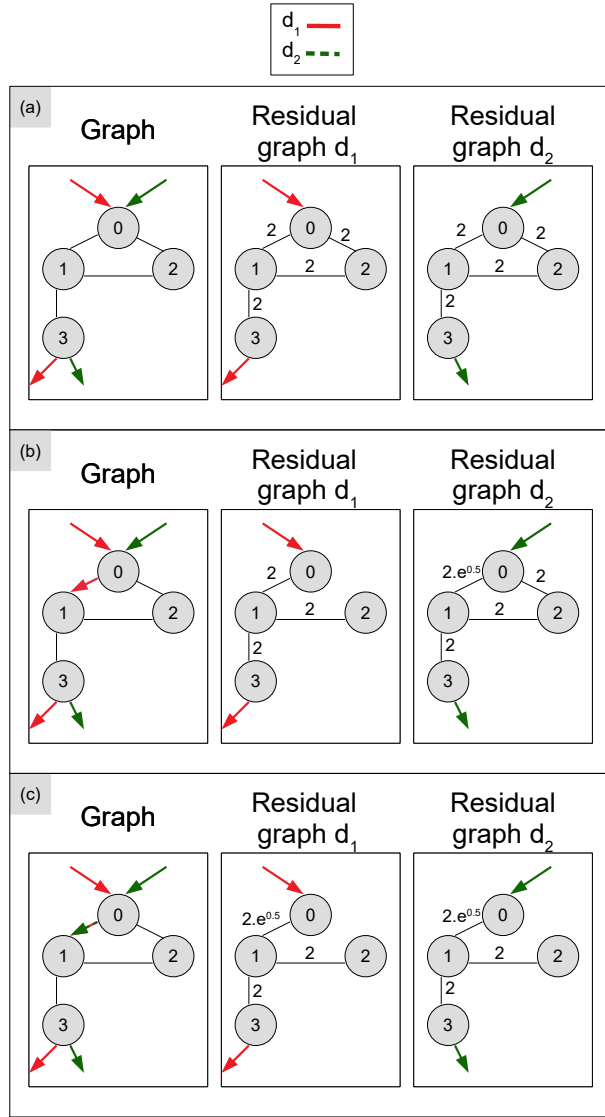


Fig. 4 Residual graph updates examples

$$v_i^d = SP_i(v_i^d), \quad \text{if } v_i^d \in SP_d$$

$$v_i^d = i, \quad \text{otherwise}$$

Where $SP_d(v_i^d)$ gives the successor of the node v_i^d for the demand d 's shortest path. We call this value strategy SP (for *Shortest_Path*). It is done with Dijkstra's algorithm, considering the progression of the already assigned part of the other demands. This

information on the best future path is used to choose the next node of the path when needed (i.e. the variable value). Note that the Dijkstra algorithm only considers the nodes of the paths already assigned at a given point of the search tree for computing the congestion. In particular, the congestion is not cumulative for two demands which share the same future link. The same idea has been implemented in [10] but with specific path variables.

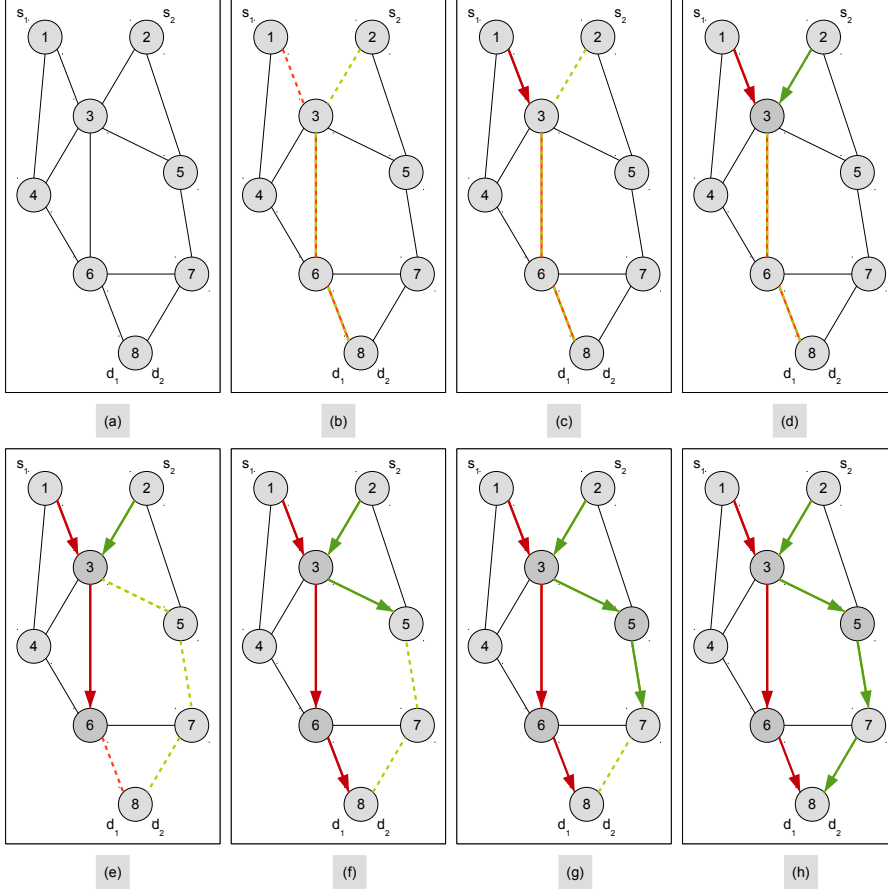


Fig. 5 Labeling of paths for two demands

Example 3 In Figure 5 is represented a small example of two demands being routed on a 8-nodes network (5a) by shortest path heuristic. This example aims to show the dynamical aspect of the shortest path computation. The demand r_1 has to be routed from the nodes 1 to 8 and the demand r_2 from the nodes 2 to 8. The source and destination of r_1 (resp. r_2) are the nodes labeled by respectively s_1 and d_1 (resp. s_2 and d_2). Actual paths taken by the demands are depicted by solid arrows while shortest paths computed by the search strategy are with hatched lines. In other words,

the solid lines represent the variables already instantiated in the problem while the hatched ones represent the current shortest path computed by the value search strategy. At first (5b), the two demands compute their shortest paths: $(1 - 3 - 6 - 8)$ for r_1 and $(2 - 3 - 6 - 8)$ for r_2 . The shortest future paths correspond to hatched lines. In (5c), one labeling step is performed for r_1 . Since there is no change on r_2 's path, no update of r_2 's shortest path is necessary. Hence in (5d) one step is performed for r_2 . In (5e), the next move of r_1 causes congestion on the link from 3 to 6. Thus r_2 updates its shortest path to $(2 - 3 - 5 - 7 - 8)$ in (5f) to lower its minimal cost. It yields a next move by r_2 in the direction of node 5 in (5g). Then the last edge is selected, resulting in the complete paths instantiation (5h).

4.2.2 Variable Selection

Once again, a variable strategy selecting the variables outside a path scope would be very inefficient, this is why we choose to select the variables all along the paths. In other words, our variable selection respects the path order, it selects the next uninstantiated successor variable along the path. Note that this is a partial variable selector since it is only once the demand is chosen that the actual variable is determined by the next step to be extended. For the variable selection to be completely defined, we have considered three strategies for choosing the demand. The first one, called MB (for *Max_Bandwidth*), consists in routing the next remaining demand with the maximum asked bandwidth up to its completion. Then we have defined two strategies based on conflicts analysis. The strategies react on a solution (by MB if no fail occurs) or when a fail occurs. For each demand and each link, we compute the marginal cost (with congestion) induced by the presence of the demand on this very link. The marginal cost corresponds to the difference between the cost with and without routing the demand, all being equals. Then, we sum up all these numbers for each demand along the taken path to obtain a score. The first one, called CO (for *Conflict*), chooses the demand of the highest score and develop its path up to the destination. The second one, called CO1 (for *Conflict_1_Step*), also chooses the demand of the highest score but only develops one step in the path before reconsidering the situation. In CO1, the conflicts are stored for each path variable and for each demand and scores are only computed for the uninstantiated variables.

Example 4 (Strategies in action) In this example, we show the selection process of the three strategies with SP value search strategy. Three demands d_1 , d_2 and d_3 with respectively a bandwidth of 4, 3 and 2 have to be routed in a 6-nodes network. To keep things easy, all edges have a capacity of 7, a cost of 0 and congestion parameters are set to $a = 1$ and $b = -0.5$. The selection process of MB, CO and CO1 are shown respectively in Figure 6, Figure 7 and Figure 8.

MB Strategy. Given the network in Figure 6a), MB selects the demand having the highest bandwidth to be routed. At first, d_1 is chosen and is instantiated from its source to its destination (see Figure 6b). Afterwards, the next demands to be instantiated are going to be iteratively d_2 and d_3 (Figure 6c and d).

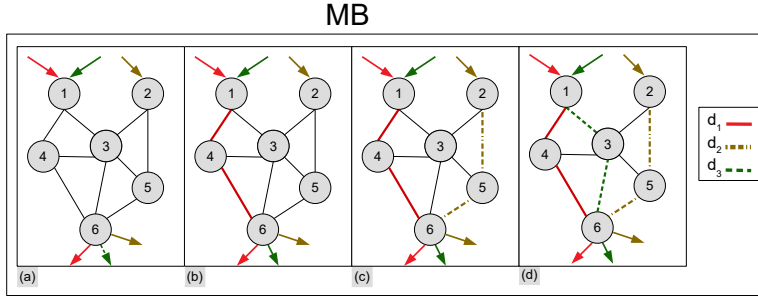


Fig. 6 Selection process of MB strategy

CO strategy. Given the network in Figure 7a), where a first solution has been found. CO analyses the conflicts in the solution to try to redirect the conflicting demands. In this solution the demands d_1 and d_2 are in conflict. The marginal costs are computed for the demands as follow:

$$\Delta(price_{d_1}) = 2 \times (7 \times e^{\frac{3}{7} + \frac{4}{7} - 0.5} - 3 \times e^{\frac{3}{7} - 0.5}) = 20.19$$

$$\Delta(price_{d_2}) = 2 \times (7 \times e^{\frac{4}{7} + \frac{3}{7} - 0.5} - 4 \times e^{\frac{4}{7} - 0.5}) = 17.47$$

The demand d_1 is the one having the highest score and thus selected to be routed (see Figure 7c). And then to finish the demand d_2 is the last one to be routed until its destination (see Figure 7d).

