VNFs reconfiguration in 5G networks

Kristina Kumbria, Denis Demko, Dritan Nace, Artur Tomaszewski, Mustapha Bouhtou, Hanane Biallach

VNFs reconfiguration in 5G networks

Kristina Kumbria
Heudiasyc,
Université de Technologie de Compiègne
Compiègne, France
kristina.kumbria@etu.utc.fr

Denis Demko
Heudiasyc, UTC
Polytechnic University of Tirana
Tirana, Albania
denis.demko@fti.edu.al

Dritan Nace
Heudiasyc,
Université de Technologie de Compiègne
Compiègne, France
dritan.nace@hds.utc.fr

Artur Tomaszewski
Warsaw University of Technology
Warsaw, Poland
artur.tomaszewski@pw.edu.pl

Mustapha Bouhtou
Orange Labs
Châtillon, France
mustapha.bouhtou@orange.com

Hanane Biallach
Orange Labs - Heudiasyc
Châtillon, France
hanane.biallach@hds.utc.fr

***Abstract*—We consider a problem of allocating Virtual Network Functions (VNFs) in 5G networks, treating it as a multiple multidimensional knapsacks reconfiguration problem where items represent VNFs and knapsacks represent servers on which VNFs are able to run. Given an initial and final assignments of items to knapsacks, the task is to define a sequence of steps for moving items from a source knapsack and/or to a destination knapsack. The objective is to minimize the total number of steps in which items are not in the knapsack, which reflects the total time that VNFs are not operational. We derive a non-compact formulation and a new compact formulation of the problem, and evaluate their performance with numerical experiments.**

***Index Terms*—resource, allocation, migration, knapsack, VNFs**

I. Introduction

Dynamic allocation of resources and dynamic adjustment of control parameters are crucial for the operation of wireless network systems. Allocating spectrum and bandwidth, changing modulation and coding schemes let the wireless network adapt to varying network and traffic conditions and use network resources efficiently while providing the required Quality of Service (QoS). In 5G cellular networks, network functions are also allocated dynamically. In view of network programmability and function virtualization, network functions are implemented as Virtual Network Functions (VNFs), which are software components that perform user, transport, and management tasks, and which are dynamically assigned to and executed on pools of servers constituting a shared computation infrastructure.

In this study, we consider the problem of ensuring continuity of VNF operation. As network and traffic conditions and user needs vary over time, so might the required characteristics of VNFs. Regarding servers, the allocation of VNFs to servers might sometimes need to be changed, either periodically or on demand, in order to satisfy capacity constraints relating to individual VNFs or to server characteristics.

This paper looks at the VNF migration problem. We assume that the initial and final states that define the assignment of the VNFs to their source and destination servers are known. The goal is to define a sequence of steps in the migration from the initial to the final state (see Figure 1). At each step, VNFs can only be moved from their source server and/or to their destination server. The objective is to minimize the total time when VNFs are not assigned to a server and therefore must be suspended. Another version of this problem is where the destination state is not fixed, so defining the final assignment of the VNFs to servers becomes an additional part of the problem. All this is encapsulated in what we call the Multiple Multidimensional Knapsacks Reconfiguration (MMKR) problem, which is the focus of our paper; the multiple knapsacks correspond to servers, and the multiple dimensions are different capacity-related characteristics of the VNFs and the server.

Fig. 1. MMKR – VNF reconfiguration problem

The contribution of the paper consists in proposing a new method of solving the MMKR problem, based on a non-compact Integer Linear Programming (ILP) problem formulation and Column Generation, together with an alternative new compact ILP formulation. We illustrate the resulting effectiveness of solving the problem with a series of numerical experiments, comparing the results to the results of applying the compact ILP problem model reported in [1].

The paper is structured as follows. Section II presents the MMKR problem together with a short review of the related literature. Section III first recalls a known compact ILP formulation of the MMKR problem and then derives its non-compact ILP formulation and the corresponding Column Generation algorithm, as well as a new, alternative compact ILP formulation. Section IV presents the results of our numerical experiments used to assess the effectiveness of the two proposed formulations. Section V contains concluding remarks and identifies further research perspectives.

