Quantum Metropolis Solver: a quantum walks approach to optimization problems

Roberto Campos1,2 · P. A. M. Casares1 · M. A. Martin-Delgado1,3

Received: 12 December 2022 / Accepted: 11 June 2023 © The Author(s) 2023

Abstract
The efficient resolution of optimization problems is one of the key issues in today’s industry. This task relies mainly on classical algorithms that present scalability problems and processing limitations. Quantum computing has emerged to challenge these types of problems. In this paper, we focus on the Metropolis-Hastings quantum algorithm, which is based on quantum walks. We use this algorithm to build a quantum software tool called Quantum Metropolis Solver (QMS). We validate QMS with the N-Queen problem to show a potential quantum advantage in an example that can be easily extrapolated to an Artificial Intelligence domain. We carry out different simulations to validate the performance of QMS and its configuration.

1 Introduction
Optimization problems are frequently encountered in various contexts, such as determining the optimal selection of products to buy in the supermarket, based on their quantity and quality within a given budget (known as the knapsack problem Bretthauer and Shetty (2002)), identifying the best combination of public transportation options to minimize commute time (a variant of the Traveling Salesman Problem, TSP Hoffman et al. (2013)), or selecting the most interesting landmarks to visit a city during a limited vacation period (goal oversubscription problem Smith (2004)). These types of complex optimization problems are commonly faced by different industries in their routine operations. However, unlike daily optimization problems, industrial optimization problems often require significant computational resources to achieve an optimal solution because of the large number of possible combinations that need to be evaluated.

Optimization problems of special interest to the industry often involve multiple variables with a large number of dimensions and complex optimization functions, making it challenging to evaluate each possible configuration of problems. Typically, the function that is optimized is a valuable resource that can have significant economic implications for both individuals and companies. Examples of these types of problem include the routing problem, which involves identifying the optimal path between multiple locations Bektas and Laporte (2011); Kumar (2012); Toth and Vigo (2014), portfolio optimization in finance, which requires balancing risk and reward when deciding which products to buy and sell Markowitz (1968); Rubinstein (2002), and the protein folding problem, which aims to minimize energy by rotating the protein structure Kryshtafovych et al. (2019).

A common phenomenon for which these problems suffer is the “curse of dimensionality” Bellman (1956); Kuo and Sloan (2005) defined by Bellman. It occurs when the dimensionality of the data grows rapidly, leading to a significant increase in the volume of data. As a result, the data become scattered and difficult to cluster, which poses a significant challenge for problem solving.

The scattering of data caused by this phenomenon makes brute-force solutions impractical for larger optimization problems, as the size of the problem grows exponentially Kolaitis and Thakur (1994). Despite the fact that the procedure for solving larger problems only checks a minimal subset of all possible combinations, it is still necessary...
to evaluate numerous potential combinations. The selection of an appropriate technique to solve optimization problems can significantly affect the time required to obtain a solution. For example, it may be possible to reduce the time required to solve a problem from centuries to hours or days. This transformation can make a seemingly intractable problem solvable.

Optimization problems have been extensively studied from a theoretical perspective, and numerous toy problems have been developed as simplified versions of real-world problems with fundamental similarities. These problems serve as benchmarks for testing the performance of new algorithms on easily executable instances. Examples of such benchmark problems include the TSP Hoffman et al. (2013), the knapsack problem Bretthauer and Shetty (2002), and the N-Queen problem Gent et al. (2017), which are similar to the routing problem, the risk/reward finance problem, and the problem of selecting the best action to execute, respectively.

The fundamental aspect of optimization problems is the requirement to traverse numerous states with an associated value until the state with the minimum value is reached. The trial and error process, followed by refinement, can be automated using computer simulations to accelerate the problem-solving process. Consequently, it is crucial to establish an accurate correspondence between the real-world problem and the simulation.

Due to the complexity explained above, most complex optimization problems belong to the NP complexity class Crescenzi et al. (1995). Thus, polynomial advantages are often the best one may hope to attain. Quantum computing is a natural approach, based on the fact that quantum walks can achieve a quadratic speedup in hitting time over their classical counterparts Montanaro (2015).

2 Optimization algorithms and random walks

There is an extra difficulty in dealing with optimization problems, the representation. Some of them, such as the connections between cities in the traveling salesman problem (TSP), can be easily represented on a computer. However, others, such as modeling the air around the wing of an airplane, pose a more significant challenge, necessitating the use of simplified models. In certain cases, oversimplification may even be required due to the intractability of the original problem. This was historically the case with lattice models of protein folding Robert et al. (2021).