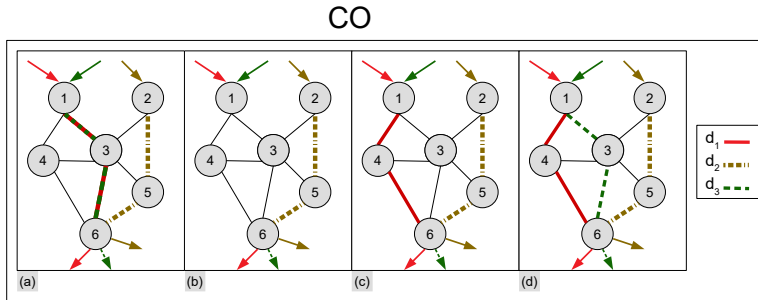


Fig. 7 Selection process of CO strategy

CO1 strategy. The initial situation of CO1 is the same as CO: a first solution has been found. CO1 analyses the conflicts as well as CO but instantiates only edge by edge while selecting the demands with the highest conflict score on the non instantiated variables. The marginal costs for a demand d_i , given an edge from the nodes i to j , named $\Delta(price_{d_i}(n_i, n_j))$ are computed as follow:

$$\begin{aligned}\Delta(\text{price}_{d_1}(1,3)) &= (7 \times e^{\frac{3}{7} + \frac{2}{7} - 0.5} - 3 \times e^{\frac{3}{7} - 0.5}) = 10.09 \\ \Delta(\text{price}_{d_1}(3,6)) &= (7 \times e^{\frac{3}{7} + \frac{2}{7} - 0.5} - 3 \times e^{\frac{3}{7} - 0.5}) = 10.09 \\ \Delta(\text{price}_{d_2}(1,3)) &= (7 \times e^{\frac{4}{7} + \frac{3}{7} - 0.5} - 4 \times e^{\frac{4}{7} - 0.5}) = 8.74 \\ \Delta(\text{price}_{d_2}(3,6)) &= (7 \times e^{\frac{4}{7} + \frac{3}{7} - 0.5} - 4 \times e^{\frac{4}{7} - 0.5}) = 8.74\end{aligned}$$

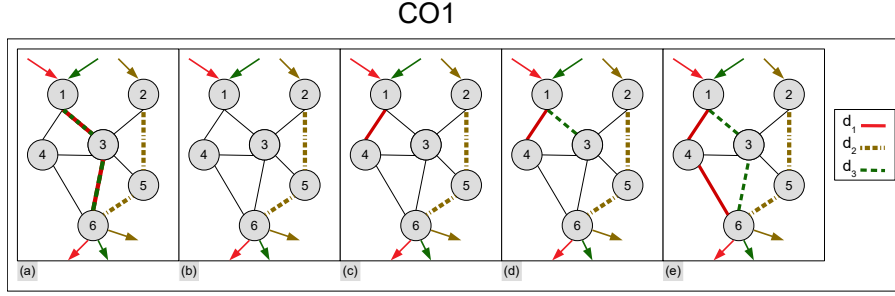


Fig. 8 Selection process of CO1 strategy

After a solution was found, the solver backtrack until the Figure 8b. Then it selects the demands with the highest score and instantiates its first element on the path, which corresponds to the edge between the node 1 and 4 (see Figure 8c). Afterwards, d_3 has the highest score since the edge between the nodes 1 and 4 is not anymore considered for the demand d_1 . d_3 is routed by the edge between the nodes 1 and 3 (see Figure 8d). This process is continued until all the destinations are reached (see Figure 8e)

4.3 Problem's relaxation

Relaxation techniques are commonly used in constraint optimization. However, CP solvers offer a restricted and uninformed version. When minimizing the variable *ProblemCost* and after having found a solution of value A , it simply adds to the remainder of the search the constraint $\text{ProblemCost} < A$. The CP solver is unaware of the problem structure. While efficient, it requires that the lower bound of *ProblemCost* to exceed A to cut the search tree and backtrack. In our case, the possible values of *ProblemCost* are strongly constrained by the current branch of the search tree leading to a node, but very loosely for the remaining part of the problem. In order to cut earlier, we need a better estimation of the lower bound of *ProblemCost*. This is done by adding to the lower bound the cost of individual routing along the path computed by the Dijkstra algorithm used for the value search strategy. We use the previously defined *residual graph* in which congestion is taken into account to estimate the cost lower bound of the current search tree state. We need this to provide a better yet safe estimate of the lower bound which does not exceed

the future real cost. We call the classical CP relaxation CP and the one which uses the bound provided by the shortest path SP .

Let $[a_1^d, \dots, a_i^d, \dots, a_{n_d}^d]$ be a demand's path composed of a first part $[a_1^d, \dots, a_{i-1}^d]$ assigned by the search tree and a second part $[a_i^d, \dots, a_{n_d}^d]$ computed by the Dijkstra algorithm from node a_i^d . We have $a_1^d = \text{src}(d)$ and $a_{n_d}^d = \text{dst}(d)$ and $\forall j < i$, the value of a_j^d is given by the instantiated part of the path in $[v_1^d, \dots, v_n^d]$ (up to the current node of the search tree). The cost contribution of demand d is given by:

$$\begin{aligned} \text{cost}(d) = & \sum_{\{e=(a_j^d, a_{j+1}^d) \mid j < i\}} bw(d) * (\text{cost}(e) + \text{cong}(e)) + \\ & \sum_{\{e=(a_j^d, a_{j+1}^d) \mid i \leq j < n_d\}} bw(d) * \text{cost}(e) \end{aligned} \quad (7)$$

Proposition 5 *Given a monotonic cost function (see equation (3)), the bound given in equation 7 is sound.*

Proof Suppose by contradiction that the proposition is not correct and the equation is not sound. This statement implies that it exists at least one node's cost which is overestimated by the Dijkstra algorithm. The latter is either located on the instantiated nodes or on the future path. This is impossible because the given costs corresponds to the lower bound and are at worst underestimated. That is why Dijkstra algorithm and thus the computed path is computing correct lower bound for the shortest path algorithm.

Note that, due to the presence of link capacity constraints, a fail is triggered when Dijkstra algorithm is unable to find a path from the source to the destination [47].

Example 6 The example in Figure 9 illustrates how the relaxation technique based on shortest paths works. Two demands: d_1 and d_2 have to be routed through a 5 nodes network (see Figure 9(a)). Each demand has a bandwidth of 2 and each arc in the network can transport 4 units of bandwidth. The two demands (d_1 and d_2) have both node 0 as source and respectively nodes 3 and 4 as destination. The parameters of the congestion cost function are $a = 1$ and $b = -0.5$. To simplify the problem, each edge's cost is 0.

In each subfigure is depicted on the left the state of the current graph with the decisions already taken and on the right the residual graphs of d_1 and d_2 . The shortest path algorithm of each demand is computed on its own residual graph. Because it is implemented on the CP variables, the SP computation is aware of the current bandwidth and the successor variables in order to consider only feasible paths.

In the beginning of the problem resolution, the initial propagation is triggered, updating the minimal reachable global cost. To do so, the cost of each demand is evaluated (see Figure 9a). The minimal possible cost corresponds to the demand shortest path without any added congestion due to other demands. For d_1 and d_2 it is obtained by the following computation: $2 \times 2 \times e^{\frac{2}{4}-0.5} = 4$. For each demand, the shortest path's cost is 4. Then the problem is explored by instantiating a first edge for d_1 (Figure 9b). The cost is updated in the residual graphs. Taking the edge from the nodes 0 and 1 costs now $2 \times e^{0.5}$, this update is done in the residual graph of d_2 . In the residual

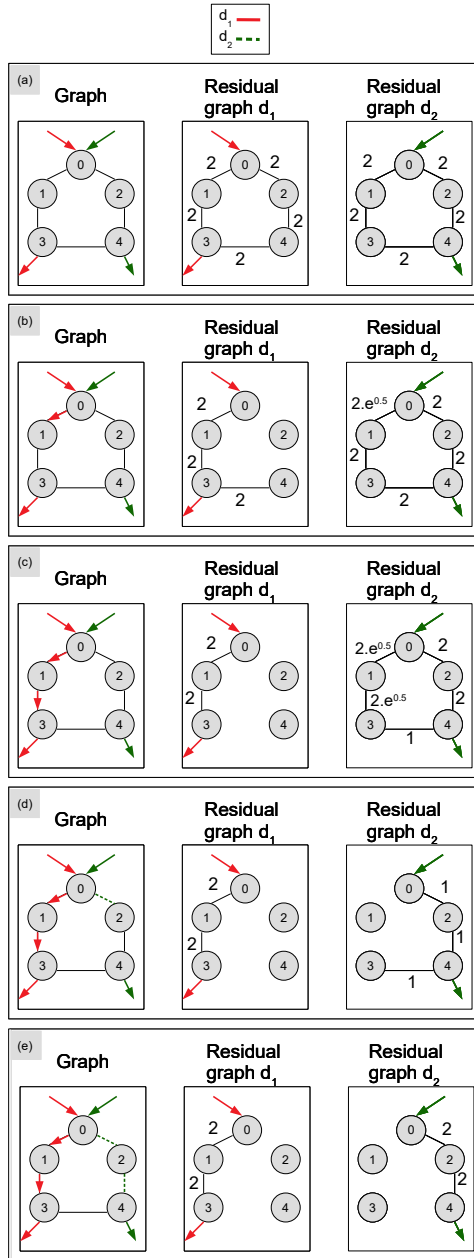


Fig. 9 Problem's relaxation for SDN

graph of d_1 only the possible paths are updated: node 2 cannot be taken anymore. The same process is repeated when the path of d_1 is expanded (Figure 9c). After, it is the second demand which is routed (Figure 9d and Figure 9e). While taking these

decisions, the residual graph of d_2 is updated by removing the edges between nodes 1 and 3 and nodes 3 and 4. The edges of the residual graph of d_1 are not impacted since d_2 does not take the same edges. The solution found has a cost of 8. The Dijkstra relaxation help to state that it does not exist better solution since at the beginning the lower bound for the problem was also 8. The problem's exploration is thus finished.

