NExt ApplicationS of Quantum Computing

# D5.8: Specification and implementation of coloring-based QAOA algorithm for minimizing charging station numbers

## Document Properties

| | |
|---|---|
| Contract Number | 951821 |
| Contractual Deadline | 30-10-2023 |
| Dissemination Level | Public |
| Nature | Report |
| Editors | Emmanuel Jeandel, UL |
| Authors | Emmanuel Jeandel, UL |
| Reviewers | Vedran Dunjko, ULEI<br>Arseny Kovyrshin, AstraZeneca |
| Date | 14-06-2023 |
| Keywords | QAOA, Branch and Price |
| Status | Reviewed |
| Release | 0.2 |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

## History of Changes

| Release | Date | Author, Organisation | Description of Changes |
|---|---|---|---|
| 0.1 | 03/10/2023 | E Jeandel, UL | First Version |
| 0.2 | 26/10/2023 | E Jeandel, UL | Reviewers comments taken into account |
| | | | |
| | | | |

# Table of Contents

# 1 Executive Summary

This deliverable is part of a series of three deliverables, provided jointly by EDF and Université de Lorraine:

- D5.3 Specification of a QAO algorithm for algorithm for the minimization of charging time
- D5.8 Specification and implementation of coloring-based QAOA algorithm for minimising charging station numbers
- D5.10 Software suite for benchmarking of quantum algorithms applied to the two typical smart-charging optimisation problems

As a midpart of the triptych, this deliverable will contain some overlap with the other two deliverables. In particular, deliverable D5.3 already supplied some details on coloring problems, that will be made more precise here.

The deliverable describes the specification of an algorithm to find the minimum number of colors required to color the vertices of a graph such that two connected vertices do not have the same color. This quantity is known as the chromatic number.

This problem arises naturally when considering scheduling problems for charging stations, that can easily be encoded into a graph coloring problem.

The main specificity of the algorithm is that the QAO (Quantum Approximate Optimization) algorithm is not used directly on the problem, but is part of a general Branch and Price method, where the QAO algorithm serves as a heuristics.

# 2 Introduction

This deliverable will be focused on a method to solve coloring problems using QAOA, and on practical details of the implementation.

The subject are therefore graph coloring problems, and more precisely the problem of finding the *chromatic number* of a graph, i.e. the minimum number of colors to assign to the vertices of a undirected graph so that no two adjacent vertices have the same color. This problem is one of the fundamental NP-complete problems (Garey & Johnson, 1979), and one of the most difficult ones, as it is well known that it is hard to approximate (Feige & Kilian, 1998; Garey & Johnson, 1976). This problem appears naturally in many applications, such as scheduling or register allocation (Chaitin et al., 1981; Marx, 2004).

This problem can be easily encoded into a Quadratic Binary Optimization Problem (QUBO) (Kochenberger et al., 2005), which makes it readily exploitable using the Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al., 2014). One technical difficulty in exploiting QAOA is to be certain that the solution obtained by the algorithm is indeed a proper coloring. Alternative solutions have been proposed (Hadfield et al., 2019), based on a mixing operator that guarantees that the evolution stays in the subspace of proper colorings.

Based on our work presented in our previous deliverable, we have found however that the quality of QAOA on current hardware (with few qubits and especially with small depth) does not give solutions that are good enough for industrial problems, which usually needs the optimal solution or a solution very close to the optimal. We suggested in D5.3 a new approach, based on Branch and Price, where QAOA is used as a subroutine in the pricing step.

We present in this deliverable the underlying specification. In the first section, we present the coloring problem, and how it can be written as a linear program, which is key to use Branch and Price. The next section is devoted to the explaination of the principles of the Branch and Price algorithm in general, and how it can be used in our context; the pricing problem that needs to be solved inside the Branch and Price method for solving the coloring problem is a different NP-complete problem, the Maximum Weighted Independent Set (MWIS). The next section presents our implementation of (R)QAOA for this specific problem. Then we give specific implementation details about Branch and Price in the last section

While the whole deliverable is written specifically for the coloring problem, the main takeaway is that QAOA is a **good** algorithm to find solutions fast that a greedy algorithm might have missed. For hard optimisation problems, QAOA is not always suitable as the main optimization algorithm. However as this series of deliverables will show, even then, it can provide good performance when used inside a whole optimization chain, typically by applying QAOA as a subroutine inside a Branch and Price algorithm.

# 3 Graph Coloring

In this short section, we present the graph coloring problem, and how it can be represented as a linear program.

