Unsplittable Multi-commodity Flow Problem via Quantum Computing

Miguel Pineda Martín, Sébastien Martin
Huawei Technologies & co. 20 Quai du Point du Jour, Boulogne Billancourt, France
miguelpinedamartin@gmail.com
sebastien.martin@huawei.com

Abstract*—Quantum computing is a promising area to tackle optimization problems. Several algorithms are derived from this paradigm. One interesting method is called Quantum Approximate Optimization Algorithm (QAOA) where the goal is to solve a Quadratic Unconstrained Binary Optimization. This method is heuristic in practice but can converge on an optimal solution with enough time and a computer with enough resources. In this paper, we focus on the unsplittable multi-commodity flow problem and we propose a dedicated algorithm using QAOA as a sub-routine. Thanks to well-known methods, cutting plane, column generation, and Lagrangian relaxation, we propose a framework to solve this problem based on QAOA. *Keywords: QAOA, Lagrangian relaxation, Columns Generation, Cutting Plane

I. Introduction

Quantum computing has had a fast theoretical development since its born in the 80s. This new type of computing allows us to give algorithms that can solve efficiently difficult well-known problems such as factorizing. However, quantum computing hardware development has been a much more hard task and there is no expectation about being able to execute algorithms such as Shor's factorization algorithm [1] for real-world problems in the near or middle term [2]. Nevertheless, in recent years there has been a great development of hybrid quantum-classical variational algorithms, designed to solve combinatorial optimization problems, which are much more suitable to be executed on short-term quantum devices.

One of the most promising variational algorithms is the so-called Quantum Approximate Optimization Algorithm (QAOA), proposed in [3]. It is a hybrid quantum-classical algorithm. The quantum part consists of a parameterized quantum circuit that returns solutions to the Quadratic Unconstrained Binary Optimization (QUBO) problems. A QUBO problem with n variables is given by

$$\min_{x\in\{0,1\}^n} c^T x + x^T C x, \quad C \in \mathbb{R}^{n\times n}, c \in \mathbb{R}^n.$$

The algorithm uses the solutions returned by the quantum circuit to compute the expected value of the objective function and with some classical methods the parameters are updated until some stop condition is achieved. One of the main features of this algorithm is that it allows us to control the number of parameters and the depth of the circuits. In addition, the quality of the solution is increased according to the circuit depth, achieving optimality in the limit. All these features make QAOA one of the most suitable algorithms to be executed on near-term noisy quantum hardware. Despite its potential for a quantum advantage over the best classical solvers, such as CPLEX, it must be explored empirically. QAOA has shown good performance on problems that can be formulated as a QUBO problem or with few constraints, such as Maximum Cut problem [3]. However, combinatorial optimization problems are not usually given as QUBOs. The most common way of giving such problems is Binary Linear Programming (BiLP). A common way of mapping a BiLP to a QUBO consists of adding the constraints to the objective function as a quadratic term with a penalty. However, when this method is applied, QAOA does not have such a good performance on problems where there are hard constraints that must be satisfied, such as flow problems. It usually happens that the final solution for the QUBO problem given by QAOA is not a feasible solution. In order to avoid this issue there are several proposals for changing the parameterization circuit of QAOA [4], [5].

In this paper, we focus on the unsplittable multi-commodity flow problem (UMCFP). This problem consists of finding, for a set of commodities, one path for each commodity such that the capacity constraints are respected. The UMCFP has many applications in telecommunication networks [10] and is a basic version of several variants. This problem is NP-Hard and two mathematical models are traditionally used to solve the problem, the first one is the compact model with arc variables and the second one is the extended model with path variables [11]. The compact has too many constraints and variables and the automatic transformation into a QUBO is not relevant. In the following, we focus on the extended model to transform it into a QUBO model. The main goal of this paper is to provide a good design framework to tackle the UMCFP by solving successively small QUBO problems to converge on a good feasible solution. In the next section, a state-of-the-art is given on QAOA method and integer model to solve the UMCFP. Section III is dedicated to the presentation of our method to solve the UMCFP problem. In section IV some results are provided to evaluate the performances of the different improvements proposed in Section III. In the last section, we conclude and give some perspectives.