5 ILP model

ILP techniques are commonly used to solve multicommodity flow problems [3], even in the context of SDN [37]. However, the model we presented in Section 3 is not suited to an ILP formulation because it is very difficult to model paths as in CP. Instead, most formulations either use a flow model or use a pre-computation of paths for the different demands and associate a Boolean variable to each possible path. We will use this technique despite it yields an exponential number of variables. But they can be generated on the fly using column generation.

5.1 Master Problem

First we reformulate the multicommodity flow problems with Boolean path variables in what we call a Master Problem, then we provide a linearization and the pricing problem used to introduce new columns. For each demand $d \in D$, we associate the set P_d of all paths from $src(d)$ to $dst(d)$. By a slight abuse of notation, we also call p a Boolean variable associated to a path $p \in P_d$ because paths are only manipulated through their Boolean variable. Because a path is statically defined and because we need to sum up the bandwidths associated to the various arcs of the network in order to enforce the capacity constraints, we associate to a path variable p and each arc $e \in E$, a variable p_e which is true if the arc e is taken by the path p . Note that this variable p_e is used just to simplify the notation and does not belong to the implemented model. We ensure that exactly one path is chosen for each demand:

$$\forall d \in D, \quad \sum_{p \in P_d} p \geq 1 \quad (8)$$

The capacity constraints become:

$$\forall e \in E, \quad \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \leq cap(e) \quad (9)$$

We aggregate all costs in the following expression to be minimized:

$$\min \quad \sum_{e \in E} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times (cost(e) + cong(e)) \quad (10)$$

Where the congestion is defined by equations 1 and 2. There are two sources of non-linearity in these formulas. First the load of an arc uses an exponential function. It yields that it is easier to break up equation 10 in two for its linearization. A first part

we call *cost with congestion* $cwc(e)$ for a given arc e and a subsequent aggregation on the set of demands:

$$cwc(e) \geq \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times (cost(e) + cong(e)) \quad (11)$$

Note that since we deal with a minimization problem, only the \geq part of the equation is mandatory to enforce equality. Then the expression to be minimized is:

$$\min \sum_{e \in E} cwc(e) \quad (12)$$

But then a more subtle source of non-linearity is that, since the cost depends on the load and the load depends on the path chosen for each demand, we have to consider for the cost the cases where two or more demands are routed by the same arc. It yields a product between the Boolean variables p_e^d and $p_e^{d'}$ for any pair $d, d' \in D$. We now address these two relaxations.

5.2 Column generation

The problem we get with a model based on paths and its subsequent linearization involves an exponential number of variables (since there are exponentially many paths between a source and a destination). Moreover, only one variable for each demand will be set to 1 because we seek a single path for each demand. It is impossible to represent all these variables but fortunately they can be generated on the fly (along with the constraints they are subject to) using Column Generation (see Figure 10).

Column Generation alternates between solving the linear Restricted Master Problem with a limited number of variables (or columns) and generating new variables by solving a sequence of subproblems called Pricing Problems. The first step before iterating is to initialize the linear Restricted Master Problem (or RMP) with initial columns. It is then possible to get dual values and to compute reduced costs. A reduced cost is associated to a dual variable and tells how much the objective changes if this variable increases by a small amount. In other words, it is the first derivative from a certain point on the polyhedron that constrains the problem.

Column generation methods were invented from the observation that often in problems many variables do not belong to the optimal solution and thus their values are set 0 and not used. The idea is to try to generate only the columns useful to solve optimally the problem. For instance in our problem, often only few paths are needed to find and prove the optimal solution.

A Pricing Problem is used to determine which column should be introduced. It yields either to add a new variable or to ensure that there are no further variables with negative dual feasibility i.e. which can potentially improve over the current solution. When no more column can be generated, the linear solution is rounded to give an integer one.

We consider only column generation at the root node. This method can be incomplete unlike Branch and Price which is a bit different since it considers a tree obtained by solving the ILP problem for different sets of columns.

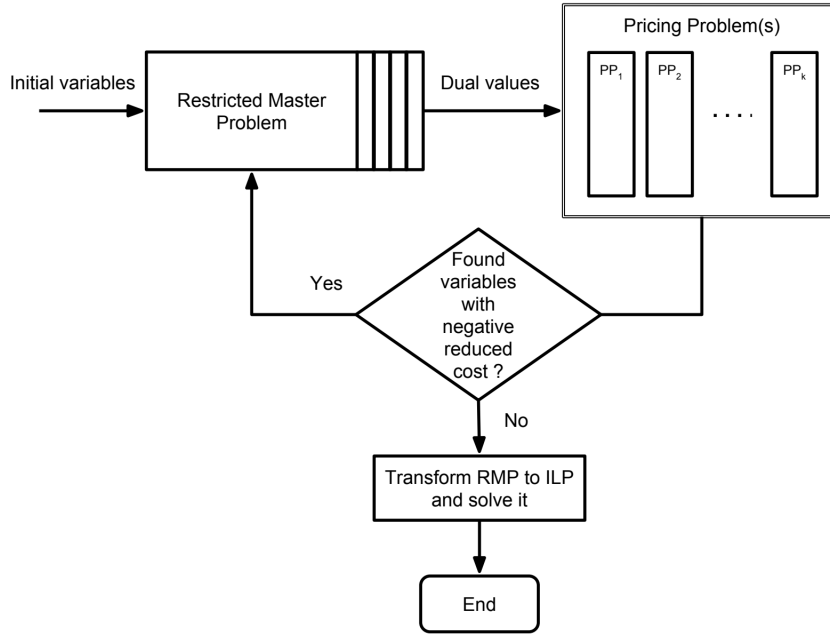


Fig. 10 Column generation procedure

5.3 Linearisation of the master problem

What we call Linearized Master Problem (or LMP) is essentially a linear approximation of the Master Problem introduced above. It means that the solutions we will find with ILP are solutions to the approximate model and not exact solutions of the original problem. However, if the linearization is good, it is likely that the solution paths for the demands will be the same as if the exact model was solved, although it cannot be ensured in all cases. In practice, we have not observed any difference.

The first thing to come is to transform the Boolean variables into continuous ones in the interval $[0..1]$.

5.3.1 Piecewise linear approximation of the exponential function.

One of the relaxation concerns the exponential function. We approximate it with multiples tangents. Let I be a set of numbers in $[0..1]$. For each point of the exponential curve $(i, \text{cong}(i))_{i \in I}$, a tangent $t_i(x) = a_i x + b_i$ is computed. In Figure 11 is depicted a 3-points approximation of an exponential function. The red lines correspond to the computed tangents approximating the function. The difference between the approximation and the real function is shown in light gray. In this zone, the congestion is underestimated and may induces less filtering. For the sake of simplicity, we use the same set I for all links of the network.

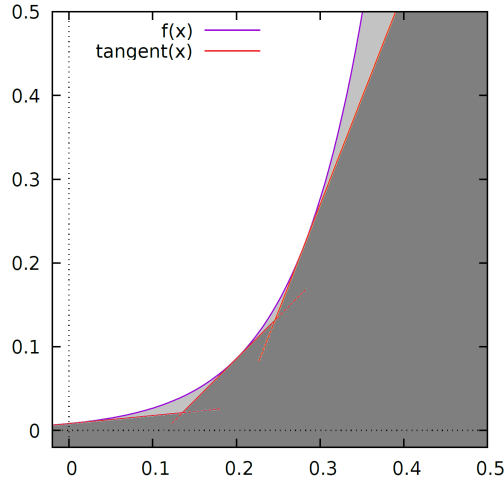


Fig. 11 Three pieces linear approximation of the exponential function

The approximation of the exponential correspond to the maximum value of these tangents $\max_{i \in I} t_i(\text{load}(e))$. In order to get a linear formulation of the maximum, we can introduce for each arc e and each $i \in I$ a variable $\text{cong}_i(e)$ giving the value of each tangent for a given load and one variable $\text{cong}(e)$ for the maximal value. The $\text{cong}_i(e)$ reuses the definition of the load given in equation 1:

$$\forall e \in E, \forall i \in I, \quad \text{cong}_i(e) = \left(\frac{a_i}{\text{cap}(e)} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \right) + b_i$$

And for each link, the following constraints are added:

$$\forall e \in E, \forall i \in I, \quad \text{cong}(e) \geq \text{cong}_i(e) \quad (13)$$

5.3.2 Linearization of the products.

Unfortunately, when computing the cost with congestion $\text{cwc}(e)$ of an arc e with equation 11, all demands crossing this arc actually cause the congestion to increase. If we develop the formula by mixing equations 11 and 13 with respect to each $i \in I$, it yields for a given edge e :

$$\forall i \in I, \quad \text{cwc}_i(e) \geq \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times \left(\text{cost}(e) + \left(\frac{a_i}{\text{cap}(e)} \sum_{d' \in D} \sum_{p' \in P_{d'}} p'_e \times bw(d') \right) + b_i \right)$$

By splitting the linear and non-linear part we get:

$$\begin{aligned} \forall i \in I, \quad cwc_i(e) \geq & \\ & (cost(e) + b_i) \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) + \\ & \frac{a_i}{cap(e)} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times \sum_{d' \in D} \sum_{p' \in P_{d'}} p'_e \times bw(d') \end{aligned}$$

The last expression is quadratic because it contains a product between p_e and p'_e . To get a linear formulation, we introduce new Boolean variables pp'_e for each arc e and each path p for d and each path p' for demand d' such that pp'_e is true if and only if p and p' share e as common arc. Since we model only one simple path by demand, we can use another trick by summing all the path for each demand. The meaning of the pp'_e can be reformulated as: pp'_e is true if and only if it exists p and p' for respectively d and d' that share the arc e .