**Definition 1.** *Let $G = (V, E)$ be an undirected graph.*

*A coloring of a graph $G$ with $N$ colors is a function $f : V \to \{1, \dots, N\}$ that assigns to each vertex of $G$ an integer, i.e., its color. A coloring is proper if every two adjacent vertices have different colors: $\forall(u, v) \in E, f(u) \neq f(v)$.*

*The chromatic number of a graph, denoted $\chi(G)$, is the minimum number $N$ such that there is a proper coloring of $G$ with $N$ colors.*

As explained in the introduction, for a fixed value of $k$, deciding if $\chi(G) \leq k$ (if $k$ colors are enough to color the graph properly) is NP-complete (Garey & Johnson, 1979). Furthermore, there exists no constant-factor approximation of $\chi(G)$ (Garey & Johnson, 1976), and not even an approximation algorithm up to a factor $n^{1-\epsilon}$ where $n$ is the number of vertices (Feige & Kilian, 1998; Zuckerman, 2007).

In the following, we will use integer linear programming to solve the coloring problem.

The canonical linear program for this problem is straightforward. If colors are to be chosen in a fixed set $C$, a linear program for coloring with $k$ colors can be achieved using variables $x_{uc}$, of value $1$ if the vertex $u$ is of color $c$ and $0$ otherwise.

The constraints should therefore state that every vertex has a color, and that two adjacent vertices have different color (it is not necessary to state that each vertex has *at most* one color), which leads to the following formulation:

$$\begin{array}{lcll}
\sum_{c \in C} x_{uc} & \geq & 1 & u \in V \\
x_{uc} + x_{vc} & \leq & 1 & (u, v) \in E, c \in C \\
x_{uc} & \in & \{0, 1\} & u \in V, c \in C
\end{array}$$

One can obtain a linear programming for the chromatic number by choosing $C = \{1, \dots, n\}$, where $n$ is the number of vertices, and minimizing the number of colors that are actually used.

This formulation has several drawbacks (**Mala**). The first problem is that, if the graph can be colored with $k$ colors, it can be done in at least $\binom{k!n}{k}$ ways by permuting the colors. Indeed, we can first decide of any choice of $k$ integers of $C$ as the colors to be used, and then permute the colors. The second problem is that the relaxation of the problem (with variables in $[0, 1]$ rather than $\{0, 1\}$), which is the single most important technique for solving integer linear programs, has an obvious solution $x_{uc} = 1/2$ for all $c \in C, u \in V$, which does not give any information on how to solve the problem.

In most applications, it is thefore more practical to use another linear program for this problem, based on independent sets:

**Definition 2.** *An independent set in a graph $G$ is a subset $I$ of vertices s.t. there are no edges between the vertices of $I$: $\forall u \in I, \forall v \in I, (u, v) \notin E$.*

A coloring with $n$ colors is therefore a partition[1] of $V$ into $n$ independent sets. By choosing to work with independent sets instead of colors, some symmetries of the problem are avoided. For instance a proper coloring of the vertices of $V$, and another coloring obtained by permuting the colors in the first coloring, correspond to the same partition into independent sets.

Let $S$ be the set of all independent sets of a graph. The problem of coloring can be reformulated as follows: we must identify a subset of $S$ (i.e. a set of independent sets) s.t. every vertex is covered by (at least) one chosen element.

Let $x_I$ be the variable saying that we chose the independent set $I$ in our partition (covering).

Then the linear problem becomes

$$\begin{array}{lcll}
\sum_{I \in S, u \in I} x_I & \geq & 1 & u \in V \\
x_I & \in & \{0, 1\} & I \in S
\end{array}$$

[1] A covering would also work.

and the goal is to minimize the number of independent sets, so the final linear problem is:

$$\begin{array}{llcll} \min & \sum_{I\in S} x_I & & & \\ s.t. & \sum_{I\in S, u\in I} x_I & \geq & 1 & u \in V \\ & x_I & \in & \{0,1\} & I \in S \end{array}$$

There are however two problems with this formulation. First, it is exponential in the number of vertices, as the number of independent sets of a graph with $k$ vertices can be exponential in $k$. Second, there are no easy way to describe, or enumerate the independent sets.

This problem therefore cannot be solved efficiently directly using classical integer linear programming techniques. One way to solve it involves column generation and pricing, that we explain in the next section.

# 4 Introduction to Branch-and-Price

We now present the Branch and Price algorithm in general, and how we use QAOA as a subroutine inside the algorithm. To do this, we provide an introduction to (integer) linear programming.

## 4.1 Linear Programming

### 4.1.1 Primer on linear programming

Linear programming is devoted to solving problems of the form:

$$\begin{array}{lll} (P): & \min & \sum c_i x_i \\ & s.t. & \sum_i a_{ij} x_i \geq b_j \\ & & x_i \geq 0 \end{array}$$

In matrix form:

$$\begin{array}{lll} (P): & \min & c^T x \\ & s.t. & Ax \geq b \\ & & x \geq 0 \end{array}$$

where $x$ is the variables and $A, b, c$ are inputs of the problem.

It is important to note that most of the techniques presented in this section work only for linear programs, that is programs where the variables are real variables, and not for integer linear programs, i.e. programs where the variables are integer.

