Constraint Games for Stable and Optimal Allocation of Demands in SDN [*]

Anthony Palmieri[1,2] , **Arnaud Lallouet**[1,2] and **Luc Pons**[1]
[1]Huawei Technologies Ltd, French Research Center
[2]Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France
{anthony.palmieri, arnaud.lallouet, luc.pons}@huawei.com

Abstract

Software Defined Networking (or SDN) allows to apply a centralized control over a network of computers in order to provide better global throughput. One of the problem to solve is the multi-commodity flow routing where a set of demands (or commodities) 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 selfish routing with Constraint Games. Selfish routing reaches a Nash equilibrium and 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 with hundreds or thousands players and we show that for this problem the worst selfish routing is often close to the optimal centralized solution.

1 Introduction

With the internet of things, all kinds of devices are going to communicate, from washing machines, light bulbs to autonomous cars. The amount of data transfer increases with the rise of 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 [Vassilaras *et al.*, 2017]. SDN (see Figure 1) allows to apply a centralized control over a network of computers in order to increase the overall performance. A full SDN controller is a nice source for many optimization problems [Leguay *et al.*, 2016] including online ones.

In this paper, we consider the independent routing of multiple demands across a network, also called *multi-commodity flow routing problem*. Each demand has a source and a destination and each link has a capacity. The overall goal is to assign a route to each demand that minimizes the global cost of routing. This problem has been studied since a long time [Even *et al.*, 1975], mostly with Linear Programming or with other incomplete methods [Azzouni *et al.*, 2017]. A survey can be found in [Mendiola *et al.*, 2017]. 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 [Puri and Tripakis, 2002]. This justifies the modern interest in unsplittable routing of demands. Since capacity constrained shortest path is already NP-complete, 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. But we do consider a congestion model that increases the cost of a link according to the traffic routed and we propose a Constraint Programming model to solve it optimally. For this, we use a natural heuristic based on increasing paths and we introduce a lower bound that can be used in branch and bound efficiently.

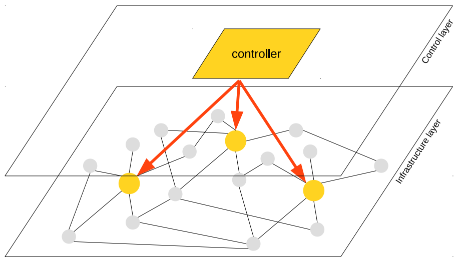

Figure 1: Software Defined Networking

Small 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 which is a Nash equilibrium will be preferred as it ensures the user that the quality of service he gets cannot be improved by any selfish move. For this, we make use of the recently introduced Constraint Games framework [Palmieri and Lallouet, 2017] to compute routings which are at Nash equilibrium. In addition, we derive exact and approximate bounds for Price of Anarchy [Fudenberg and Tirole, 1991] that allow to evaluate the loss of efficiency of decen-

[*] This paper was published under the same title at the CP'2018 conference. An extended version will appear in the Constraints journal.

tralized algorithms. In the benchmarks, we show games with hundreds or even thousands of players solved up to optimality. These results have been obtained with ConGa, an extension of the Choco solver for Constraint Games [Palmieri and Lallouet, 2017] and show that a practical use of game theory is now possible at industrial scale.

Path heuristics have been introduced in CP in [Pape *et al.*, 2002] in the context of network design, which includes as a subproblem the routing problem we focus on (but without taking congestion into consideration). More recently, in [Layeghy *et al.*, 2016], the authors model a SDN problem with CP. Our work differs in multiple points. We consider only single-path routing, we take congestion into account, and more important, we take into account the quality of service through the computation of Nash equilibria. From the game theory point of view, the closer class of game studied are the congestion games [Rosenthal, 1973]. This class of games tries to study the impact of the congestion over a network. However this work is different of ours because it never considers hard constraints in the model.

2 Multicommodity Path Routing

A multicommodity path routing problem (MCPRP) consists in a graph defining a network and a set of commodities (flow demands) to be routed on this graph. We consider here the simple 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 every time a link is taken and is reflected by a congestion cost which helps to ensure an homogeneous distribution of the routes. The overall objective is to minimize the sum of costs of the routed demands, and in case of games, while preserving optimality for each player.

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 $cost(e) \in \mathcal{R}^+$ and a capacity $cap(e) \in \mathcal{R}^+$. 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 $bw(d) \in \mathcal{R}^+$ 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 $i \neq j \to v_i \neq v_j$. 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:

1. (correctness) $\forall d \in D$, $src(path(d)) = src(d)$ and $dst(path(d)) = dst(d)$
2. (admissibility) $\forall e \in E$,

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

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 will be close to saturation. For this, we define the load of an edge e to be:

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

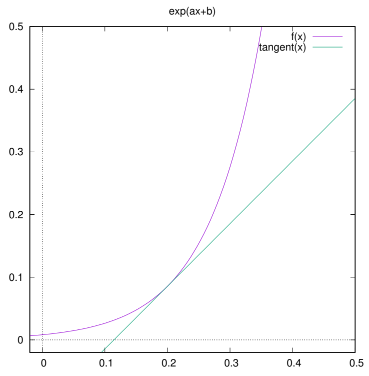

Figure 2: A plot of the congestion function for $Mxc = 1000$ and $cong'(0.2) = 1$

The congestion model we use has an exponential increase of the form $cong(x) = e^{ax+b}$ where x is the load of the arc. In order to choose the parameters a and b, we pose some conditions 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 Mxc. We then have the equation $e^{a+b} = Mxc$. 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 $Mxc = 1000$ and $cong'(0.2) = 1$.