II. State of the art

A. The Quantum Approximate Optimization Algorithm

We start giving a brief description of QAOA, more details can be found on [3]. Let's consider a QUBO problem given by,

$$\min_{x\in\{0,1\}^n} \varphi^T x + x^T \Phi x, \quad \Phi \in \mathbb{R}^{n\times n}, \varphi \in \mathbb{R}^n.$$

It can be mapped easily to the problem of finding the ground state of an Ising Hamiltonian H, i.e.

$$\min_{|\psi\rangle:\langle\psi|\psi\rangle=1} \langle\psi|H|\psi\rangle, \tag{1}$$

where $|\psi\rangle \in \left(\mathbb{C}^2\right)^{\otimes n}$ and $H \in \mathbb{C}^{2^n\times 2^n}$ is an Hermitian matrix that is built with Φ and φ. The approach of QAOA for solving the problem (1) consists of parameterizing the vector $|\psi\rangle$ with a quantum computer and making a search for the parameters which minimize the objective function using some iterative classical method. First, we choose a number $p \in \mathbb{N}$ and a vector of initial parameters $\theta \in \mathbb{R}^{2p}$. The number p is usually called the depth because it controls the depth of the circuit and the number of parameters. The parameterization is done via a quantum circuit, represented by a matrix that we can call $U(p,\theta)$. We use a quantum computer to prepare the parameterized quantum state $|\psi(p,\theta)\rangle = U(p,\theta)|0\rangle$. Measuring several shots of this state we can estimate the value of the objective function with some statistical methods [6]. After this, we use the estimation to update the parameters of θ with some classical methods. QAOA consists of iterating this process until some stop conditions are achieved.

If we call v^* the optimal value of (1), it was proven on [3] that,

$$\lim_{p\to\infty} \min_{\theta} \langle\psi(p,\theta)|H|\psi(p,\theta)\rangle = v^*.$$

This theoretical result tells us that we can get as close as we want to the optimal solution if we increase p enough. Nevertheless, from a practical point of view, if the value of p is too large, the parameterization won't be able to run on near-term quantum computers. Unfortunately, there is not a general convergence theorem for small values of p, but for specific problems, such as Maximum-Cut problem, QAOA has shown good results [3]. In the following, we will use QAOA as sub-routine in order to solve the UMCFP. There, we will always consider QAOA setting the value of p to 1.

B. Unsplittable multi-commodity flow problem

First of all, we consider a network, which is given by a directed graph $G = (V, A)$, where V and A are the sets of vertices and arcs respectively, and a set of commodities $\mathcal{K} = \{k_1, \ldots, k_n\}$. Each commodity is a 4-tuple $k_i = (s_i, t_i, b_i, r_i)$, where $s_i \in V$ and $t_i \in V$ are the source and destination nodes of $k_i \in K$ respectively, $b_i \in \mathbb{R}^+$ is the quantity of flow which has to be routed of k_i and $r_i \in \mathbb{R}^+$ is a reward for accepting the commodity k_i. In addition, each arc $a \in A$ has a capacity c_a associated and a cost $\omega_a \in \mathbb{R}^+$. The classical compact model consists in considering one binary variable for each couple commodity and arc representing the utilization of a link by a commodity and one binary variable for each commodity representing the acceptance of the commodity. The model can be found in [9] where flow conservation constraints are considered for each commodity and capacity constraints are associated with each arc.

Another formulation for the UMCFP, which is widely used and will be very useful for our purposes because of its practical properties, is called path formulation where for each possible path and for each commodity, a path variable is considered. The variables are defined as follows

$$x_p^k = \begin{cases} 1 & \text{if the commodity } k \text{ is routed on path } p \in P_k \\ 0 & \text{otherwise,} \end{cases},$$

$$y_k = \begin{cases} 1 & \text{if the commodity } k \in K \text{ is accepted,} \\ 0 & \text{otherwise,} \end{cases}$$

where for each $k \in \mathcal{K}$, P_k is the set of paths of the graph from the source node of the commodity k to its destination node. The extended model is given by the following model

$$\min \quad \sum_{k\in\mathcal{K}} \sum_{p\in P_k} \left(\sum_{a\in p} \omega_a\right) \cdot x_p^k - \sum_{k\in\mathcal{K}} r_k y_k \tag{2}$$