A solution of the problem is any $x \in \mathbb{R}^n$. It is feasible if it satisfies the constraints, and optimal if it is as good as any feasible solution (we don't ask for an optimal solution to be feasible).

To simplify the exposition, we will make some assumptions, in particular all programs we are looking at will not be degenerate, and will have a feasible and optimal solution.

The most important solutions of a linear program are *basic* solutions. We will not give an exact definition, for which we would need the definition of linear programs in standard form. If we think of the set $\{x|Ax \geq b, x \geq 0\}$ geometrically, i.e. as a polyhedron, the feasible basic solutions correspond to the "corners" of the polyhedron. If there is a feasible optimal solution to a problem, there is always a basic one. Most LP solvers, and in particular all solvers based on the simplex method, always return basic solutions.

To each linear program, one can associate a *dual* program:

$$\begin{array}{lll} (D): & \max & \sum b_i y_i \\ & s.t. & \sum_i a_{ji} y_j \geq c_i \\ & & y_i \geq 0 \end{array}$$

in matrix form:

$$\begin{array}{lll} (D): & \max & b^T y \\ & s.t. & Ay \geq c \\ & & y_i \geq 0 \end{array}$$

with the following property: to each basic solution $x$ of the first problem, one can associate a basic solution $y$ of the dual problem s.t.:

- $x$ is feasible iff $y$ is optimal
- $y$ is feasible iff $x$ is optimal

When finding an optimal and feasible solution, the simplex method always finds both $x$ and its dual $y$ simultaneously. The data of $y$ is an additional data we can use for further optimizing.

Notice that constraints in the dual program correspond to variables in the primal (original) program, and conversely.

### 4.1.2 Duality for Coloring

Let's go back to the integer linear program for coloring:

$$(COL): \quad \begin{array}{lllll} \min & \sum_{I\in S} x_I & & & \\ s.t. & \sum_{I\in S, u\in I} x_I & \geq & 1 & u \in V \\ & x_I & \in & \{0,1\} & I \in S \end{array}$$

Forgetting about the integral constraints for a moment, its dual is:

$$(DCOL): \quad \begin{array}{lllll} \max & \sum_{u\in V} y_u & & & \\ s.t. & \sum_{u\in I} y_u & \leq & 1 & I \in S \end{array}$$

Suppose that our LP solver provided a basic feasible solution $x$, which is integral, together with its dual $y$, and we are interested in knowing whether $x$ is optimal[1]. This is equivalent to $y$ being feasible, i.e.

$$\forall I \in S, \sum_{u\in I} y_u \leq 1$$

This can be interpreted in terms of a weighted independent set problem. Consider $y_u$ as the weight of the vertex $u$. For an independent set $I$, let $w(I)$ be the sum of the weights of all vertices in $I$. Then $x$ is optimal if and only if all independent sets of the graph $G$ are of weight less than or equal to 1. Said otherwise, $x$ is optimal if and only if the maximum weighted independent set on the graph is less than or equal to 1.

This means verifying whether a given coloring is optimal is related to computing a maximum weighted independent set. This approach is at the basis of the Branch and Price algorithm for this problem.

## 4.2 Column generation

### 4.2.1 Primer on column generation

Column generation is a method to deal with linear programs with a large number of variables, by first solving the problem with fewer variables, and then adding more variables if necessary.

Consider the following LP:

$$(MP): \quad \begin{array}{ll} \min & c^T x + c'^T x' \\ s.t. & (Ax + A'x') \geq b \\ & x, x' \geq 0 \end{array}$$

and assume we have found with the simplex method an optimal feasible solution for the problem with the subset of variables $x$.

$$(RMP): \quad \begin{array}{ll} \min & c^T x \\ s.t. & Ax \geq b \\ & x \geq 0 \end{array}$$

(The notation MP and RMP are for "Master Problem" and "Restricted Master Problem"). Typically $x'$ will have a larger dimension than $x$.

[1] To be precise, the duality criterion will tell us whether $x$ is optimal amongst all feasible solutions of the linear program, not if it's optimal among all *integral* feasible solutions.

Now the dual of the original problem is:

$$\begin{array}{lll} (MPD): & \max & b^T y \\ & s.t. & A^T y \leq c \\ & & A'^T y \leq c' \\ & & y \geq 0 \end{array}$$

From an optimal and feasible solution $x$ of the problem $(RMP)$ one can easily obtain a feasible solution of the problem $(MP)$ by setting $x' = 0$, but the solution for the extended problem might not be optimal anymore.

If $y$ is the dual of the solution $x$ of the restricted problem $(RMP)$, it is still the dual of the solution for the extended problem $(MP)$. However, while it was feasible for $(RMP)$ it might not be feasible anymore (which means $x$ might not be optimal anymore for $(MP)$) as we added new constraints $A'^T y \leq c'$ that needs to be checked.