II. MMKR Problem

The KNAPSACK problem in its basic form has a prominent place in the study of Integer Programming (IP) models with binary variables. The problem consists in packing a given set of items with given values and sizes (such as weights or volumes) into a container with a given maximum capacity C. With the total size of the items potentially exceeding the capacity of the container, usually, it is not possible to pack all of them. The problem is therefore to select a subset of items with the greatest possible total value to place in the container. The ILP formulation of the problem uses one binary decision variable per item to indicate the items that are selected.

A variant of the standard KNAPSACK problem is the MULTIPLE KNAPSACKS problem. Instead of a single knapsack, here there are multiple knapsacks, where knapsack k ($k = 1 \dots K$, K being the number of knapsacks) has capacity C_k. Not only must be selected the items to pack, but it must also be decided which knapsack each item should be placed in.

Another often-studied version of the standard KNAPSACK problem is the MULTIDIMENSIONAL KNAPSACK problem. In this variant of the problem, the set of items selected for packing into the knapsack should satisfy not one but several capacity constraints corresponding to individual capacity dimensions. And putting together multiple dimensions and multiple knapsacks gives rise to the MULTIPLE MULTIDIMENSIONAL KNAPSACKS problem, a problem that has so far been very little addressed in the literature, but which appears to be NP-hard and computationally challenging.

Finally, a reconfiguration aspect can be introduced into the KNAPSACK problem. The idea behind reconfiguration is to start from a given initial assignment of items to knapsacks and to sequentially move items between knapsacks so as to reach a given final assignment. Including reconfiguration brings out the scheduling aspect of the problem which, in conjunction with multiple multidimensional knapsacks, results in the MMKR problem that is our concern here. The objective will be to minimize the interruption time. It is a novel version of the problem, and it turns out to be considerably more difficult.

Arguably, versions of the problem examined in this work have already appeared in other contexts linked to computer networking. Sirdey et al. [2] looked at a process move problem that arises in relation to the operability of a certain class of high-availability real-time distributed systems. Given an initial and a final system state defining which processes are assigned to which processors in a distributed system, the goal is to find a minimally disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system is in the final state; at no step can the capacity of any processor be exceeded. Thinking of each processor as a knapsack and of each process as an item, this problem has a lot in common with the MMKR problem. The main difference lies in the technical context: in our 5G VNF reconfiguration problem migrations are realized according to rules (known as hot and cold migration of VNFs) that impose specific constraints upon feasible migration sequences.

Solano and Pióro [3] studied a problem of lightpath reconfiguration in optical transport networks. Lightpath reconfiguration is a networking task that is performed in order to improve resource utilization and limit network congestion. The problem consists in finding an optimal transition from an initial set of currently operational lightpaths to a new set of lightpaths guaranteeing better network performance. The problem becomes nontrivial when establishing the new set of lightpaths requires the release of resources held by the currently operational lightpaths while ensuring continuity of traffic flows (an operational lightpath cannot be removed before the corresponding new lightpath is set up). The reconfiguration scheduling seeks to make the maximum number of simultaneously disrupted connections at any point during the process as small as possible [3]. Although the application context is different from that of the MMKR problem, the formulations of the two problems have a lot in common. The existing and the new lightpath connections can be seen as items migrating over network links which, being bundles of optical wavelengths, may be considered as knapsacks, insofar as they provide the resources for the connections.

There has also been a lot of work done on problems that overlap with the MMKR problem. [4] proposes a resource allocation and management mechanism for 5G networks using Multi-access Edge Computing (MEC) technology. A simple mathematical model of the MEC resource (re)allocation mechanism is proposed to meet the requirements of the user.