$$\text{s.t.} \quad \sum_{p\in P_k} x_p^k = y_k \quad \forall k \in \mathcal{K}, \tag{3}$$

$$\sum_{k\in\mathcal{K}} \sum_{p\in P_k : a\in p} b_k x_p^k \le c_a \quad \forall a \in A, \tag{4}$$

$$x_p^k, y_k \in \{0,1\} \quad \forall k \in \mathcal{K}, p \in P_k. \tag{5}$$

The objective (2) minimizes the utilization cost of the solution and maximizes the acceptance. The inequalities (3) are the convexity constraints provided by the Dantzig-Wolf [11] decomposition and give the relations between the two types of variables. The inequalities (4) are the capacity constraints ensuring that the capacity of each link is respected.

This model is the most common choice in practice, but, instead of using all paths, only some paths are considered. A traditional method to choose the paths is the column generation algorithm [7]. This method consists of solving, alternatively, a relaxed problem and a pricing problem in order to generate paths with good features, iteratively. This method cannot be applied in our case because QAOA only returns binary solutions. So, we propose in III-D some heuristics that can be used to generate good paths.

C. Classical reformulation of BiLP to QUBO (baseline)

The general method usually used in this context has two steps. The first one consists in adding slack variables to the inequality constraints in order to get a problem only with equality constraints, such as,

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & x \in \{0,1\}^n. \end{aligned}$$

The second step consists of adding the constraints to the objective function as penalized quadratic term, in the following way,

$$\begin{aligned} \min \quad & c^T x + M(Ax-b)^T(Ax-b) \\ \text{s.t.} \quad & x \in \{0,1\}^n, \end{aligned}$$

where $M > 0$ penalizes the objective function if some constraints are not satisfied. Remark that, if our problem is given as a BiLP model with too many constraints, this mapping does not have a good performance in practice. We have encountered two main issues.

1) On the one hand, if our problem has hard constraints and we add all of them as it is done on II-C the local optimal solution returned by QAOA for the QUBO may violate some constraints. It is quite frequent that in this case, QAOA outputs unfeasible solutions.
2) On the other hand, the mapping method described in II-C needs to add several slack variables to the model for each inequality constraint. It usually makes that, even for small instances, the QUBOs associated with the problems have too many variables to execute the QAOA on a simulator. In addition, the memory of actual quantum devices is very low, so it is also a problem for the execution of QAOA in near-term quantum devices

In order to avoid these issues, we propose in the next section a framework with different tools which allow us to solve UMCFP instances using QAOA as a subroutine.

III. Another mapping QAOA for UMFP

In this section, we provide some tricks to use QAOA in a sub-routine and reduce the size of the QAOA model. We also describe the whole framework to solve the UMFP.

A. Quadratic reformulation of convex function

In this section, we propose a reformulation of the path formulation to reduce the number of constraints. This reformulation adds quadratic variables which is a bad idea for the traditional solver but interesting for the QUBO reformulation. The convexity constraints (3) can be replaced by a quadratic formula in the objective function. Indeed, we can remove the convexity constraints (3) and replace the objective function (2) by

$$\begin{aligned} \min & \sum_{k\in K}\sum_{p\in P_k}(\sum_{a\in p}\omega_a)x_p^k \\ & - \sum_{k\in K}\sum_{p\in P_k} r_k x_p^k + M\sum_{k\in K}\sum_{p\in P_k}\sum_{p'\in P_k : p'\neq p} x_p^k x_{p'}^k \end{aligned} \quad (6)$$

where M is a penalty coefficient. Indeed, if two paths are selected for a given commodity then the penalty cost is activated. We call this model a partial QUBO model where capacity constraints are still in the model.

B. Lagrangian relaxation

The Lagrangian relaxation consists in moving capacity constraints in the objective function. In our case, we use this method to transform the partial QUBO model given by the previous sub-section in a QUBO. Using the Lagrangian relaxation it is not necessary to add slack variables like in the classical reformulation of BiLP to QUBO.