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

$$cost(d) = bw(d) * \sum_{e \in path(d)} (cost(e) + cong(e)) \qquad (1)$$

Then the cost of the whole problem is given by:

$$cost(P) \quad = \quad \sum_{d \in D} cost(d) \qquad (2)$$

Note that this function is strictly monotonic, resulting that each addition of demand increases the edge cost. The problem of finding a Nash equilibrium adds to this the constraint that there should not exist a better path for each demand provided that the other paths remain identical. In other words, path should also be optimal for each demand.

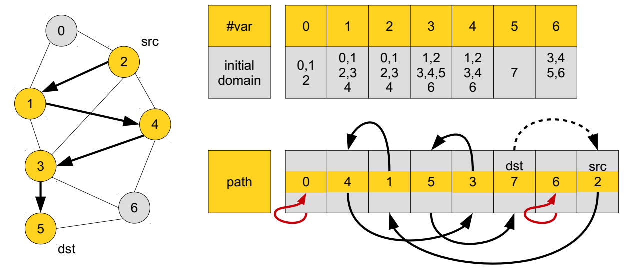

Figure 3: Encoding of a path

3 Constraint Model

A path is represented by an array $path$ of $|V|$ variables, each one being assigned to the next node along the path. The initial domain of a variable associated to a node v is given by the 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 allow any vertex to be starting the path (see Figure 3 for the encoding of the path $(2, 1, 4, 3, 5)$ from $src = 2$ to $dst = 5$).

Path-oriented problems are particularly sensitive to heuristics, and not surprisingly, a standard dynamic CP heuristic (denoted by `CP` in this paper) would be of weak efficiency for this type of problem. Indeed, it is likely that this heuristic will label any node in the path without knowing if it could be linked to the source or destination. Therefore, we have chosen to label path variables in order of *increasing* path length. Note that this is a partial variable heuristic 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 heuristics 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 bandwidth up to its completion. Then we have defined two strategies based on conflicts which analyze the current solution once a first solution has been found by `MB` or the situation after a fail. For each demand and each link, we compute the marginal cost (with congestion) induced by the presence of the demand on this very link. 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 highest score and develop its path up to the destination. The second one, called `CO1` (for *Conflict_1_Step*), also chooses the demand of highest score but only develops one step in the path before reconsidering the situation. In `CO1`, the conflicts are stored for each path variable for each demand and score are only computed for the uninstantiated variables.

For each variable, the value heuristics determines the direction the path will take. Since the goal is to find the best path for each demand, it would be inefficient to start the path in a wrong direction. In order to start with the most promising path, we maintain at each node of the search tree the shortest path to destination in the residual network for each demand in isolation. We call this heuristic `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. 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 [Chabrier *et al.*, 2004] but with specific path variables.

Branch and bound is a common technique used in constraint optimization. However, CP solvers offer a restricted and uninformed version of branch and bound: when minimizing the variable $ProblemCost$ and after having found a solution of value A, it simply adds to the end of the search the constraint $ProblemCost < A$. 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 heuristic. Congestion is taken into account only for the already assigned nodes in demand paths and the current demand to estimate. It means that two demands whose future path would take the same link do not create congestion in their future paths. 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 branch & bound `CP` and the one which uses the bound provided by the shortest path `SP`.

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 [Palmieri and Lallouet, 2017]. 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. 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 1. The social welfare function represents the global cost to be minimized as defined in equation 2. 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). To compute this, 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 asymetric in term of the branch and bound we can implement. For PoS, the problem is still a minimization. Thus

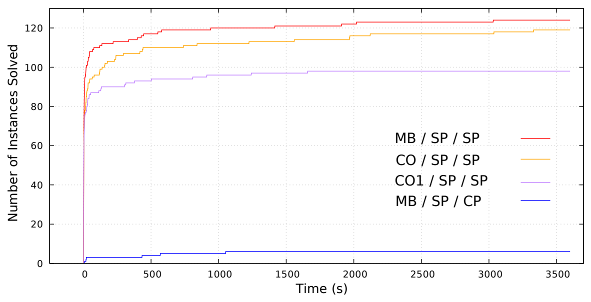

Figure 4: Comparison of different heuristics on synthetic benchmarks

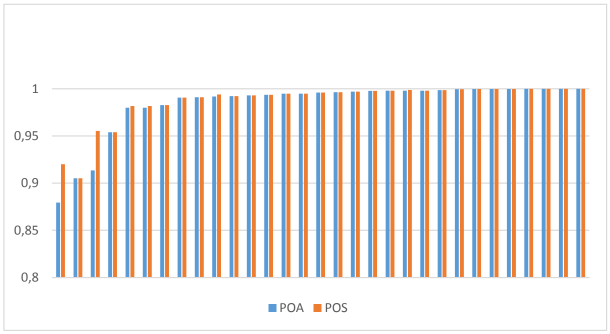

Figure 5: Price of Anarchy and Price of Stability for small synthetic instances

we can use the same branch and bound as the one we use in the centralized version. 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. This is why we propose to approximate the longest path by a Maximum Spanning Tree in the residual graph, computed by considering the upper bound value of the cost of the edges. It is clear that the cost of the tree is always greater than the cost of the longest path.