There are two cases:

1. $A'^T y \leq c'$. In which case $x$ is still an optimal solution of the problem
2. There exists a row $i$ s.t. $A'^T_i y > c'_i$. The solution $x$ is therefore not optimal anymore. Recall that each constraint of the dual corresponds to a variable in the primal problem (with our notation, the variable corresponding to the $i$th row of $A'$ is the variable $x'_i$). Such a variable is called an *ameliorating variable*. We can therefore add this new variable to the problem, and reiterate until we go back to the first case.

Choosing a row $i$ s.t. $A'^T_i y > c'_i$, or proving no such row exists is called *pricing*.

If the problem has a combinatorial origin, the dual problem usually also has a combinatorial nature, and finding such a row is a combinatorial problem that can hopefully be solved with specific techniques.

### 4.2.2 Pricing for the Coloring problem

If we go back to the definition of the coloring in terms of a linear program, a restricted problem, with less variables, corresponds to solving the problem with a subset $S'$ of the independent sets:

$$\begin{array}{lllll} (RCOL): & \min & \sum_{I \in S'} x_I & & \\ & s.t. & \sum_{I \in S', u \in I} x_I & \geq & 1 & u \in V \\ & & x_I & \in & \mathbb{R}^+ & I \in S' \end{array}$$

Knowing the feasible and optimal solution of this restricted problem, one can decide whether it's still an optimal solution of the original problem using duality:

$$\begin{array}{lllll} \max & \sum_{u \in V} y_u & & & \\ s.t. & \sum_{u \in I} y_u & \leq & 1 & I \in S \end{array}$$

As hinted at in a previous subsection, knowing whether the solution stays optimal for the whole set $S$ instead of $S'$ amounts to solving a *weighted independent* set problem:

- Consider the graph $G = (V, E)$ where vertex $u$ has weight $y_u$.
- Find the maximum weighted independent set $I$ of $G$.
- If this set has weight $> 1$, then add this set as a new variable in $S'$.
- If it doesn't, then the solution stays optimal, and the optimisation has ended.

It is important to note that we don't always need to actually find the maximum weighted independent set in the second step: Indeed, in the case where there is an ameliorating variable $I$, we just need to find this variable; A heuristics that gives independent sets of large weight might be enough to know our solution is not optimal, and to reiterate the simplex method adding this variable. However, it is necessary to have computed the maximum weighted independent set to actually prove that there is no ameliorating variable $I$.

**To find an ameliorating variable, we suggest to use QAOA on the maximum weighted independent set problem.** Indeed our previous deliverable shows that, while QAOA is not always better than a greedy algorithm at finding good solutions, it still performs better in some cases. Combining both the greedy algorithm with QAOA will therefore augment the number of cases where we are able to find an ameliorating variable without having to resort to find the optimal solution of the MWIS problem.

## 4.3 Branching

### 4.3.1 Primer on branching

The previous column generation method has a major drawback: it doesn't take into account the fact that the variables $x$ need to be integer.

If the solution found by the method is not integral, a classical solution is to do *branching*. There are many forms of branching. Possibly the easiest form of branching is to take one variable that is not integral in the solution, say, the variable $x$ is equal to $z \notin \mathbb{N}$ and to branch on two cases $x \leq \lfloor z \rfloor$ and $x \geq \lceil z \rceil$, adding this new constraint to the linear program.

In the context of pricing, some precautions have to be taken, and some typical branching rules cannot be used. If we go back to the original problem, a typical branching scenario would be to branch on whether some given independent set is part or not of the solution. However, this branching is highly nonsymmetric: there are at most $n$ independent sets in the optimal solution, and there are possibly $2^n$ candidate independent sets. Therefore the two branches in the branching tree are unbalanced.

Furthermore, the whole idea of the pricing step is that the dual problem has a specific combinatorial structure. The new constraints we add to the problem should therefore not perturb this structure. In our example, it is difficult for instance to take into account the fact that some independent set is not part of the solution: we would have to force the MWIS solver to look for a maximum weighted independent set, purposefully avoiding to look at some good independent sets.

### 4.3.2 Branching for coloring

For our specific case, there is a well-known branching rule that guarantees that the subproblems have the same combinatorial structure as the original one, the *Ryan-Foster* rule (Ryan & Foster, 1981).