We obtain the following model,

$$\begin{aligned} \min \quad & \sum_{k\in\mathcal{K}}\sum_{p\in P_k}\left(\left(\sum_{a\in p}\omega_a\right) - r_k\right)\cdot x_p^k \\ & + M\sum_{k\in K}\sum_{p\in P_k}\sum_{p'\in P_k : p'\neq p} x_p^k x_{p'}^k \\ & + \sum_{a\in A}\lambda_a\left(\left(\sum_{k\in\mathcal{K}}\sum_{p\in P_k : a\in p} b_k x_p^k\right) - c_a\right) \quad (7) \\ \text{s.t.} \quad & x_p^k \in \{0,1\}. \quad (8) \end{aligned}$$

where $M > 0$ and $\lambda \in \mathbb{R}_{\geq 0}^{|A|}$. We use QAOA to solve (7), (8) in order to find a feasible solution for (2)-(5), and if it is not we use the solution to update λ, increasing the penalty of violated constraints and decreasing (or eliminating) the penalty for those constraints which are checked. Based on the traditional Lagrangian relaxation we use a common practice for this method, which is making use of a sub-gradient algorithm to update the value of λ.

In the literature of sub-gradient algorithms, a common practice is to use $step = \frac{1}{i}$, where i is the iteration of the sub-gradient algorithm. However, in [8], Polyak proposes the following step-size function which does not depends on the iteration,

$$step = \alpha\frac{UB - LB}{||gap||^2},$$

where α is a positive real number, UB is the value of the objective function (2) of the previous iteration, LB is the best feasible solution found and gap is the vector

$$(gap)_{a\in A} = \left(\left(\sum_{k\in\mathcal{K}}\sum_{p\in P_k : a\in p} b_k x_p^k\right) - c_a\right)_{a\in A}.$$

In this case, the algorithm starts with $LB = 0$. Based on some practical evidence we have chosen this second method for the simulation.

C. Cutting plane algorithm

Another well-known method in mathematical programming is the cutting plane algorithm. The goal is to generate a subset of a family of constraints. Originally this method is used to dynamically generate a polynomial number of constraints from a family of constraints with an exponential number of constraints. In our case, this method generates dynamically the capacity constraints. As QAOA is able to find an integer solution then we can use the cutting plane method as a lazy constraints generator, used to separate only integer solutions and not fractional solutions.

From an integer solution x^* that represents at most one path by commodity, we check for each link if the capacity constraints are violated and we add to the model the most violated capacity constraint following a mapping method, we have chosen here the method of II-C. We stop when some feasible solution is obtained or when a prefixed maximum number of iterations is reached.

D. Path generation

The path model presented in the previous section has an advantage in terms of the number of constraints, but the number of paths can be too high. In addition, we need to enumerate all paths if we want to solve the complete model. A very common classical way to handle this issue consists in solving a relaxation of (2)-(5), which consists of changing the condition (5) by $x_p^k \geq 0, 0 \leq y_k \leq 1$. This relaxed problem is solved by the column generation algorithm, it consists in solving iteratively the main problem, with a selected set of variables, and a pricing problem. In each iteration those prices are used to compute the path with the smallest reduced costs, using a shortest path algorithm, for each commodity. Those paths correspond to some variables which are added for the next iteration. This algorithm stops when no path has a negative reduced cost. The solution given by this algorithm is not necessarily binary, so it is needed to choose a rounding procedure in order to obtain a binary solution. A more detailed description of this algorithm can be found in [7].

Based on this idea, we propose a heuristic method to iteratively generate paths where a QUBO model is solved at each iteration to obtain a solution to the restricted problem. We cannot use a column generation algorithm because the solution of a QUBO is binary. However, we propose a heuristic to find new paths using binary solutions. The proposed method consists of assigning a weight to each arc on each iteration, based on the solution of the previous one. With these weights, we compute new paths using a shortest path algorithm. These new paths are added to the master problem. As with the classical method, we stop when the solution does not change after adding new paths.