We implement the logical AND (see chapter 7 of [4]) by this set of linear constraints:

$$pp'_e \leq \sum_{p \in P_d} p_e, \quad \forall e \in E \quad (14)$$

$$pp'_e \leq \sum_{p' \in P_{d'}} p'_e, \quad \forall e \in E \quad (15)$$

$$pp'_e \geq \sum_{p \in P_d} p_e + \sum_{p' \in P_{d'}} p'_e - 1, \quad \forall e \in E \quad (16)$$

Then the cost constraints can be reformulated as follows:

$$\begin{aligned} \forall e \in E, \forall i \in I, \quad cwc_i(e) \geq & \\ & (cost(e) + b_i) \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) + \\ & \frac{a_i}{cap(e)} \sum_{d \in D} \sum_{p \in P_d} \sum_{d' \in D} \sum_{p' \in P_{d'}} pp'_e \times bw(d) \times bw(d') \end{aligned}$$

As in equation 13 we aggregate all costs for the different tangents:

$$\forall e \in E, \forall i \in I, \quad cwc(e) \geq cwc_i(e)$$

And thus the expression to be minimized as in equation 12 becomes:

$$\min \sum_{e \in E} cwc(e)$$

5.4 Pricing problem

The reduced cost for a given variable determines how the objective changes if the variable increase of one unit. A Linear problem is optimal if its reduced cost is 0. However, if the reduced cost is negative, the solution can enter the basis as a new column. If the reduced cost is greater or equal than zero, the lower bound for the optimal solution has been found, although this may not be an integer solution. Note that the reduced cost can be computed on each edge individually. In order to find an improving path for each demand, we could perform a shortest path computation with Dijkstra's algorithm on the graph where arcs are labeled with reduced costs. The new variable of the discovered path already implicitly exists, and we just compute it on the fly. When it is not possible to improve the LP solution, it will be also not possible to find a path such that the reduced costs are negative.

Note that in our problem the decision variables (i.e. the paths) are not directly present with a coefficient in the objective function but instead appear through pp'_e . And thus, the coefficients of the decision variable do not appear in the pricing problem.

In order to formulate the dual, let us give names to the constraints of the problem. We consider only the constraints that are potentially affected by the introduction of a new column. Let us call ONE_d the constraint given in equation 8, CAP_e the capacity constraint given in equation 9, and $AND1_e$, $AND2_e$, and $AND3_e$ respectively the constraints in equations 14, 15 and 16. By using these dual values found when solving the RMP, we are able to define the graph of reduced costs for a given demand d . For each edge e and demand d , we have:

$$rcost_d(e) = -CAP_e \times bw(d) + \left(\sum_{d' \in D} AND1_e + AND2_e - AND3_e \right)$$

Then the pricing problem for each demand d become now finding a shortest path in the graph of reduced costs, i.e. which minimizes the following formula for a path p defined by its Boolean variables p_e :

$$\min \left(ONE_d + \sum_{e \in E} p_e \times rcost_d(e) \right)$$

Unfortunately, the network labelled with reduced costs has negative cycles and thus Dijkstra's algorithm cannot be used to find a shortest path. Since we are only interested in simple path (i.e. a path without cycle), the pricing problem can be solved through a new Integer Linear Problem by the following flow model. Like before, let p_e be the (continuous) variable associated to the arc e .

The following constraints ensure that only one unit of flow comes out from the source of the demand d and nothing enters in, and the reverse for the destination.

$$\begin{aligned} \sum_{e=(src(d),y) \in E} p_e &= 1 & \sum_{e=(x,src(d)) \in E} p_e &= 0 \\ \sum_{e=(x,dst(d)) \in E} p_e &= 1 & \sum_{e=(dst(d),y) \in E} p_e &= 0 \end{aligned}$$

Here are the flow conservation constraints:

$$\forall v \in V, \sum_{e=(x,v) \in E} p_e - \sum_{e'=(v,y) \in E} p_{e'} = 0$$

Then we state non-splittability and no-cycle constraints:

$$\forall v \in V, \sum_{e=(x,v) \in E} p_e \leq 1 \quad \forall v \in V, \sum_{e=(v,y) \in E} p_e \leq 1$$

The objective becomes:

$$\min \left(ONE_d + \sum_{v \in V} \sum_{e=(v,y) \in E} p_e \times rcost(e) \right)$$

We extract from this flow the minimum path and introduce the corresponding variable.

5.5 Solution

A solution for the ILP model when using Column Generation is not equivalent to a solution with the Constraint Programming model. First the approximation introduced by the linearization of the exponential function tend to underestimate the congestion. Thus the value of the objective may be lower for the ILP model even if the solution paths are the same. Second, we solve the ILP problem only when the Column Generation procedure has ended. It may happen that in some cases this procedure does not terminate in a reasonable time. Then the integer solution is not computed and we get no solution.

6 Constraint Games

In this section, we briefly introduce Constraint Games [30,35], which are an extension of Constraint Programming allowing to find Nash equilibria.

6.1 Game theory and constraint game background

A *game* [15] is a situation where a set of players \mathcal{P} can perform actions and get a reward which depends on their own choice of action, but also on the actions of the other players. Players are selfish and always aim to increase their utility by changing their own action if they have an opportunity to do so. A (pure) *Nash equilibrium* (PNE) [53,29] is a situation where all players cannot improve their own utility by changing their own action. A game may or may not have an equilibrium, and the existence of an equilibrium is an NP-complete problem [17].

In many cases, the efficiency of a solution can be evaluated by an external measure called *social welfare* function which should be maximized. This global function allows to compute the best centralized solution (by discarding the players objectives). Then it is possible to quantify the loss of efficiency induced by the selfish behavior of the players by considering the ratio "best centralized solution / best equilibrium" called Price of Stability (PoS) and "best centralized solution / worst equilibrium" called Price of Anarchy (PoA).

Constraint Games allow to represent in a compact and natural way games with multiple players and also give a powerful solving method by lifting consistency techniques to the equilibrium property [35]. In Constraint games, actions are represented by the possible assignments of controlled variables. Utility is represented with constraint optimization, and the rich language of most constraint solvers is available to express a large spectrum of problems in a concise and meaningful way.

A *Constraint Satisfaction Game* (or CSG) is a 4-tuple (\mathcal{P}, V, D, G) where \mathcal{P} is a finite set of players, V is a finite set of variables composed of a family of disjoint sets $(V_i)_{i \in \mathcal{P}}$ for each player and a set V_E of *existential* variables disjoint of all the players variables, D is defined as for CSP, and $G = (G_i)_{i \in \mathcal{P}}$ is a family of CSP on V representing the *goal* of each player. In a CSG, all players seek for satisfaction of their goal. However, it may happen that a player is not satisfied in an equilibrium if none of his/her move allows for satisfaction. Determining whether a game has a PNE in a Constraint Satisfaction Game is Σ_2^P -complete. Note that [30] has introduced satisfaction and optimization variants of Constraint Games. A *Constraint Optimization Game* (COG) is a variant $(\mathcal{P}, V, D, G, opt)$ where $opt = (opt_i)_{i \in \mathcal{P}}$ and $\forall i \in \mathcal{P}, opt_i \in V$ is the variable whose value defines the utility function u_i of Player i . All players want to maximize their utility.

In addition, Constraint Games are able to represent easily *hard constraints* that define situations which are globally possible or forbidden [44] by adding a global CSP C to the problem. Nash equilibria can only be sought in the satisfiable part of the hard constraints. A global optimization condition on a variable w allows to model the social welfare function. Without further information, we call *Constraint Game* a COG with constraints and social welfare and we refer to it by $CG = (\mathcal{P}, V, D, G, opt, C, w)$.

The solving technique introduced in [30] and further developed in [35] is based on tree search. Players' preferences are represented by *Nash constraints* and their filtering is based on the detection of *never best responses*, which are values that never lead to an improvement. The strong filtering of [35] works only for Constraint Games without hard constraints (or if the hard constraints are functional), otherwise we can fall back to the weaker form of [30], which is the case in this problem because of the capacity constraints on the links.

Incomplete algorithms can also be used to find quickly a first Nash equilibrium. *Iterated Best Response* (or IBR) [48] is the simplest local search algorithm to find a PNE in any game representation. This iterative process starts from any strategy profile. At each step, if there exists a player for whom the current strategy profile is not a best response, then this player deviates to his best response which will be considered as the candidate in the next step. The process stops when all players are no longer able to change their strategy or if the algorithm fails to find an equilibrium in a given time

credit *Max_Step*. In the first case, the last profile is a Nash equilibrium. In this paper, we have used IBR as a heuristic to go from the first solution to the first equilibrium.

6.2 Constraint games for SDN

The MCPRP defined in section 2 can be simply extended to a game by considering each demand as a player who wants to find the best route from source to destination. Then each player wants to minimize her/his own cost as defined in equation 3.

If we denote by $S = D^V$ the total search space and by N the set of Nash equilibria, we can define formally the welfare of the best centralized solution adapted to our cost minimization problem by $W^* = \min\{w(s) \mid s \in S\}$. The welfare of the best Nash equilibrium is defined in a similar way by $N^* = \min\{w(s) \mid s \in N\}$ and the one of the worst one by $n^* = \max\{w(s) \mid s \in N\}$. Thus the Price of Stability is simply $PoS = W^*/N^*$ and the Price of Anarchy $PoA = W^*/n^*$. Note that usually the classical definitions of PoS and PoA yield a result greater than 1, this is not the case here because we have a minimization problem.

In our problem, the social welfare function is simply the global cost to be minimized as defined in equation 4. We proceed in two steps. First the best centralized solution is computed as a Constraint Optimization Problem, then the Nash equilibria using our Constraint Games solver. We can immediately see that PoS and PoA are asymmetric in term of the relaxation technique we can implement. For PoS, the problem is still a minimization. Thus we can use the same relaxation technique as the one we use in the centralized version (equation 7).