[5] considers a control problem in the reconfiguration of photovoltaic arrays that involves changing the connections in the solar panel equipped with the dynamic switching matrix. The proposed formulation of the problem is based on the well-known subset sum problem, which is a special case of the KNAPSACK problem. The solution method uses a dynamic programming algorithm that is capable of computing an optimum reconfiguration (in marked contrast to [6], where a solution method based on fuzzy logic under partial shading conditions is put forward as a solution to the same problem). The characteristic feature of this problem is that the final configuration of the array is not given.

[7] considers centralized spectrum allocation in cognitive radio networks as a MULTIPLE MULTIDIMENSIONAL KNAPSACKS problem. The formulation of the problem models the behaviour of the Centralized Coordinator Node (CCN) that shares spectrum availability information with a set of cognitive users. Each primary user acts as a two-dimensional knapsack,

with bandwidth and temperature interference as the limiting resources. CCN has to assign the cognitive users, considered as items, to primary users in such a way that the overall traffic throughput is maximized while the resource constraints are satisfied. The proposed formulation is similar to our problem formulation, but it is limited to the computation of a static assignment of items to knapsacks and does not include the scheduling aspect.

III. Solution methods

In this section we discuss exact methods of solving the MMKR problem. These methods are essentially based on Integer Linear Programming. Table I lists the notation used.

A. First compact formulation

We start by recalling a compact ILP problem model reported in [1]. We use the following variables. For each $i \in I$, x_i is a continuous positive variable denoting the interruption time of item i. For all $i \in I$, $t \in T$, y_i^t is a binary variable that equals 1 if, and only if, item i is interrupted at its source at time t, and z_i^t is a binary variable that equals 1 if, and only if, item i is placed at its destination at time t. For all $k \in \mathcal{K}$, $t \in T$, c_k^t is a continuous positive variable denoting the residual capacity of knapsack k at time t. The MMKR problem can then be written as:

$$\min \sum_{i \in \mathcal{I}} w(i) x_i \tag{1a}$$

$$\sum_{i \in \mathcal{O}(k)} w(i) + c_k^1 = C(k) \quad \forall k \tag{1b}$$

$$\sum_{i \in \mathcal{O}(k)} w(i) y_i^t - \sum_{i \in \mathcal{D}(k)} w(i) z_i^t + c_k^t = c_k^{t+1} \quad \forall k,\ \forall t < T \tag{1c}$$

$$\sum_{t \in \mathcal{T}} y_i^t = 1 \qquad \forall i \tag{1d}$$

$$\sum_{t \in \mathcal{T}} z_i^t = 1 \qquad \forall i \tag{1e}$$

$$\sum_{t \in \mathcal{T}} t z_i^t - \sum_{t \in \mathcal{T}} t y_i^t + 1 \leq x_i \qquad \forall i \tag{1f}$$

$$x_i \geq 0 \qquad \forall i \tag{1g}$$

$$y_i^t, z_i^t \in \{0,1\} \quad \forall i,\ \forall t \tag{1h}$$

$$c_k^t \geq 0 \qquad \forall k,\ \forall t. \tag{1i}$$

The objective is to minimize the weighted interruption time over all items. Constraints (1f) force the interruption time of the item to be greater than the difference between the time of placing the item at the destination and the time of interrupting at the source (increased by 1 because the VNF is not available until the next time period after transfer); we note that the nature of the objective forces this constraint to equality.

Constraints (1b) and (1c) serve to link residual capacities overtime periods for each knapsack.

By constraints (1d) and (1e) we force each item to be interrupted and transferred exactly once, which does not allow any knapsack to be used as a temporary one during migrations.

K	the number of knapsacks
$\mathcal{K}$	the set of knapsacks, $\mathcal{K} = \{1, 2, \ldots, K\}$
k	a specific knapsack
$C(k)$	the resource capacity of knapsack k
I	the number of items
$\mathcal{I}$	the set of items, $\mathcal{I} = \{1, 2, \ldots, I\}$
i	a specific item
$o(i)$	the source knapsack of item i
$d(i)$	the destination knapsack of item i
$w(i)$	the resource requirement of item i
$\mathcal{O}(k)$	the items with knapsack k as source, $\mathcal{O}(k) \subseteq \mathcal{I}$
$\mathcal{D}(k)$	the items with knapsack k as destination, $\mathcal{D}(k) \subseteq \mathcal{I}$
T	the number of time periods
$\mathcal{T}$	the set of time periods, $\mathcal{T} = \{1, 2, \ldots, T\}$
t	a specific time period

TABLE I: Notation

Unfortunately, formulation (1) does not scale well because of the large number of integer variables and equality conditions. Our goal in this work is to propose alternative solution methods. The following is a method based on Column Generation.