Let us consider a UMCFP instance with the extended formulation, i.e. (2)-(5), $P'_k \subset P_k$ a restricted set of paths and the complete set of paths for each commodity k respectively and $\bar{x}$ a feasible solution for the UMCFP problem over the graph generated by the paths $\bigcup_k P'_k$. On each heuristic, we use this data to compute a weight for each arc.

a) Loads heuristic: The purpose of this heuristic is to find different paths from the ones we already have. In order to do so, we use what we have called loads. The load of an arc a is given by

$$\sum_{k \in \mathcal{K}} \sum_{p \in P_k : a \in p} b_k \bar{x}_p^k.$$

As $\bar{x}$ is a feasible solution $\sum_{k \in \mathcal{K}} \sum_{p \in P_k : a \in p} b_k \bar{x}_p^k \leq c_a$, so

$$l_a(\bar{x}) := \frac{\sum_{k \in \mathcal{K}} \sum_{p \in P_k : a \in p} b_k \bar{x}_p^k}{c_a} \in [0, 1].$$

We call this value the load proportion of a for the solution $\bar{x}$. As we want to penalize the arcs that support more load in relation to their capacity, we assign the following weight for each arc,

$$\alpha_a := \frac{1}{1 - l_a(\bar{x}) + \varepsilon},$$

where ε is a positive small value that avoids the division by 0.

b) Loads and costs heuristic: The purpose of this heuristic is again to find new paths but this time we do not penalize so much relatively that an arc has been used. Again, we consider the load proportions $l_a(\bar{x})$ but this time we use the cost of each arc. In this way, the weight for each arc $a \in A$ in this heuristic is

$$\alpha_a := \frac{\omega_a}{1 - l_a(\bar{x}) + \varepsilon}.$$

c) Lambda heuristic: The purpose of this heuristic is the same as the previous one, but this heuristic only can be used if we are solving the subproblems using the method based on lagrangian relaxation (III-B). In this case, we use the lagrangian penalties λ_a from (7) in the last iteration, which we denote $\lambda_a(\bar{x})$ as the cost of each arch, i.e.

$$\alpha_a := \lambda_a(\bar{x}).$$

E. Whole framework

In this section, we explain the framework and each step for our framework to converge to a good feasible solution. First, we explain how to solve a UMCFP instance when a set of paths is given. After this, we explain how we can generate good paths.

1) Framework with a given set of paths : We start with a set of paths $\mathcal{P}$ and a UMCFP instance given as (3)- (5), where the paths considered are those from $\mathcal{P}$. We start by mapping this problem to a partial QUBO model as we explained in III-A. Now, the only constraints in our model are capacity constraints. In order to give a solution for this model, we have proposed two iterative methods: the cutting method (III-C) and the lagrangian method (III-B). So we choose one and apply it to solve the problem.

2) Framework to generate paths: In order to initialize the algorithm, we need an initial selection of paths. It is done by solving the shortest-path problem for each commodity using the cost of the arcs. Let $\bar{P}_k$ be the set of paths for each commodity at each iteration and let $\bar{A}$ the set of arcs of each path of $\bar{P}_k$ for each commodity. At the first step, for

each commodity, $|\bar{P}_k| = 1$ with only the shortest path. Let us consider the restricted problem,

$$\begin{aligned}
\min \quad & \sum_{k\in\mathcal{K}} \sum_{p\in\bar{P}_k} \left(\sum_{a\in p} \omega_a \right) \cdot x_p^k - \sum_{k\in\mathcal{K}} r_k y_k \\
\text{s.t.} \quad & \sum_{p\in\bar{P}_k} x_p^k = y_k \quad \forall k \in \mathcal{K}, \\
& \sum_{k\in\mathcal{K}} \sum_{p\in\bar{P}_k : a\in p} b_k x_p^k \leq c_a \quad \forall a \in \bar{A}, \\
& x_p^k, y_k \in \{0,1\} \quad \forall k \in \mathcal{K}, p \in \bar{P}_k.
\end{aligned}$$

In order to solve this problem we can use now the previous framework. Thanks to it, we obtain a feasible solution $\bar{x}$. Now, we have to choose one of the heuristics presented in III-D. Then, we use $\bar{x}$ to compute the cost of each arc according to the heuristic chosen. With these costs, we compute again the shortest path for each commodity and we add the new paths found to the sets $\bar{P}_k$. This process is repeated until no new paths are found and the best feasible solution found is returned. The heuristic chosen may be changed on each step conveniently.

IV. Results

In order to test the framework proposed above, we have used a quantum computing simulator to solve random instances of UMCFP using the proposed techniques. The code for these simulations has been written in python using the IBM library qiskit [14] and the simulator used has been the qasm_simulator. We have made this choice because it also simulates the noise that is typical in actual and near-term quantum computers.