4 Benchmarks

We have tested our framework on a library of instances called SNDlib [Orlowski *et al.*, 2007] and a personal problem generator that is able to generate instances close to real ones. 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 and the Constraint Game model with a timeout fixed at 1 hour.

For the synthetic benchmarks, we have displayed the results in Figure 4. As a preliminary test, we have tried the pure CP heuristic based on impact [Refalo, 2004] 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.

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 4 show how many instances are solved in a specific delay. Clearly, the MB/SP/SP heuristics outperforms the other ones. This is not surprising compared with the CP-style B&B, but it shows that a more dynamic heuristics based on conflicts is not effective on this type of problems.

Then 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 branch and bound by giving a better first solution which is also an equilibrium.

The run-time comparison of the different strategies on the SNDlib instances shows that the improved branch and bound allows to solve many instances up to optimality. On the Constraint Game side, it is interesting to see that games of unprecedented size (up to 1482 players) can be solved up to optimality by Conga [Palmieri and Lallouet, 2017]. Interestingly, we have observed that IBR slightly degrades the time of computation, 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.

We report the results for the computation of PoA and PoS for small synthetic instances in Figure 5. 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 by the MST 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.

5 Conclusion

This paper includes two practical contributions. First we have modeled and solved efficiently the unsplittable multi-commodity flow routing problem with congestion in Constraint Programming. We have provided an accurate branch and bound technique that allows to solve real-world size instances up to optimality. Our second contribution is a Constraint Game model that allows to evaluate the potential of decentralized routing in this context. Decentralized routing is crucial for the online version of the problem where demands come as a flow. 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 up to optimality by a general-purpose Game Theory solver.

References

[Azzouni *et al.*, 2017] Abdelhadi Azzouni, Raouf Boutaba, and Guy Pujolle. 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*, pages 1–6. IEEE Computer Society, 2017.

[Chabrier *et al.*, 2004] Alain Chabrier, Emilie Danna, Claude Le Pape, and Laurent Perron. Solving a network design problem. *Annals OR*, 130(1-4):217–239, 2004.

[Even *et al.*, 1975] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, SFCS '75, pages 184–193, Washington, DC, USA, 1975. IEEE Computer Society.

[Fudenberg and Tirole, 1991] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, 1991.

[Layeghy *et al.*, 2016] Siamak Layeghy, Farzaneh Pakzad, and Marius Portmann. 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*, pages 83–88. IEEE, 2016.

[Leguay *et al.*, 2016] Jérémie Leguay, Moez Draief, Symeon Chouvardas, Stefano Paris, Georgios S. Paschos, Lorenzo Maggi, and Meiyu Qi. Online and global network optimization: Towards the next-generation of routing platforms. *CoRR*, abs/1602.01629, 2016.

[Mendiola *et al.*, 2017] Alaitz Mendiola, Jasone Astorga, Eduardo Jacob, and Marivi Higuero. A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys and Tutorials*, 19(2):918–953, 2017.

[Orlowski *et al.*, 2007] Sebastian Orlowski, Michał Pióro, A. Tomaszewski, and Roland Wessäly. SNDlib 1.0 – Survivable Network Design Library. In *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007. http://sndlib.zib.de, extended version accepted in Networks, 2009.

[Palmieri and Lallouet, 2017] Anthony Palmieri and Arnaud Lallouet. Constraint games revisited. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 729–735. ijcai.org, 2017.

[Pape *et al.*, 2002] Claude Le Pape, Laurent Perron, Jean-Charles Régin, and Paul Shaw. Robust and parallel solving of a network design problem. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *LNCS*, pages 633–648. Springer, 2002.

[Puri and Tripakis, 2002] Anuj Puri and Stavros Tripakis. Algorithms for the multi-constrained routing problem. In Martti Penttonen and Erik Meineche Schmidt, editors, *Algorithm Theory - SWAT 2002, 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002 Proceedings*, volume 2368 of *Lecture Notes in Computer Science*, pages 338–347. Springer, 2002.

[Refalo, 2004] Philippe Refalo. Impact-based search strategies for constraint programming. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2004.

[Rosenthal, 1973] Robert W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[Vassilaras *et al.*, 2017] Spyridon Vassilaras, Lazaros Gkatzikis, Nikolaos Liakopoulos, Ioannis N. Stiakogiannakis, Meiyu Qi, Lei Shi, Liu Liu, Merouane Debbah, and Giorgios S. Paschos. The algorithmic aspects of network slicing. *IEEE Communications Magazine*, 2017.