B. Non-compact formulation

To use the Column Generation method, we need an alternative non-compact problem formulation. The main challenge is deciding on the format of the columns. We are seeking to represent (over the relevant time periods) the movement of an item from the source knapsack to the destination knapsack as a path in a graph with (knapsack, time) tuples as nodes. For each knapsack and each time period there will be a specific node in the graph, and at each time period the item is to be found in at least one knapsack, and in at most two (there are two knapsacks when the migration occurs). A path in this graph therefore represents the placement of the item over all time periods. The arcs (and consequently the paths) allowed in the graph should reflect all possible item migrations. We have T time periods and $K+1$ knapsacks replicated at each period, the knapsack $(K+1)$ being a virtual knapsack of infinite capacity, containing all the interrupted items. If an arc enters a node (k, t) the corresponding item consumes resources of knapsack k at time period t. Since the number of possible paths is large, they cannot all be included in the problem formulation, and the Column Generation method deals neatly with this situation. We start with a basic feasible set of paths and then search for a new path that improves the objective function.

Each column takes the form $a = (a_{1,1}, \ldots, a_{K+1,1}, a_{1,2}, \ldots, a_{K+1,2}, \ldots, a_{1,T}, a_{2,T}, \ldots, a_{K+1,T})$, where $a_{k,t}$ is 1 if item i is placed at knapsack k at time t, and 0 otherwise.

Each of these columns represents a specific path p that some item i can use to migrate from its source to its destination knapsack. Since at the outset there will be only a small set of paths guaranteeing an initial feasible solution, we need to search for new paths (columns) in the above format to be included in the path set P_i for item i, with a view to

obtaining a better objective function value. When searching for such a column all its elements $a_{k,t}$ will be set as binary variables. In addition, each column in the Master Problem has a corresponding binary path variable y_{ip} that indicates whether path p for item i is used (y_{ip}=1) or not used (y_{ip}=0).

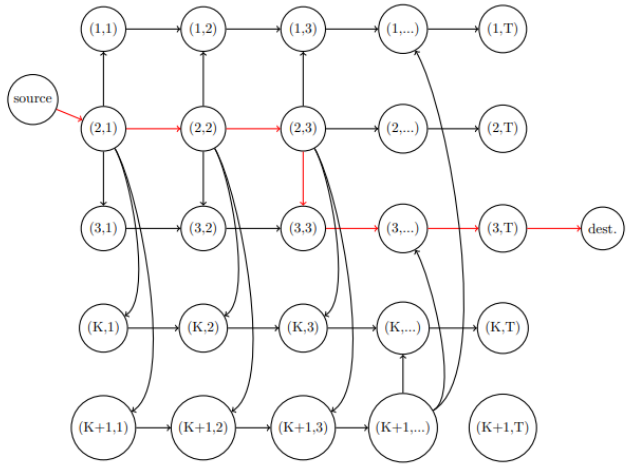

Fig. 2. Graph representation

The graph in Figure 2 shows a migration from source knapsack 2 to destination knapsack 3 at time period 3. At time period 3 resources are thus consumed in both knapsack 2 and knapsack 3. For the sake of clarity only a subset of arcs are shown just to give an idea of the allowed arcs and paths. We can move from the source knapsack to other knapsacks during each time period except the last one. If this move is to the destination knapsack, then the item will stay there until the end. If the move is to the interruption knapsack, then we are forced to migrate back to the destination knapsack before the last time period, with vertical moves coming out from knapsack $(K+1)$.