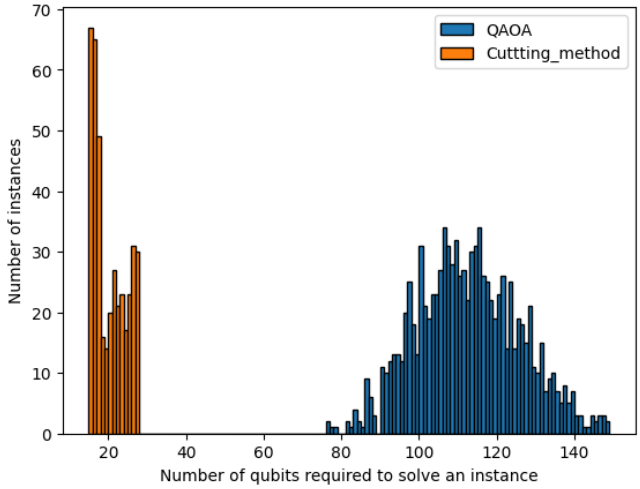
We have generated two datasets of random UMCFP instances. The first one has been used to compare the baseline with both cutting and Lagrangian methods, and the second one has been used to compare it with the algorithm using the heuristic to generate paths. As the memory that quantum computing simulators can simulate is quite low, we have bounded the number of paths of both datasets. Both datasets have been generated using the Erdos-Renyi method. In the following table, we present the common characteristics of the problems of each dataset. As we explained in II-C, the usual

	Dataset 1	Dataset 2
Number of nodes	10	30
Number of commodities	3	6
Minimal number of paths	7	18
Maximum number of paths	10	32

mapping method has two main problems. First, the memory required to solve a problems is too high because of the necessity of slack variables and, secondly, the solution given by QAOA is not usually a feasible solution. Here we make some comparison between our framework and the baseline in these aspects. Additionally, we compare the quality of the solution of our method with the quality obtained by the classical column generation algorithm.

A. Efficiency of the method compared with the baseline

Here we compare the memory that is needed to simulate the problems of Dataset 1, using the cutting method, with the memory that would be needed to solve them with QAOA using the mapping of II-C.



The unit of memory used is the qubit, a qubit is the minimal unit of memory in a quantum computer, which is usually represented by a vector of $\mathbb{C}^2$, but in our case, we can think of the number of qubits needed as the maximum number of variables of the models that are solved in the iterative process of III-E1. It shows that even for small instances of UMCFP, like Dataset 1 ones, the classical approach of QAOA would need an amount of memory that cannot be simulated, the limit is usually set at 50. However, using the cutting method we could simulate 883 instances out of 1000 of Dataset 1. The case of Lagrangian is even better because the number of variables that are used by this method does not increase with the iterations. So we could do the simulation in every problem and we have obtained a feasible solution for every problem of Dataset 1.

B. Quality of the solution

We want to compare the quality of the solution given by the classical column generation algorithm and the method to generate paths explained in III-E. In order to use it, we need to do two choices, we have to choose the method to solve the problem with a given subset of paths, i.e. we have to choose if we want to use the Lagrangian method III-B or the Cutting method III-C, and we have to choose which heuristic we use to generate paths in each iteration.

We have chosen the following criteria to select the heuristic for each iteration.

1) First, we start using the Loads heuristic until we do not obtain new paths.
2) Secondly, we use the loads and costs heuristic until we do not obtain new paths.

The idea behind this choice is that in the first phase, we do not penalize the cost of the paths, because we only want to

know the existence. In the second phase, we try to find paths with a small cost. We have not used the lambda heuristic because we have found empirically that it has much worse performance than these two heuristics.

We have used the method of III-D with the above criteria for the heuristics to solve the problems of Dataset 2, using Lagrangian and Cutting methods. In the case of the Lagrangian method, we have set $\alpha = 0.05$, because of practical evidence. In addition, we have solved those problems with the classical column generation method. In order to compare the quality of a solution x^*, which gives us a value C^* of the objective function, we compute the approximation ratio of this solution, given by

$$r = \frac{C^*}{\overline{C}},$$

where $\overline{C}$ is the optimal value of the objective function of our problem, which can be computed using CPLEX because of the small size of the problems.