For the PoA, we have a maximization problem. But each player still wants to minimize her/his cost. The situation is then to find a set of *shortest* paths of *maximal* global cost. The standard relaxation technique provided by the CP solver provides a loose upper bound for this problem by summing up all upper bounds of the costs of the edges. But we know that the upper bound is at most the cost of the longest path in the residual network. Unfortunately, computing the longest path is NP-complete in the general case, since it corresponds to determine if it exists an Hamiltonian cycle, which is NP-complete [16]. This problem has been already addressed in CP [38] where the authors propose a model and a local search algorithm to solve this problem. In our case, we are interested in a polynomial sound algorithm. This is why we propose to approximate the longest path by a Maximum Spanning Tree (MST) in the residual graph. The MST is computed by considering the upper bound value of the cost of the edges. The algorithm is like Prim's algorithm, we add to all remaining edges the cost of the demand to compute congestion, then we start by taking the most costly edge and add edges linking a new node in descending order of cost. It is clear that the cost of the MST is always greater than the cost of the longest path.

7 Related work

Combinatorial methods. SDN allows fast and remote network reconfiguration and thus is a nice source for many optimization problems [25] including online ones. Due to this dynamic aspect and the increasing size of the controlled networks, it is very likely that decentralized algorithms will be mandatory to provide both the expected quality of service and short time response. A survey of most techniques can be found in [26]. Since then, many extensions have been considered like the very important case of demands coming online [18,37], service provisioning [21], energy-aware routing [22], controller placement [41,46], fault prevention [51,6] or even congestion-aware algorithm [49].

Concerning the specific CP framework and to our best of our knowledge, only a few works have been considered: a problem of Service Function Chaining deployment [27] and a general framework providing through CP a high-level programming language to model SDN problems [24].

Our article differs from these methods because first, we propose a CP model taking into account non-linear congestion. Then, we optimize our model by proposing a relaxation technique based on Dijkstra algorithm as well as fast heuristics to solve the problem to optimality.

Quality of service and Game theory Quality of service (or QoS) is an important problem in SDN and has been addressed in multiples ways. From combinatorial methods with for example with genetic algorithms [42], or even multiples linear programs [50] optimizing multiples criteria such as bandwidth or energy consumption. These criteria concern the whole network which is different from game theory wherein each flow is considered as a criterion. A mechanism design method for multicommodity flow games has been proposed [12]. Nonetheless, Game theory studies have been mainly concentrated with routing games [45] to model uncapacitated networks in order to determine how selfish behaviors impact solutions and to quantify it by the price of anarchy [12]. Other mathematical studies on more general networks and solution's degeneration have been done such as on on capacitated network [11], unsplittable flow [1]. And even on different model based on distributed games [23,19].

Our approach is different since we propose a model to compute the PNEs and POA. The two Relaxations are given to fast the exact computation of the PNEs and POA giving a more practical way to this kind of problems.

8 Instances and experimental results

We have tested our framework on a library of instances called SNDlib [33] and a personal problem generator that is able to generate instances close to real ones. This problem and other variants such as robust SDN networks have been studied for a long time [14,7] with some computational approaches including linear programming [2]. A survey can be found in [28]. Interesting theoretical results have been found, like the one which states that when the problem has a sufficient size and capacity, all flows are actually routed along single-paths [40].

8.1 Generator

We have designed a generator to create synthetic problems that allow to test the algorithms against the different hypothesis. Several parameters allow obtaining a great variety of graphs. The generation process is mainly constituted of two phases:

- Generation of the topology, that is nodes as well as arcs and their respective costs;
- Generation of demands, along with their bandwidth requests, which also determines the capacities of the arcs.

During the generation of the topology, N_{nodes} nodes are created. Each of these nodes n_i with $i \in [1, N_{nodes}]$ is assigned to random coordinates in a fixed size space of $topologyDimension$ dimensions. In case the boolean $topologicalCost$ is set to *true*, the cost of an arc is given by the distance between the source and destination node. Note that for dimension 2, this is not sufficient to ensure that the resulting graph is planar. The size of the space in one given dimension is irrelevant, as we refer to it only with percentage. Each node is also assigned to a degree, randomly chosen in an interval $[deg_{min}, deg_{max}]$. To obtain graphs similar to actual networks, we introduce hubs which are nodes of higher degree than regular nodes. Each node has a probability P_{hub} of being a hub. If a node is a hub, then its degree is randomly chosen in a different interval $[degh_{min}, degh_{max}]$.

We first build a spanning tree over all nodes to ensure that the graph is fully connected, then we create the remaining links in the graph. For each node, we look for candidates, so the desired degree is reached. For a link to be created, we ensure that a) the other node is not already connected with this edge and b) its distance in the space is not greater than $maxDistance$, expressed as a percentage of the space size ($\sqrt{topologyDimension}$ is the maximum). Using this process, it is possible that certain nodes do not reach the desired degree, but as the network grow larger, this situation becomes less and less likely to happen.

Once the topology is generated, $N_{demands}$ are generated. For each of these, a starting node is randomly selected, as well as a bandwidth in a $[bw_{min}, bw_{max}]$ interval. We then generate what we refer to as an "initial path". For that purpose, different strategies are available. The first strategy, called random generation, consists in selecting a random number of hop h in the $[hop_{min}, hop_{max}]$ interval, and randomly navigating in the graph for h hops, starting from the initial node. The last node is then considered to be the destination node of the request. During the navigation, we only make sure to never reach a node that is already in the initial path. The second strategy consists in randomly selecting a destination node, and applying a shortest path algorithm to find the path from the source node to the destination node with the least number of hops. The path yielded by the algorithm is considered to be the initial path. Regardless of how the initial path is constructed, for each of its arc, there is a probability P_{bw} that we increase its capacity of the amount of bandwidth of the request. The list of all generation parameters, as well as short description can be found in Table 1.

Parameter	Type	Range used in the benchmarks
N_{nodes}	integer	50, 75, 100, 120, 140, 160, 180, 200, 500
$topologyDimension$	integer	2
$topologicalCost$	boolean	<i>true, false</i>
deg_{min}, deg_{max}	integer	[1, 2, 4, 8], [2, 5, 7, 10]
P_{hub}	integer	0, 1, 5, 10, 20
deg_{min}, deg_{max}	integer	[25, 50, 75, 100], [25, 50, 75, 100]
$initbw_{min}, initbw_{max}$	integer	[50, 100, 200], [50, 100, 200]
$initcost_{min}, initcost_{max}$	integer	[100, 200], [200, 500]
$maxDistance$	integer	25, 50, 100
$N_{demands}$	integer	30, 50, 100, 120, 130, 150, 200
bw_{min}, bw_{max}	integer	[50, 100, 200], [50, 100, 200]
hop_{min}, hop_{max}	integer	[0], [10]
P_{bw}	integer	[10, 30, 50, 70, 90, 100]

Table 1 Parameters of the generator

8.2 Settings and implementation issues

8.2.1 Experimentation settings

Due to the large number of parameters of the generator, we have applied a benchmark method called combinatorial testing [31,20] using the ACTS software [54]. This technique allows for p parameters and a size c to generate a set of instances where all possibilities of combinations of parameters of cardinality c are inside the set. For example, if we have 3 Boolean parameters a , b and c , a complete test of all possibilities would require $2^3 = 8$ tests. But if we decide to test only all combinations of pairs, we can achieve this with only 4 instances (see Figure 12). With our

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

a	b
0	0
0	1
1	0
1	1

a	c
0	0
0	1
1	1
1	0

b	c
0	0
1	1
0	1
1	0

Fig. 12 All pairs of parameters are covered by 4 tests

generator, by choosing an appropriate sampling of the intervals described in 1, we get roughly 500 instances to get a covering of all 3-sets of parameters. From these 500 instances, we have discarded those whose resolution lead to a timeout for all techniques. This gives a total of 123 instances which give a meaningful picture of the range of problems that can be solved.

The tests have been performed on a cluster of Intel Xeon E5-2690, each having 10 cores sequenced at 3GHz and 256 GB of RAM. We have computed experimental

results for the CP approach described in Section 3 and the Constraint Game model of Section 6 with a timeout fixed at 1 hour.

8.2.2 A note on implementations

The CP model has been implemented using the Choco solver [39], including the Constraint Game through our Choco extension called Conga [35]. Besides the search techniques and different heuristics and relaxation techniques described in Section 4, our first implementation was using the Ibex solver [8,9] in association with Choco. Real variables linked to Ibex were used to model the computation of the load, congestion and costs while discrete decision variables remained in Choco. However, this was not efficient because the two solvers need to communicate through Java Native Interface. In addition, many auxiliary real variables and constraints (e.g. constraint for the congestion cost and auxiliary variable for the sum) were used to compute intermediate values through constraint propagation despite this part is purely functional. In addition, the cost is also obtained as a by-product of the shortest path algorithm since at the end of the search tree the sum over the demands costs computed by the Dijkstra algorithm is the real cost. Therefore, we have replaced all the auxiliary variables and constraints computing the objective value by a single global constraint which also encapsulates the Dijkstra algorithm. At the end, the model only contains path and capacity constraints and the global constraint computing the objective.

The Column Generation model (called CG hereafter) has been implemented using CPLEX [13] version 12 with its Python interface. At first we tried to post all constraints at the problem initialization for all possible paths. However, it was not efficient since it takes a lot of time to initialize. Since most of the constraints (i.e. consider all the edges in the graph) are useless to solve the problem, we instead choose to post the constraints on the fly along each path when it was required after having generated the columns.

Once again, in the following, when we are saying that an instance is solved it has different meaning when we are talking about CP or CG. For CP, an instance is solved when the optimal solution has been found and proved. For CG, the instance is considered as solved when the generation procedure is finished and the ILP problem solved within the generated columns. Because of that, Column Generation does not prove optimality. Because of the linearization, the objective values are most of the time different between the two techniques.