1) Master Problem: Now that we have a better theoretical grasp of the context of the problem, we are able to put forward a corresponding non-compact Linear Programming (LP) formulation as the first step in our column generation algorithm. For sake of simplicity, the one-dimensional case is assumed.

$$\min \sum_{t\in\mathcal{T}} \sum_{i:(K+1,t)\in p, p\in\mathcal{P}_i} w(i)y_{ip} \tag{2a}$$

$$\sum_{i:(k,t)\in p, p\in\mathcal{P}_i} w(i)y_{ip} \le C(k) \quad \forall k,\ \forall t \quad (\lambda_{kt}) \tag{2b}$$

$$\sum_{p\in\mathcal{P}_i} y_{ip} = 1 \quad \forall i \quad (\gamma_i) \tag{2c}$$

$$y_{ip} \ge 0. \tag{2d}$$

The objective function aims at minimizing the overall path weights going through the interruption knapsack summed over all time periods, which is equivalent to minimizing the total interruption cost. For a specific time period t the second sum iterates over all items that have $(K+1,t)$ in a path p belonging to their pathset $\mathcal{P}_i$.

2) Pricing Problem: The second step in the algorithm requires defining the Pricing Problem. As mentioned above, the Pricing Problem will be an Integer Linear Programming using the dual variables found in the first step as parameters and column elements a_{kt} as variables.

$$\min\ w(i) \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} \lambda_{k,t}a_{k,t} - \gamma_i \equiv \tag{3a}$$

$$\min \sum_{k\in\mathcal{K}} \sum_{t\in\mathcal{T}} \lambda_{k,t}a_{k,t} \equiv \tag{3b}$$

$$\min \sum_{(k,t)\in p\in\mathcal{P}_i} \lambda_{k,t} \tag{3c}$$

$$\sum_{k\in\mathcal{K}} a_{k,t} \ge 1 \quad \forall t \tag{3d}$$

$$a_{k,t} \in \{0,1\} \quad \forall k,\ \forall t. \tag{3e}$$

The above model is complemented with the following constraint if no interruptions are allowed

$$\sum_{t\in\mathcal{T}} \sum_{k\in\{1,\dots,K\}} a_{k,t} \le T+1, \tag{4}$$

and with the following one if interruptions are allowed

$$\sum_{t\in\mathcal{T}} \sum_{k\in\{1,\dots,K+1\}} a_{k,t} \le T+2. \tag{5}$$

As we can see from the above, the objective function above is a graph shortest path problem, where the arc weights are denoted by dual variables $\lambda_{k,t}$. We are searching for a shortest path from source to destination knapsack in a directed graph where each incoming arc at node (k,t) has a weight $\lambda_{k,t}$.

The condition (3d) indicates that an item should be placed in at least one knapsack at each time period.

The conditions (4) and (5) restrict the movements of items between knapsacks. Only one knapsack change, denoted by a vertical move in the graph, is allowed if there are no interruptions (i.e., knapsack $(K+1)$ is not used). But, if we are searching for a path with interruption (i.e., knapsack $(K+1)$ is used), then we allow at most two vertical moves.

3) Achieving an integer solution: As the final step in the column generation algorithm, we need to obtain an integer solution to the problem.

a) Final ILP: One good way of not shifting the results too much in relation to the continuous results is to solve the final Master Problem as an Integer Linear Programming problem. In this final Master Problem we use the final set of paths found by iteratively solving the continuous version of Column Generation until optimality. The path variables are then forced to binary. This method is referred to below as Solving the Final Master Problem (*Final ILP*).

b) Post optimization: The main idea in this method is to start with an initial solution obtained by rounding and to add new paths until a possible solution is reached. This initial solution may be infeasible after rounding the variables. We therefore identify the knapsack in which the capacity constraint has been violated and the items that it contains. We then modify the paths of some of the items in the knapsack by adding interruptions in them to make the solution feasible. This is noted with *PostOpt*.

c) A mixed method: Another method involves solving the Master Problem one more time, but now with the addition of the paths established using the heuristic method. A comparison of the objective functions of the different methods shows that solving the Master Problem once again in this way gives us the lowest cost. The ILP version of the final Master Problem (after the addition of paths identified by the post optimization method) is consequently the best version, maintaining feasibility and having the lowest interruption cost, that is to say the closest to the continuous interruption cost. This method is noted as *PostOpt + Final ILP*.