The idea is to branch on whether two specific vertices should be or not in the same independent set. Indeed, one can prove that in a fractional solution of the linear program, one can find two vertices $u, v \in V$ s.t.

$$0 < \sum_{I \in S' | u,v \in I} x_I < 1$$

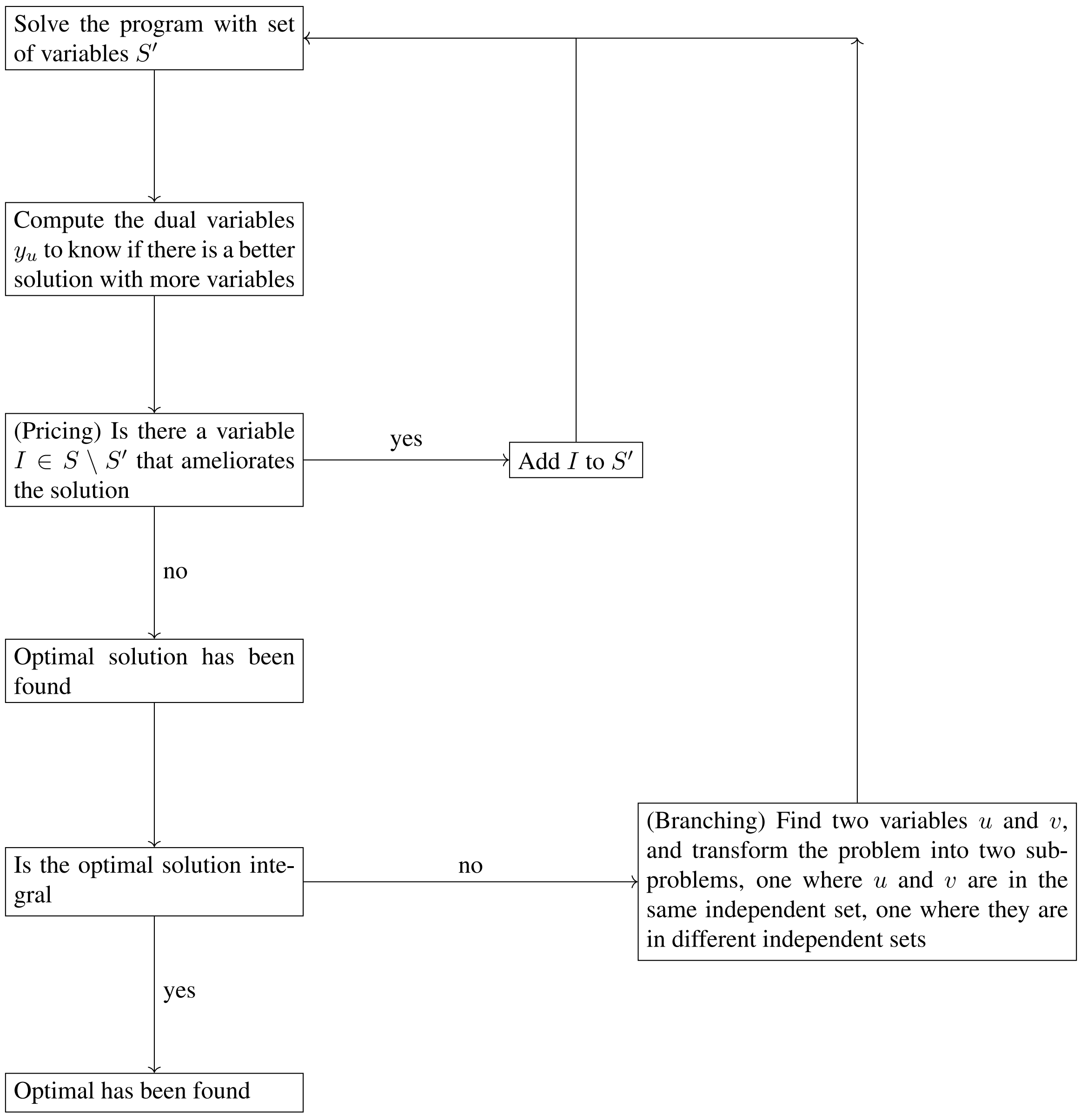
Notice that, if all independent sets of the solution contains at most one of the two vertices $u$ and $v$, then this sum is $0$ If all independent sets of the solution that contains one vertex also contains the other, the sum should be $1$. Therefore, the fact that the sum is fractional means that the solution of the program did not "decide" whether $u$ and $v$ should be in the same independent set or not.

We will therefore branch on whether $u$ and $v$ are in the same independent set. The most important point about this branching is that the two cases can be easily translated in terms of graph operation:

- To indicate that $u$ and $v$ should be in the same independent set, one can merge the two vertices into a unique vertex, with neighbors the union of the neighbors of $u$ and $v$
- To indicate that $u$ and $v$ should be in different independent sets, one can just add an edge between $u$ and $v$.

## 4.4 Branch and Price

The whole Branch and Price algorithm can be summed up by the following diagram:


Solve the program with set of variables $S'$
Compute the dual variables $y_u$ to know if there is a better solution with more variables
(Pricing) Is there a variable $I \in S \setminus S'$ that ameliorates the solution
yes
Add $I$ to $S'$
no
Optimal solution has been found
Is the optimal solution integral
no
(Branching) Find two variables $u$ and $v$, and transform the problem into two subproblems, one where $u$ and $v$ are in the same independent set, one where they are in different independent sets
yes
Optimal has been found

# 5 A QAO algorithm for Max Weighted Independent Set

The previous section has explained the general principle of the Branch and Price algorithm. Before giving details to the implementation, we have to focus on the quantum part of the algorithm, which is the resolution of the Max Weighted Independent Set problem. From the previous section, we know that we need an algorithm to find a solution as good as possible to this problem, but that it is not necessary to find the optimal solution, which is why RQAOA is a good candidate algorithm.