8.3 Experimental results

8.3.1 Constraint programming model

For the synthetic benchmarks, we have displayed the results in Figure 13. As a preliminary test, we have tried the pure CP heuristic based on impact [43] to measure the gap with the SP value heuristics. A problem which should be easy (13 nodes and 9

demands) is solved in less than 1 second by using the shortest path strategy, whereas the impact strategy took 878.969 seconds. Due to this, we have not displayed this CP/CP/CP heuristics in the figure and we only present results for the SP strategy.

instance	#Demands	#Nodes	#Edges	MB/SP/CP	MB/SP/SP
<i>SAT</i>					
dfn-bwin	90	10	45	TO	0.670
dfn-gwin	110	11	47	TO	0.746
di-yuan	22	11	42	TO	0.472
giul39	1471	39	172	TO	26.554
india35	595	35	80	TO	8.739
newyork	240	16	49	TO	3.486
nobel-eu	378	28	41	TO	4.919
norway	702	27	51	TO	7.962
pdh	24	11	34	TO	0.553
<i>UNSAT</i>					
geant	462	22	36	2.729	2.550
germany	662	50	88	6.489	6.241
janos-us	650	26	84	3.508	3.605
janos-us-ca	1482	39	122	25.926	25.926
<i>UNKNOWN</i>					
france	300	25	15	TO	TO
pioro40	780	40	89	TO	TO
polska	66	12	18	TO	TO

Table 2 Results of the Constraint Programming model on real-world instances from SNDlib

For each instance, we have run the combinations MB/SP/CP and MB/SP/SP, and the two conflict variants CO/SP/SP and CO1/SP/SP. The plot in Figure 13 shows how many instances are solved in a specific delay. Clearly, the MB/SP/SP heuristics outperform the other ones. This is not surprising compared with the CP-style B&B,

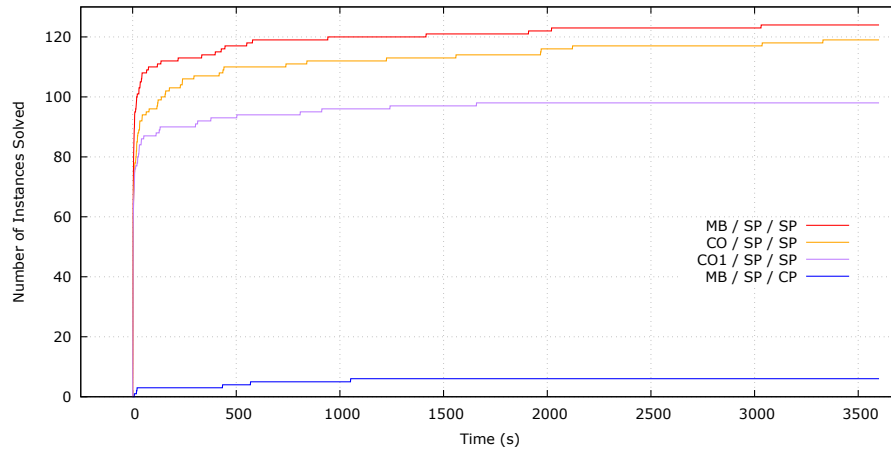


Fig. 13 Comparison of different CP heuristics on synthetic benchmarks

but it shows that a more dynamic heuristic based on conflicts is not effective on this type of problems. We also compared the performances of the different strategies on the unsolved instances. Since, the solver did not finish either because it did not prove the solution's optimality, or because it did not find any solution, the only valuable comparison is the current solution and the time to find a first solution.

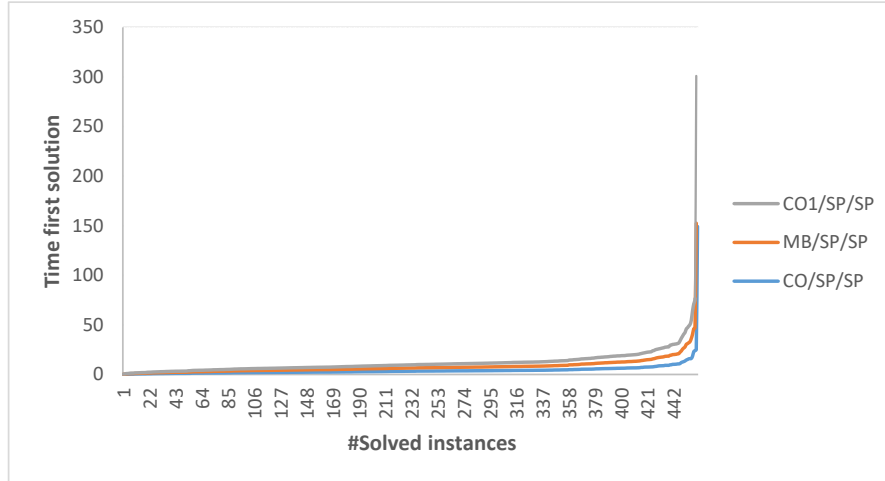


Fig. 14 Comparison of the strategies for finding the first solution

The performances for finding a first solution of the different strategies are shown in Figure 14. This figure presents the time required to find a first solution for all instances and given the three CP strategies. The instances are sorted by increasing time. As we can see, the strategies are very good at finding a first solution and on most of the instances. The strategies provide comparable performances when the goal is to find a first solution. Another interesting comparison is about the performances of the strategies on unsolved instances for getting the best solution. The table 3 presents how many times a strategy has found the best current solution while it timeout. MB is the strategy finding most of the time the best solution after a timeout.

	MB/SP/SP	CO/SP/SP	CO1/SP/SP
# best sol	340	321	303

Table 3 Comparison of solutions on the unsolved instances

It appears that these instances are hard for multiples reasons. First it not a simple parameter which makes those harder. An instance can be hard even with 30 nodes and 30 demands. A problem become hard when the back-propagation of the relaxation is not enough and requires a lot of search. This effect is visible in the Figure 13, while comparing the instance which benefit from the SP relaxation against the one using

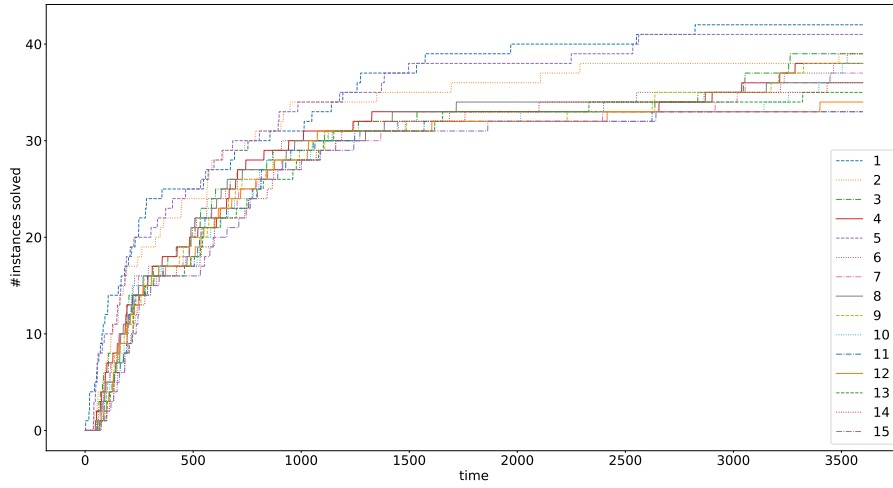


Fig. 15 Comparison of different initialization of the Column Generation model

the CP one. Most of the time the strategies take time to prove the optimality of a solution.

In the following, we only compare with the $MB/SP/SP$ heuristic. The time in seconds for real instances from the SNDlib are shown in Table 2 only for two heuristic combinations. The improved relaxation technique allows to solve many instances to optimality. Each instance is described by its name (which corresponds usually to a network in a particular country), then the number of demands, nodes and links of the network. The SAT and UNSAT instances are the one for which we can find the optimal solution or prove unsatisfiability. For some instances depicted as UNKOWN, our method was unable to find the optimal routing. But still the best solution can be reported.

8.3.2 ILP model with Column Generation

We have run the ILP model on the same synthetic instances as the CP model. In our model, the number of initial columns can be parametrized. We show in Figure 15 a cumulative plot comparing the number of instances solved with different initializations. Each method starts with a different number of path from 1 to 15. It appears that starting with an unique path gives better performances. A reason which can explain this behavior it that if too many path are generated at start, many constraints have to be added and it slows down the initial simplex iterations and the next ones for each demands.

The real instances from SNDlib are shown in Table 4 along with the ratio of objective value between CG and CP. Due to many timeouts, there is no meaningful conclusion to be analyzed.

Instance	#Demands	#Nodes	#Edges	CG	MB/SP/SP	Obj $\frac{CP}{CG}$
<i>SAT</i>						
dfn-bwin	90	10	45	456.742	0.670	1,000067618
dfn-gwin	110	11	47	<i>TO</i>	0.746	N/A
di-yuan	22	11	42	<i>TO</i>	0.472	N/A
giul39	1471	39	172	<i>TO</i>	26.554	N/A
india35	595	35	80	<i>TO</i>	8.739	N/A
newyork	240	16	49	<i>TO</i>	3.486	N/A
nobel-eu	378	28	41	<i>TO</i>	4.919	N/A
norway	702	27	51	<i>TO</i>	7.962	N/A
pdh	24	11	34	108.3804	0.553	1
<i>UNSAT</i>						
geant	462	22	36	<i>TO</i>	2.550	N/A
germany	662	50	88	<i>TO</i>	6.241	N/A
janos-us	650	26	84	<i>TO</i>	3.605	N/A
janos-us-ca	1482	39	122	<i>TO</i>	25.926	N/A

Table 4 Column Generation results on real-world instances from SNDlib

8.3.3 Constraint programming against Column Generation

To get a very synthetic insight of the respective strengths of the two approaches, we have depicted a set of comparisons in Table 5. It simply shows how many times each method has found a better solution or finished the resolution before the other. From this table, we can see that in general the CP model performs better, but not all the time.