C. Second compact formulation

Below we introduce a new compact ILP model of the MMKR problem. We define the following variables. For all $i \in \mathcal{I}$, $t \in \mathcal{T}$, x_i^t is a binary variable that equals 1 if, and only if, item i is at its source at time t, y_i^t is a binary variable that equals 1 if, and only if, item i is at its destination at time t, and z_i^t is a binary variable that equals 1 if, and only if, item i is interrupted at time t. Then, using the observable monotonicity of the values of variables x and y with time periods, the MMKR problem can be formulated as:

$$\min \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} w(i) z_i^t \tag{6a}$$

$$x_i^t \geq x_i^{t+1} \quad \forall i,\ \forall t < T \tag{6b}$$

$$y_i^t \geq y_i^{t-1} \quad \forall i,\ \forall t > 1 \tag{6c}$$

$$x_i^t + y_i^t + z_i^t \geq 1 \quad \forall i,\ \forall t \tag{6d}$$

$$\sum_{t\in\mathcal{T}} (x_i^t + y_i^t + z_i^t) \geq T + 1 \quad \forall i \tag{6e}$$

$$\sum_{i\in\mathcal{O}(k)} w(i) x_i^t + \sum_{i\in\mathcal{D}(k)} w(i) y_i^t \leq C(k) \quad \forall k,\ \forall t \tag{6f}$$

$$x_i^t, y_i^t, z_i^t \in \{0,1\} \quad \forall i,\ \forall t. \tag{6g}$$

Once again, the objective is to minimize the weighted interruption time over all items. While constraints (6d) require that each item is either in a knapsack or is interrupted, constraints (6e) result from the mentioned specifics of VNF migration that the VNF is not available until the next time period after transfer, and require that after the item has been moved to the destination knapsack and the item is not still in the source knapsack it must be regarded as interrupted.

IV. Numerical Results

This section is devoted to numerical results. We ran our methods on a set of realistic problem instances, using randomly generated topologies to build both acyclic and cyclic graphs. The graphs, of different sizes (small and medium), were generated randomly using NetworkX, (a Python lib), based on [8]. To solve the linear programming models we have used the PuLP library, based on [9], an open source package that allows mathematical programs to be described in Python programming language.

The nodes of the graph represents the servers, and the links represent the VNF migration. The node and link capacities were generated randomly, $(1 - 50)$ for CPU capacity and $(10 - 90)$ for RAM capacity. The type number of the VNFs was randomly generated in the range $(25 - 220)$ and the number of servers in the range $(10 - 100)$. The datasets are presented in Table II. Another element included in the tables is the number of periods (Periods) allowed for the migration. This aspect may be important in some situations where the migrations have to be realized in the shortest possible time. In theory, the maximum number of periods is bounded by the number of migrations (items), but in practice several migrations can be processed in parallel. An assessment of the Column Generation method together with its several post-optimization methods for both cyclic and acyclic graphs can be seen in Table III and Table IV, respectively. It will be remarked that the mixed method (that is PostOpt + Final ILP) performed significantly better. This method is used below for the purposes of comparison with the ILP formulations.