## 5.1 Definitions

To make this section self-contained, we recall back the definition of an independent set:

**Definition 3.** *Let $G = (V, E)$ be a finite graph, and $w : V \mapsto \mathbb{R}^+$ a weight function. We write $w_v$ instead of $w(v)$ for the weight of the vertex $v$.*

*An independent set is a subset $I$ of $V$ that contains no edges: $\forall u, v \in I (u, v) \notin E$. The weight of a independent set is the sum of the weight of the vertices in the set: $w(I) = \sum_{v\in V} w_v$.*

The maximum weighted independent set (MWIS for short) problem is to find the independent set of maximum weight.

A classical formulation of this problem as a linear program is as follows:

$$\begin{array}{lll} \max & \sum w_v x_v & \\ s.t. & x_u + x_v \leq 1 & (u,v) \in E \\ & x_v \in \{0,1\} & v \in V \end{array}$$

This can be transformed as a QUBO problem as follows:

$$\begin{array}{lll} \max & \sum w_v x_v - \lambda \sum_{(u,v)\in E} x_u x_v & \\ s.t. & x_v \in \{0,1\} & v \in V \end{array}$$

Where $\lambda$ is a parameter chosen large enough. We can take for instance $\lambda = 1 + W$ where $W$ is the maximum weight.

It is important to note that, while optimal solutions of the two problems are equivalent, non-optimal solutions of the second problem might not correspond to an independent set, and therefore might not correspond to a feasible solution of the first problem.

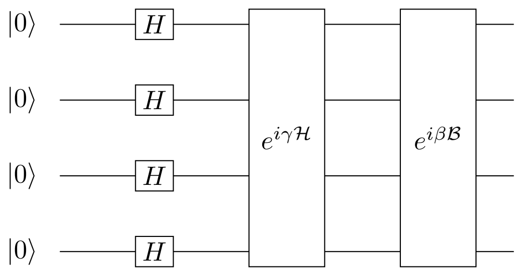
## 5.2 A RQAOA algorithm for Maximum Weighted Independent Set

Based on the QUBO formulation, we see that the problem correspond to maximizing the Hamiltonian

$$\mathcal{H} = \sum w_v Z_v - \lambda \sum_{(u,v)\in E} Z_u Z_v$$

We will optimize this Hamiltonian using RQAOA at depth 1.

The circuit for QAOA at depth 1 is as follows:



where $B = \sum_v X_v$ is the mixing Hamiltonian. We call $|\psi(\gamma,\beta)\rangle$ the state obtained by the previous circuit.

In the QAOA algorithm, the general idea is to find the optimal values of the parameters $\gamma, \beta$ that minimize $\langle\psi(\gamma,\beta)|\,\mathcal{H}\,|\psi(\gamma,\beta)\rangle$, and then use the circuit to sample some possible solutions to the problem that we expect to be good.

On the contrary, in the RQAOA algorithm, the values of $\gamma$ and $\beta$ are used to fix some variable in the problem, and then reiterate the algorithm (hence the R for recursive in RQAOA) on a smaller problem. Specifically:

- For each $u \in V$, compute the expectation of the observable $\langle\psi(\gamma,\beta)|\,Z_u\,|\psi(\gamma,\beta)\rangle$
- For each $u, v \in V$, compute the expectation of the observable $\langle\psi(\gamma,\beta)|\,Z_u Z_v\,|\psi(\gamma,\beta)\rangle$
- If the highest value observed corresponds to a variable $u \in V$, fix the variable to $0$ or $1$ depending if the expectation is closer to $1$ or $-1$.
- If the highest value observed corresponds to a pair $(u, v) \in V$, fix one variable to be positively, or negatively, correlated to the other one, depending if the expectation is closer to $1$ or $-1$.

## 5.3 Practical implementation of the Algorithm

We now detail the specifics of the implementation of the Algorithm. The first thing to note is how $\beta$ and $\gamma$ are optimised. At large depths, one has to use a quantum computer to produce, for each value of $\beta$ and $\gamma$, a large number of samples to be able to estimate $\langle\psi(\gamma,\beta)|\,\mathcal{H}\,|\psi(\gamma,\beta)\rangle$.

At depth $p = 1$, it is possible to obtain an analytical expression for $\langle\psi(\gamma,\beta)|\,\mathcal{H}\,|\psi(\gamma,\beta)\rangle$, so that we can run it without a quantum computer.