	#better run-time	#better solution	#better run-time and solution
Column generation	11	17	0
Constraint Programming	124	382	11

Table 5 Comparison between Column Generation and CP

#Node	#demands	time CP	time CG	solution CP	solution CG
120	80	<i>TO</i>	1906.63	707329	704229
180	90	<i>TO</i>	2245.03	956690	897772
180	50	<i>TO</i>	2095.51	708083	683188
100	30	2.29	<i>TO</i>	4564010	<i>TO</i>
500	10	3.812	547.638	131182	133039
40	20	1.87	208.979	84291.7	86397.5
75	200	<i>TO</i>	182.183	914526	<i>TO</i>
200	40	<i>TO</i>	1651.83	707515	713809

Table 6 Comparison between Column Generation and CP

Furthermore, we extracted some meaningful synthetic instances presented in Table 6. These instances present different kinds of behavior. In a few instances, the CG approach is able to find a better solution in shorter time. Sometimes the CG generation procedure times out and we are not able to find an integer solution in the given time. But interestingly, even if the CG has finished his generation procedure and the CP model times out and fails to prove optimality, it happens that the integer solution found by CG is still worse than the one returned at the end by CP.

We can see that very often CP performs better. One possible explanation about the bad result of Column Generation is how the RMP is linearized. When a new variable is entering the problem, the objective constraint is not modified. And thus the computed reduced cost are less efficient to improve the linear solution. This in turn slows down the global resolution by forcing the generation procedure while it is not required. In addition, the generation of one column needs to solve a NP-complete problem and it has to be embedded in ILP since no shortest path algorithm is applicable due to the negative cycles (see section 5.4). In contrast, the CP model benefits from a good heuristic which guides well the search space exploration, and moreover it has a good relaxation to bound the objective value when the search tree is explored in order to close the nodes.

8.3.4 Constraint games

For the Constraint Game model, we have only used the combination MB/SP/SP, with and without improvement of the first solution by IBR. Results show that IBR improves the relaxation technique by giving quickly a good first solution which is also an equilibrium.

instance	#Demands	#Nodes	#Edges	MB/SP/SP	[NASH] MB/SP/SP
<i>SAT</i>					
dfn-bwin	90	10	45	0.670	3.871
dfn-gwin	110	11	47	0.746	5.681
di-yuan	22	11	42	0.472	2.012
giul39	1471	39	172	26.554	1571.197
india35	595	35	80	8.739	215.716
newyork	240	16	49	3.486	18.173
nobel-eu	378	28	41	4.919	41.861
norway	702	27	51	7.962	154.520
pdh	24	11	34	0.553	2.016
<i>UNSAT</i>					
geant	462	22	36	2.550	2.92
germany	662	50	88	6.241	6.783
janos-us	650	26	84	3.605	5.174
janos-us-ca	1482	39	122	25.926	50.486

Table 7 Constraint Games results on real-world instances from SNDlib

We present in Table 7 the run-time in second of the different strategies on the SNDlib instances. It is interesting to see that games of unprecedented size (up to 1482 players

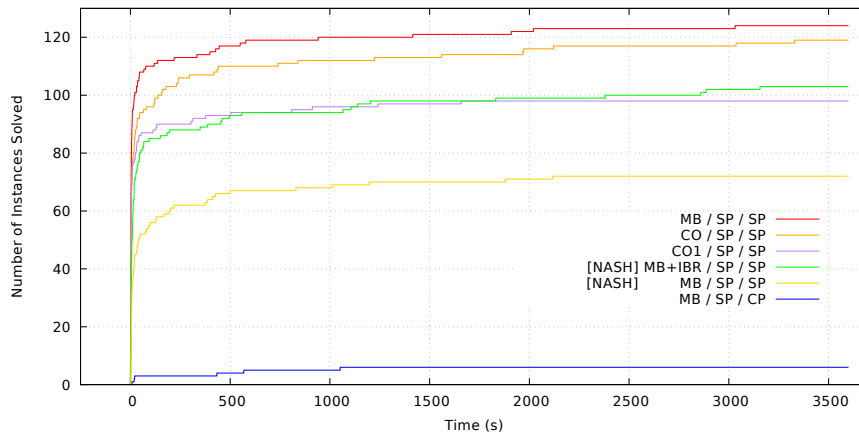


Fig. 16 Comparison Nash and the different heuristics on synthetic benchmarks

in the *janos-us-ca* instance) can be solved to optimality by Conga [35]. Interestingly and in contrast with the synthetic instances, we have observed that IBR slightly degrades the computation time, this is why we did not include the column in the table. We believe that in these problems, most first solutions computed by the MB heuristics were already at equilibrium, and thus adding IBR only adds another check.

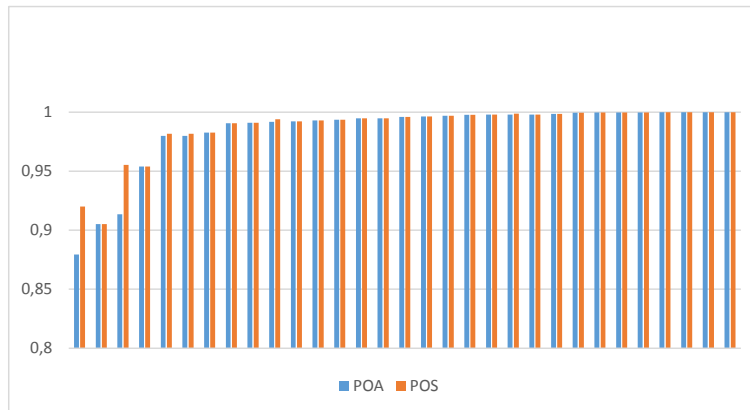


Fig. 17 Price of Anarchy and Price of Stability for small synthetic instances

We report the results for the computation of PoA and PoS for small synthetic instances in Figure 17. In most instances, we observe that the PoA and PoS are very close, and also very close to the centralized optimum. It means that on these problems, a decentralized algorithm would be very interesting to implement if we assume it scales up to larger problems. We have used much smaller instances because the PoA is very difficult to reach. The upper bound computed for the Maximal Spanning Tree overestimates the longest path which also overestimates the longest shortest path. We

pay these two approximations by a limited pruning of the search tree which has a major impact on the computation time.

9 Conclusion

This paper includes two practical contributions. First we have modeled and solved efficiently the unsplittable multicommodity flow routing problem with congestion in Constraint Programming and in ILP with Column Generation. We have provided an accurate relaxation technique that allows to solve real-world size instances up to optimality. Our third contribution is a Constraint Game model that allows to evaluate the potential of decentralized routing in this context. We have found all Nash equilibria for problems with thousands of player thanks to the Constraint Game solver Conga. This is the first time that such large instances are solved to optimality by a general-purpose Game Theory solver.

Acknowledgements We thanks Nicolas Huin for our long discussions about column generation and how to build an efficient model.

References

1. Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. *SIAM J. Comput.* 42(1), 160–177 (2013), <https://doi.org/10.1137/070702370>
2. Azzouni, A., Boutaba, R., Pujolle, G.: Neuroute: Predictive dynamic routing for software-defined networks. In: 13th International Conference on Network and Service Management, CNSM 2017, Tokyo, Japan, November 26–30, 2017. pp. 1–6. IEEE Computer Society (2017), <https://doi.org/10.23919/CNSM.2017.8256059>
3. Barnhart, C., Hane, C.A., Vance, P.H.: Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research* 48(2), 318–326 (2000), <https://doi.org/10.1287/opre.48.2.318.12378>
4. Bisschop, J.: AIMMS optimization modeling. Lulu. com (2006)
5. Braess, D., Nagurney, A., Wakolbinger, T.: On a paradox of traffic planning. *Transportation Science* 39(4), 446–450 (2005), <https://doi.org/10.1287/trsc.1050.0127>
6. Capone, A., Cascone, C., Nguyen, A.Q.T., Sansò, B.: Detour planning for fast and reliable failure recovery in SDN with openstate. In: 11th International Conference on the Design of Reliable Communication Networks, DRCN 2015, Kansas City, MO, USA, March 24–27, 2015. pp. 25–32. IEEE (2015), <https://doi.org/10.1109/DRCN.2015.7148981>
7. Capone, A., Cascone, C., Nguyen, A.Q., Sansò, B.: Detour planning for fast and reliable failure recovery in sdn with openstate. In: 2015 11th International Conference on the Design of Reliable Communication Networks (DRCN). pp. 25–32. IEEE (2015)
8. Chabert, G., al: Ibex An interval based EXplorer (2009), <http://www.ibex-lib.org>
9. Chabert, G., Jaulin, L.: Contractor programming. *Artificial Intelligence* 173(11), 1079–1100 (2009)
10. Chabrier, A., Danna, E., Pape, C.L., Perron, L.: Solving a network design problem. *Annals OR* 130(1–4), 217–239 (2004), <https://doi.org/10.1023/B:ANOR.0000032577.81139.84>
11. Correa, J.R., Schulz, A.S., Moses, N.E.S.: Selfish routing in capacitated networks. *Math. Oper. Res.* 29(4), 961–976 (2004), <https://doi.org/10.1287/moor.1040.0098>
12. Correa, J.R., Schulz, A.S., Moses, N.E.S.: Fast, fair, and efficient flows in networks. *Operations Research* 55(2), 215–225 (2007), <https://doi.org/10.1287/opre.1070.0383>
13. CPLEX, I.I.: 12.6. CPLEX Users Manual (2014)
14. Even, S., Itai, A., Shamir, A.: On the complexity of time table and multi-commodity flow problems. In: Proceedings of the 16th Annual Symposium on Foundations of Computer Science. pp. 184–193. SPCS '75, IEEE Computer Society, Washington, DC, USA (1975), <http://dx.doi.org/10.1109/SPCS.1975.21>