	Cyclic Graphs			Acyclic Graphs		
Instance	Server	Item	Periods	Server	Item	Periods
1	10	30	*I*/4	10	25	*I*
2	20	45	*I*/5	20	35	*I*/4
3	40	70	*I*/8	40	60	*I*/9
4	50	90	*I*/10	50	80	*I*/10+1
5	80	146	*I*/18	80	150	5*I*/32
6	100	220	*I*/32-1	100	200	*I*/25

TABLE II: Datasets of cyclic and acyclic graphs

Instance	Final ILP	PostOpt	PostOpt + Final ILP
1	12	5	2
2	0	4	0
3	28	34	16
4	0	17	0
5	70	63	46
6	16	33	11

TABLE III: Cyclic Graphs – Interruption cost

Instance	Final ILP	PostOpt	PostOpt + Final ILP
1	0	19	0
2	0	0	0
3	5	11	1
4	0	5	0
5	0	38	0
6	14	28	5

TABLE IV: Acyclic Graphs – Interruption cost

In the following, we report comparative numerical results for the three methods, namely column generation completed with mixed post optimization method (noted as *ColGen++*), first compact ILP, and second compact ILP. The notation TL stands for the out of time limit, the limit being set to 3600 seconds.

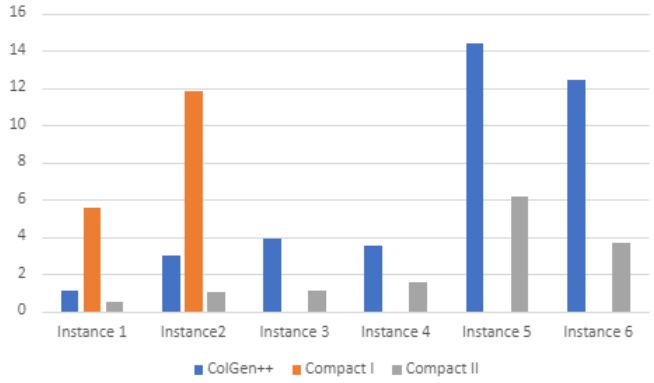

Fig. 3. Cyclic Graphs – Computation time (in seconds)

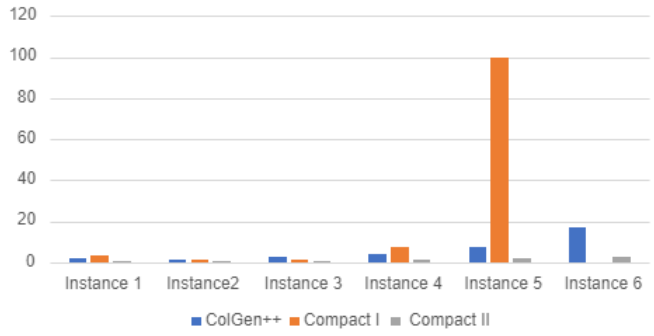

Fig. 4. Acyclic Graphs – Computation time (in seconds)

Inst	ColGen++	Compact I	Compact II
1	2	0	0
2	0	0	0
3	16	TL	2
4	0	TL	0
5	46	TL	2
6	11	TL	1

TABLE V: Cyclic Graphs – Interruption cost

Inst	ColGen++	Compact I	Compact II
1	0	0	0
2	0	0	0
3	1	0	0
4	0	0	0
5	0	0	0
6	5	TL	0

TABLE VI: Acyclic Graphs – Interruption cost

From the tables it may be concluded that the Column Generation method gives better results than the first compact formulation, while keeping the interruption cost under control. In the case of acyclic graphs, the cost is zero in most cases. However, it will also be remarked that the second compact formulation achieves the optimal solution in a very short time for the fixed target migration plan.

V. Concluding Remarks and Perspective

In this paper we have presented some new results on an optimization problem relating to resource allocation in 5G networks. We model the problem as a Multiple Multidimensional Knapsacks Reconfiguration problem and formulate it through Integer Linear Programming compact and non-compact models.

One of the main issues here is that solution choices have a high combinatorial potential, which may make the optimization problem computationally intractable for large problem instances. Indeed, from the experimental results we see that the first compact formulation becomes intractable when the number of items and the number of servers increase. We propose a Column Generation solution approach. This formulation is tractable and gives a smaller optimality gap comparing to compact formulations.

Nevertheless, the computation times remain significantly higher to those exhibited by the second compact formulation.