The following formulas were obtained in (Bravyi et al., 2020; Veshchezerova, 2022):

**Theorem 1.** *Let*

$$\mathcal{H} = \sum \alpha_v Z_v + \sum_{u \neq v} \pi_{u,v} Z_u Z_v$$

*and*

$$|\psi(\gamma,\beta)\rangle = e^{i\beta B} e^{i\gamma\mathcal{H}}\,|+\rangle^n$$

*Then*

$$\langle\psi(\gamma,\beta)|\,Z_u\,|\psi(\gamma,\beta)\rangle = \sin(2\beta)\sin(2\gamma\alpha_u)\prod_v \cos(2\gamma\pi_{u,v})$$

$$\langle\psi(\gamma,\beta)|\,Z_u Z_v\,|\psi(\gamma,\beta)\rangle = \frac{1}{2}\begin{bmatrix} \sin(4\beta)\sin(2\gamma\pi_{u,v})\cos(2\gamma\alpha_u)\prod_{w\neq v}\cos(2\gamma\pi_{u,w}) \\ +\sin(4\beta)\sin(2\gamma\pi_{u,v})\cos(2\gamma\alpha_v)\prod_{w\neq u}\cos(2\gamma\pi_{v,w}) \\ -\sin^2(2\beta)\cos(2\gamma(\alpha_u+\alpha_v))\prod_{w\neq v,u}\cos(2\gamma(\pi_{u,w}+\pi_{v,w})) \\ +\sin^2(2\beta)\cos(2\gamma(\alpha_u-\alpha_v))\prod_{w\neq v,u}\cos(2\gamma(\pi_{u,w}-\pi_{v,w})) \end{bmatrix}$$

As the Hamiltonian $\mathcal{H}$ is a sum of terms $Z_u Z_v$ and of terms $Z_u$, it is enough to give the formulas for these two terms.

The function $\langle\psi(\gamma,\beta)|\,\mathcal{H}\,|\psi(\gamma,\beta)\rangle$ can then be optimized with a classical algorithm. In our case, we chose to use the software NLopt (Johnson, 2007), starting first with a global optimizer using Multi-Level Single-Linkage (MLSL), then obtain more precise results using the COBYLA ((Constrained Optimization BY Linear Approximations) local algorithm, see the documentation of (Johnson, 2007) for details.

The recursive QAO algorithm is used until the problem becomes solvable by a brute force algorithm, which happens typically when the number of vertices is less than 16.

To obtain better results, the output of RQAOA is fed to a local search algorithm (Diogo Andrade, 2008). Specifically, the algorithm looks for $2 - improvements$, i.e. tuples $(u, v, w)$ s.t. $u \in I$, $v, w \notin I$ and $I \setminus \{u\} \cup \{v, w\}$ is an independent set of larger weight. The article (Diogo Andrade, 2008) is only concerned by maximum independent sets, but can easily be adapted to weighted maximum independent sets.

As hinted before, RQAOA does not always give the optimal solution to the QUBO problem, and in particular the solution found by RQAOA may fail to be an independent set.

# 6 Implementation

We will now provide details about the specific implementation of the Branch-and-Price algorithm for the coloring problem.

The implementation is done using the open-source software SCIP (Bestuzheva et al., 2021).

## 6.1 Initialization

The Branch and Price method needs to start with a solution of the problem. To obtain a good starting point, the DSatur algorithm of Brélaz (Brélaz, 1979) is used. The algorithm works as follows. At each step:

1. Find an uncolored vertex which has the most constraints, in the sense that the number of colors used by its neighbors is as large as possible
2. Color this vertex with the lowest color available.

The algorithm starts by coloring a vertex of maximum degree.

## 6.2 Internal representation of the problem

Based on the column generation framework, and the choice of the Ryan-Foster rule for branching, the data at any step can be represented as follows:

$$\begin{array}{lllll}
\min & \sum_{I \in S'} x_I & & & \\
s.t. & \sum_{I \in S', u \in I} x_I & \geq & 1 & u \in V \\
 & x_I & \in & \{0,1\} & I \in S' \\
 & \multicolumn{4}{l}{\text{Some vertices are in the same independent set}} \\
 & \multicolumn{4}{l}{\text{Some vertices are in different independent sets}}
\end{array}$$

where $S'$ is the subset of independent sets that we are currently looking at.

The last constraints are encoded into semantics constraints that need to be transformed into real constraints for the LP solver. This is done as follows when the linear program is generated:

- If vertices $u$ and $v$ need to be in the same independent set, then no independent set that contains only one of the two vertices can be part of the solution. Therefore set $x_I = 0$ for every independent set $I$ that contains only one of the two vertices $u$ and $v$.
- If vertices $u$ and $v$ need to be in different independent sets, then no independent set that contains both $u$ and $v$ can be part of the solution. Therefore set $x_I = 0$ for every independent set $I$ that contains both $u$ and $v$.

## 6.3 Pricing

In the pricing step, supposing having a feasible solution of the problem, we want to find an ameliorating variable, or prove no such variable exists.