15. Fudenberg, D., Tirole, J.: *Game Theory*. The MIT Press (1991)
16. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)
17. Gottlob, G., Greco, G., Scarcello, F.: Pure Nash Equilibria: Hard and Easy Games. *J. Artif. Intell. Res. (JAIR)* 24, 357–406 (2005)
18. Harks, T., Heinz, S., Pfetsch, M.E.: Competitive online multicommodity routing. In: Erlebach, T., Kaklamanis, C. (eds.) *Approximation and Online Algorithms*, 4th International Workshop, WAOA 2006, Zurich, Switzerland, September 14–15, 2006, Revised Papers. *Lecture Notes in Computer Science*, vol. 4368, pp. 240–252. Springer (2006), https://doi.org/10.1007/11970125_19
19. Hayrapetyan, A., Tardos, É., Wexler, T.: A network pricing game for selfish traffic. *Distributed Computing* 19(4), 255–266 (2007), <https://doi.org/10.1007/s00446-006-0020-y>
20. Hnich, B., Prestwich, S.D., Selensky, E., Smith, B.M.: Constraint models for the covering test problem. *Constraints* 11(2-3), 199–219 (2006), <https://doi.org/10.1007/s10601-006-7094-9>
21. Huin, N., Jaumard, B., Giroire, F.: Optimal network service chain provisioning. *IEEE/ACM Trans. Netw.* 26(3), 1320–1333 (2018), <http://doi.ieeecomputersociety.org/10.1109/TNET.2018.2833815>
22. Huin, N., Tomassilli, A., Giroire, F., Jaumard, B.: Energy-efficient service function chain provisioning. *Electronic Notes in Discrete Mathematics* 64, 265–274 (2018), <https://doi.org/10.1016/j.endm.2018.02.001>
23. Key, P.B., McAuley, D.: Differential qos and pricing in networks: Where flow control meets game theory. *IEE Proceedings - Software* 146(1), 39–43 (1999), <https://doi.org/10.1049/ip-sen:19990154>
24. Layeghy, S., Pakzad, F., Portmann, M.: SCOR: constraint programming-based northbound interface for SDN. In: *26th International Telecommunication Networks and Applications Conference, ITNAC 2016*, Dunedin, New Zealand, December 7–9, 2016, pp. 83–88. IEEE (2016), <http://doi.ieeecomputersociety.org/10.1109/ATNAC.2016.7878788>
25. Leguay, J., Draief, M., Chouvardas, S., Paris, S., Paschos, G.S., Maggi, L., Qi, M.: Online and global network optimization: Towards the next-generation of routing platforms. *CoRR abs/1602.01629* (2016), <http://arxiv.org/abs/1602.01629>
26. Lissner, A., Mahey, P.: Multicommodity flow problems and decomposition in telecommunications networks. In: Resende, M.G.C., Pardalos, P.M. (eds.) *Handbook of Optimization in Telecommunications*, pp. 241–267. Springer (2006), https://doi.org/10.1007/978-0-387-30165-5_10
27. Liu, T., Callegati, F., Cerroni, W., Contoli, C., Gabbriellini, M., Giallorenzo, S.: Constraint programming for flexible service function chaining deployment. *CoRR abs/1812.05534* (2018), <http://arxiv.org/abs/1812.05534>
28. Mendiola, A., Astorga, J., Jacob, E., Higuero, M.: A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys and Tutorials* 19(2), 918–953 (2017), <https://doi.org/10.1109/COMST.2016.2633579>
29. Nash, J.: Non-cooperative Games. *Annals of Mathematics* 54(2), 286–295 (1951)
30. Nguyen, T., Lallouet, A.: A Complete Solver for Constraint Games. In: O’Sullivan, B. (ed.) *CP 2014*, Lyon, France, September 8–12, 2014. *LNCS*, vol. 8656, pp. 58–74. Springer (2014), http://dx.doi.org/10.1007/978-3-319-10428-7_8
31. Nie, C., Leung, H.: A survey of combinatorial testing. *ACM Comput. Surv.* 43(2), 11:1–11:29 (Feb 2011), <http://doi.acm.org/10.1145/1883612.1883618>
32. Orda, A., Rom, R., Shimkin, N.: Competitive routing in multi-user communication networks. In: *Proceedings IEEE INFOCOM ’93, The Conference on Computer Communications, Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking: Foundation for the Future*, San Francisco, CA, USA, March 28 - April 1, 1993, pp. 964–971. IEEE (1993), <https://doi.org/10.1109/INFCOM.1993.253270>
33. Orłowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0 – Survivable Network Design Library. In: *Proceedings of the 3rd International Network Optimization Conference (INOC 2007)*, Spa, Belgium (April 2007), <http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz>, <http://sndlib.zib.de>, extended version accepted in *Networks*, 2009.
34. Osseiran, A., Braun, V., Hidekazu, T., Marsch, P., Schotten, H., Tullberg, H., Uusitalo, M.A., Schellman, M.: The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions. In: *Vehicular Technology Conference (VTC Spring)*, 2013 IEEE 77th. pp. 1–5. IEEE (2013)

35. Palmieri, A., Lallouet, A.: Constraint games revisited. In: Sierra, C. (ed.) *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, Melbourne, Australia, August 19-25, 2017. pp. 729–735. *ijcai.org* (2017), <https://doi.org/10.24963/ijcai.2017/101>
36. Pape, C.L., Perron, L., Régim, J., Shaw, P.: Robust and parallel solving of a network design problem. In: Hentenryck, P.V. (ed.) *Principles and Practice of Constraint Programming - CP 2002*, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, *Proceedings. LNCS*, vol. 2470, pp. 633–648. Springer (2002), https://doi.org/10.1007/3-540-46135-3_42
37. Paris, S., Destounis, A., Maggi, L., Paschos, G.S., Leguay, J.: Controlling flow reconfigurations in SDN. In: *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016*, San Francisco, CA, USA, April 10-14, 2016. pp. 1–9. IEEE (2016), <https://doi.org/10.1109/INFOCOM.2016.7524330>
38. Pham, Q., Deville, Y.: Solving the longest simple path problem with constraint-based techniques. In: Beldiceanu, N., Jussien, N., Pinson, E. (eds.) *Integration of Constraint Programming, Artificial Intelligence and Operations Research, CPAIOR 2012*, Nantes, France, May 28 - June 1, 2012. *LNCS*, vol. 7298, pp. 292–306. Springer (2012), https://doi.org/10.1007/978-3-642-29828-8_19
39. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2017), <http://www.choco-solver.org>
40. Puri, A., Tripakis, S.: Algorithms for the multi-constrained routing problem. In: Penttonen, M., Schmidt, E.M. (eds.) *Algorithm Theory - SWAT 2002*, 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002 *Proceedings. Lecture Notes in Computer Science*, vol. 2368, pp. 338–347. Springer (2002), https://doi.org/10.1007/3-540-45471-3_35
41. Qin, Q., Poularakis, K., Iosifidis, G., Tassioulas, L.: SDN controller placement at the edge: Optimizing delay and overheads. In: *2018 IEEE Conference on Computer Communications, INFOCOM 2018*, Honolulu, HI, USA, April 16-19, 2018. pp. 684–692. IEEE (2018), <https://doi.org/10.1109/INFOCOM.2018.8485963>
42. Quang, P.T.A., Sanner, J.M., Morin, C., Hadjadj-Aoul, Y.: Multi-objective multi-constrained qos routing in large-scale networks: A genetic algorithm approach. In: *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. pp. 55–60. IEEE (2018)
43. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) *Principles and Practice of Constraint Programming - CP 2004*, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, *Proceedings. Lecture Notes in Computer Science*, vol. 3258, pp. 557–571. Springer (2004), https://doi.org/10.1007/978-3-540-30201-8_41
44. Rosen, J.B.: Existence and Uniqueness of Equilibrium Points for Concave n -Person Games. *Econometrica* 33(3), 520–534 (July 1965)
45. Roughgarden, T.: *Routing Games*, chap. 18, pp. 461–486. *Algorithmic game theory*, Cambridge University Press (2007)
46. Sanner, J.M., Hadjadj-Aoul, Y., Ouzzif, M., Rubino, G.: An evolutionary controllers' placement algorithm for reliable sdn networks. In: *2017 13th International Conference on Network and Service Management (CNSM)*. pp. 1–6. IEEE (2017)
47. Sellmann, M., Gellermann, T., Wright, R.: Cost-based filtering for shorter path constraints. *Constraints* 12(2), 207–238 (2007), <https://doi.org/10.1007/s10601-006-9006-4>
48. Shoham, Y., Leyton-Brown, K.: *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press (2009)
49. Song, S., Lee, J., Son, K., Jung, H., Lee, J.: A congestion avoidance algorithm in SDN environment. In: *2016 International Conference on Information Networking, ICOIN 2016*, Kota Kinabalu, Malaysia, January 13-15, 2016. pp. 420–423. IEEE Computer Society (2016), <https://doi.org/10.1109/ICOIN.2016.7427148>
50. Szymanski, T.H.: Max-flow min-cost routing in a future-internet with improved qos guarantees. *IEEE Trans. Communications* 61(4), 1485–1497 (2013), <https://doi.org/10.1109/TCOMM.2013.020713.110882>
51. Tajiki, M., Akbari, B., Shojafar, M., Mokari, N.: Joint qos and congestion control based on traffic prediction in sdn. *Applied Sciences* 7(12), 1265 (2017)
52. Vassilaras, S., Gkatzikis, L., Liakopoulos, N., Stiakogiannakis, I., Qi, M., Shi, L., Liu, L., Debbah, M., Paschos, G.: The algorithmic aspects of network slicing. *IEEE Communications Magazine* (2017)
53. Von Neumann, J., Morgenstern, O.: *Theory of Games and Economic Behavior*. Princeton University Press (1944), <http://jmvidal.cse.sc.edu/library/neumann44a.pdf>

-
54. Yu, L., Lei, Y., Kacker, R., Kuhn, D.R.: ACTS: A combinatorial test generation tool. In: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013. pp. 370–375. IEEE Computer Society (2013), <https://doi.org/10.1109/ICST.2013.52>