The drawback of the work done until now stands in fixing the target migration plan with respect to some placement of VNF in the servers. Computing the target placement may be done easily as a multiknapsack problem while minimizing the number of migrations. For the operator may be interesting to let the final placement free and let the algorithm find the best reconfiguration leading to minimum number of interruptions. This is called the free target migration plan and is the focus of our current work. At this stage, we have noticed that the second compact ILP problem model becomes computationally difficult comparing to the free target migration plan case, while the column generation method scales very well. Work is in progress on this issue.

Acknowledgment

The work of Artur Tomaszewski was supported by the National Science Centre, Poland, under grant no. 2017/25/B/ST7/02313: ”Packet routing and transmission scheduling optimization in multi-hop wireless networks with multicast traffic”.

REFERENCES

[1] Hanane Biallach, Mustapha Bouhtou and Dritan Nace, "Optimization of the virtual network function reconfiguration plan in 5G network slicing," ICN International Conference on Networks, 2022, (to appear)

[2] Renaud Sirdey, Jacques Carlier, Hervé Kerivin and Dritan Nace, "On a resource-constrained scheduling problem with application to distributed systems reconfiguration," European Journal of Operational Research, 2006

[3] Fernando Solano and Michał Pióro, "Lightpath reconfiguration in WDM networks," Journal of Optical Communications and Networking, vol. 2, no. 12, 2010

[4] Rickson S. Pereira, Douglas D. Lieira, Marco A. C. da Silva, Adinovam H. M. Pimenta, Joahannes B. D. da Costa, Denis Rosário, Leandro Villas and Rodolfo I. Meneguette, "RELIABLE: Resource allocation mechanism for 5G network using Mobile Edge Computing," 2020, DOI:10.3390/s20195449

[5] Eleonora Riva Sanseverino, Thanh Ngo Ngoc, Marzia Cardinale, Vincenzo Li Vigni, Domenico Musso, Pietro Romano and Fabio Viola, "Dynamic programming and Munkres algorithm for optimal photovoltaic arrays reconfiguration", Solar Energy, vol. 122, 2015

[6] Loubna Bouselham, Abdelhamid Rabhi, Bekkay Hajji and Adel Mellit, "Photovoltaic array reconfiguration method based on fuzzy logic and recursive least squares: An experimental validation," Energy, vol. 232, 2021

[7] Yang Song, Chi Zhang and Yuguang Fang, "Multiple multidimensional knapsack problem and its applications in cognitive radio networks," 2008, DOI: 10.1109/MILCOM.2008.4753629

[8] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala and Srinivasan Parthasarathy, "Breaking cycles in noisy hierarchies," ACM on Web Science Conference, 2017, DOI:10.1145/3091478.3091495, source code available at https://github.com/zhenv5/breaking_cycles_in_noisy_hierarchies.

[9] "Optimization with PuLP," URL: http://coin-or.github.io/pulp

[10] Guy Desaulniers, Jacques Desrosiers and Marius M. Solomon, "Column generation," GERAD 25th Anniversary Series, 2005.

[11] Marco Caserta and Stefan Voß, "The robust multiple-choice multidimensional knapsack problem," 2018, DOI:10.1016/j.omega.2018.06.014

[12] Markus Leitner, Ivana Ljubić, Markus Sinnl and Axel Werner, "ILP heuristics and a new exact method for bi-objective 0/1 ILPs: Application to FTTx-network design," 2016, DOI:10.1016/j.cor.2016.02.006

[13] Xiaokang Cao, Antoine Jouglet and Dritan Nace, "A multi-period renewal equipment problem," 2011, DOI: 10.1016/j.ejor.2011.12.011

[14] Timo Berthold, "Primal heuristics for Mixed Integer Program," 2006

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Single-source shortest paths. Introduction to Algorithms, Third Edition (pp. 643-707)," The MIT Press, 2009

[16] Fernando Solano and Michał Pióro, "WDM network re-optimization avoiding costly traffic disruptions," Telecommunication Systems, vol. 52, 2011