	Cutting	Lagrangian	Column Generation
% of feasible solution	56.6	97.2	100
% of optimal solutions	38.9	89.9	94.2
% of feasible solution $r > 0.95$	72.9	95.2	95.8

As we can see in the table the algorithm has shown a much better performance when we have chosen the lagrangian method than when we used the cutting method. It is due to the necessity of adding slack variables to the model of the cutting method. If too many paths are generated, the simulation usually cannot finish if we use the cutting method.

On the other hand, the lagrangian method has shown a really good performance which is even comparable with the classical column generation method. It is important to notice that even using the cutting method is a much better option than the baseline we are considering because the size of problems of Dataset 2 is too high to be solved by QAOA using a simulator and it may be difficult to solve them even with actual better quantum computers.

V. Conclusion and perspectives

In this paper, we present a new framework to solve the UMCFP using QAOA approach. Although the techniques presented are specifically oriented to solve the UMCFP, some of them, such as considering linear penalties instead of quadratic ones in order to avoid the use of slack variables, could be applied to other problems. We show the efficiency of our method to reduce the size, in qubits, of the mathematical model in QUBO representation. It makes it possible to study the performance of QAOA solving UMCFP instances, which cannot be done using the baseline because of the size of the model compared with the capacity of the simulators and the memory of actual quantum devices.

Furthermore, our approach provides solutions that are close to the solutions provided by the column generation with a classical rounding method. In the next step, the goal will be to merge the methods proposed, the cutting plane method and the Lagrangian approach, to solve bigger instances. It would be also interesting to test our framework on real quantum devices in order to measure its performance.

Another interesting direction is the possibility to adapt our framework to solve some variants of the UMCFP like UMCFP with slot allocation problem presented in [12] or UMCFP with deterministic properties explain in [13].

References

[1] Peter W. Shor (1995) *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. arXiv:quant-ph/9508027

[2] Sabani, Maria and Galanis, Ilias and Savvas, Ilias and Garani, Georgia (2021), *Implementation of Shor's Algorithm and Reliability of Quantum Computing Devices*. In 25th Pan-Hellenic Conference on Informatics (PCI 2021). Association for Computing Machinery, New York, NY, USA, 392–396. https://doi.org/10.1145/3503823.3503895

[3] Edward Farhi, Jeffrey Goldstone, Sam Gutmann (2014). *A Quantum Approximate Optimization Algorithm*. https://doi.org/10.48550/arXiv.1411.4028

[4] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor G. Rieffel, Davide Venturelli, Rupak Biswas(2019) *From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz* https://doi.org/10.48550/arXiv.1709.03489

[5] Yuxuan Zhang, Ruizhe Zhang, Andrew C. Potter (2021), *QED driven QAOA for network-flow optimization*. https://doi.org/10.48550/arXiv.2006.09418

[6] Aram W. Harrow and John C. Napp (2021, *Low-Depth Gradient Measurements Can Improve Convergence in Variational Hybrid Quantum-Classical Algorithms*, Phys. Rev. Lett. 126, 140502. https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.126.140502

[7] Cynthia Barnhart, Christopher A. Hane, Pamela H. Vance, (2000)*Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems*. Operations Research 48(2):318-326. https://pubsonline.informs.org/doi/abs/10.1287/opre.48.2.318.12378

[8] B. T. Polyak, (1987) *Introduction to optimization. optimization software* Inc.,Publications Division, New York, vol. 1, 1987.

[9] Alvelos, Filipe & Carvalho, José. (2003). *Comparing branch-and-price algorithms for the unsplittable multicommodity flow problem.*

[10] Wang, I. L. (2018).*Multicommodity network flows: A survey, Part I: Applications and Formulations*. International Journal of Operations Research, 15(4), 145-153.

[11] C. Barnhart, C.A. Hane, P.H. Vance, "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems". Operations Research, 48, 318-326, 2000.

[12] N. Huin, J. Leguay, S. Martin, P. Medagliani, S. Cai, "Routing and Slot Allocation in 5G Hard Slicing". 9th INOC, 2019.

[13] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, X. Geng, "Joint routing and scheduling for large-scale deterministic ip networks". Computer Communications 165, 33-42, 2021.

[14] https://qiskit.org/