This is done as follows:

1. Deduce from the feasible solution the value of the dual variables $y_u$ for each vertex $u$.
2. View $G$ as a weighted graph, where $y_u$ is the weight of the vertex $u$
3. For every pair $(u, v)$ of vertices that are supposed to be in different independent sets, add an edge between $u$ and $v$
4. For every pair $(u, v)$ of vertices that are supposed to be in the same independent sets, merge $u$ and $v$. The weight of the new vertex is the sum of the weight of the original vertices.

5. Compute the maximum weighted independent set of the graph. If its value is less than or equal to $1$, the solution is optimal. Otherwise, add this independent set to the set of variables.

To compute the maximum weighted independent set, we proceed in three steps:

1. We first use a greedy algorithm, presented in (Held et al., 2012), complemented with a local search (Diogo Andrade, 2008)
2. if the greedy algorithm does not find a solution of weight larger than $1$, we use the RQAOA algorithm presented in the previous section, complemented again with the same local search algorithm
3. if RQAOA does not find a solution of weight larger than $1$, we use an exact algorithm. For this the problem is transformed into the following integer linear program:

$$\begin{array}{llll} \max & \sum_u y_u z_u & & \\ s.t. & z_u + z_v & \leq & 1 & (u,v) \in E \\ & z_u & \in & \{0,1\} & u \in V \end{array}$$

   that is fed into the CPLEX software (Cplex, 2009). The problem is solved exactly, and therefore costly, which is why it is hoped that the heuristic will succeed before the exact algorithm is launched.

The result of each algorithm, if they succeed, is an independent set of the modified subgraph, that can be transformed into an independent set of the original graph by unmerging the nodes.

## 6.4 Branching

As explained earlier, branching is done using the Ryan-Foster rule. From a fractional solution:

1. Find a pair of vertices $u, v$ s.t. $0 < \sum_{I \in S' | u,v \in I} x_I < 1$
2. In case multiple such pairs exists, take the one where the sum is closest to an integer
3. Branch on whether the two vertices are in the same independent set or not.

# 7 Conclusions

This deliverable contains details about the specification and implementation of a Branch and Price Algorithm that uses QAOA as a subroutine to solve the coloring problem for graphs.

While a naive utilisation of QAOA provides results that are comparable with greedy algorithms, it is not good enough for industrial cases where an optimal, or close to optimal solution, is needed. The use of QAOA as a subroutine in a Branch and Price Algorithm, an exact solution, guarantees in our case that the optimal solution is indeed found.

The next deliverable D5.10 will consist in the precise implementation of this algorithm as a dedicated software.

## Bibliography

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., … Witzig, J. (2021). *The SCIP Optimization Suite 8.0* (Technical Report). Optimization Online. http://www.optimization-online.org/DB_HTML/2021/12/8728.html

Bravyi, S., Kliesch, A., Koenig, R., & Tang, E. (2020). Obstacles to state preparation and variational optimization from symmetry protection. *Phys. Rev. Lett.*, *125*.

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, *22*(4), 251–256.

Chaitin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E., & Markstein, P. W. (1981). Register allocation via coloring. *Computer Languages*, *6*, 47–57.

Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, *46*(53), 157.

Diogo Andrade, R. W., Mauricio Resende. (2008). Fast local search for the maximum independent set problem. *Journal of Heuristics*, *18*(4), 220–234.

Farhi, E., Goldstone, J., & Gutmann, S. (2014). A quantum approximate optimization algorithm. https://doi.org/10.48550/ARXIV.1411.4028

Feige, U., & Kilian, J. (1998). Zero knowledge and the chromatic number. *Journal of Computer and System Science*, *57*(187–199).

Garey, M. R., & Johnson, D. S. (1976). The complexity of near-optimal graph coloring. *Journal of the ACM*.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability - A guide to the Theory of NP-Completeness*. W.H. Freeman; Company.

Hadfield, S., Wang, Z., O'Gorman, B., Rieffel, E., Venturelli, D., & Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, *12*(2).

Held, S., Cook, W., & Sewell, E. C. (2012). Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, *4*, 363–381.

Johnson, S. G. (2007). The NLopt nonlinear-optimization package.

Kochenberger, G., Glover, F., Alidaee, B., & Rego, C. (2005). An unconstrained quadratic binary programming approach to the vertex coloring problem. *Annals OR*, *139*, 229–241.

Marx, D. (2004). Graph colouring problems and their applications in scheduling. *Periodica Polytechnica Electrical Engineering*, *48*(1–2), 11–16.

Ryan, D., & Foster, B. (1981). An integer programming approach to scheduling. *Computer Scheduling of Public Transport*.

Veshchezerova, M. (2022). *Quantum algorithms for energy management optimization problems* [Doctoral dissertation, Université de Lorraine].

Zuckerman, D. (2007). Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, *3*(6), 103–128.