There are some different representation options to convert the problem into an algorithm-solvable instance. In this work, a four-element representation is chosen. The elements are as follows:

- States: Possible values the system can take for a given problem.
- Transitions: Possible states generated from each state.
- Evaluation function: Function to calculate the reward/value of each state.
- Goal: Objective of the problem, minimize or maximize the evaluation function.

In order to introduce the quantum algorithm that we have selected in this work, we have to explain some classical algorithms first. Classical random walks are not only very powerful, but they also form the basis for very widely used Monte Carlo algorithms, routinely used for optimization problems. The Monte Carlo method consists of a random sampling of state space to approximate a function. It works better with a larger population because the error classically decreases as $1/\sqrt{N}$ Daniell et al. (1984). The most relevant aspect of the Monte Carlo method is that it serves as a basis for optimization algorithms.

A related technique also based on random walks and inspired by statistical physics is the simulated annealing algorithm Kirkpatrick et al. (1983). The core concept is a search algorithm that always accepts transitions that lower the energy, but with a certain probability, it also accepts transitions to higher energies. The probability of moving to a higher energy state at the beginning of the execution is high because the simulated temperature starts warmer. However, as steps are executed, the algorithm goes cooler and the probability is reduced. That process helps the algorithm explore many states at the beginning, avoiding local minima. Because of this, the algorithm converges slowly to the minimum energy state.

Combining random sampling of the Monte Carlo method from a probability distribution and guided stochastic search of random walks and simulated annealing results in an algorithm called Metropolis-Hasting Metropolis et al. (1953); Hastings (1970). It is used to approximate a probability distribution $\pi_x$ by mixing it with a random walk $W$ until an equilibrium is reached, $W\pi_x = \pi_x$.

The Metropolis-Hastings algorithm requires three methods: (i) a procedure to sample initialization states, (ii) a procedure to propose state transitions, and (iii) an evaluation function that scores how good is a given state. The latter is often called “energy” $E$ due to its connection to statistical physics. It will determine the acceptance probability of the transition proposed at point (ii), $\min(1, \exp(-\beta \Delta E))$, where $\beta$ is a parameter called inverse temperature.

There exists a quantum version of random walks. Quantum walks can also be understood as a generalization of Grover’s algorithm Grover (1997). The first proposal, by Ambainis Ambainis (2004), was restricted to Johnson graphs. Soon, Szegedy presented bipartite quantum walks generalizable to any ergodic chain problem Szegedy (2004).
Both can be shown to offer Grover-like quadratic speedup in the hitting time, in other words, in the time required to find the marked item. The latter quantum walk has been widely used, for example, in the context of Quantum Metropolis algorithms Temme et al. (2011); Montanaro (2015). Also, Szegedy’s proposal has found a variety of applications Paparo and Martin-Delgado (2012); Paparo et al. (2013, 2014); Kadian et al. (2021).

Most of the previous quantum Metropolis algorithms assumed a slowly changing parameter $\beta$ and often phase estimation to evolve the state from the uniform superposition to the target stationary distribution Somma et al. (2008); Yung and Aspuru-Guzik (2012). However, this is different from the way classical algorithms operate, where $\beta$ is changed much more rapidly, and no additional techniques other than random walks are used. For this reason, Lemieux et al. proposed a quantum version of the Metropolis-Hastings algorithm that makes use of quantum walks heuristically, similar to how random walks are used classically Lemieux et al. (2020).

In this work, we discuss and analyze the behavior of the quantum Metropolis-Hastings (M-H) algorithm in an optimization problem that arises in the field of Artificial Intelligence (AI). To test the M-H algorithm and facilitate other users to use quantum M-H, we implemented a software tool called Quantum Metropolis Solver (QMS). Our tool uses as input a description of an optimization problem and generates the minimum-cost solution. Also, it has additional functionalities such as plotting, classical solution comparison, and deep analysis of quantum M-H algorithm solutions. The tool is open-source philosophy-oriented and was coded in Python using Qiskit modules. The tool code is public in https://github.com/roberCO/QMS-OSS.

QMS application is threefold. First, it can be used as a metric tool to test the performance of the quantum M-H algorithm in a concrete search problem. Additionally, QMS can be integrated into hybrid classical-quantum algorithms since QMS input is a classical problem description, but it can generate classical or quantum output. Finally, we compare classical and quantum variants to computationally assess potential quantum advantages.

3 The Metropolis-Hastings algorithm: classical vs. quantum

The M-H algorithm is a Markov chain procedure because the transition probabilities depend only on the current state, and it is a Monte Carlo technique due to the generation of a random sequence of samples from a probability distribution $g(x)$. The result of these two properties is a Markov chain Monte Carlo algorithm, able to rapidly mix and generate low-energy states.

The Metropolis-Hastings algorithm is used to sample the stationary state of the Markov chain $\pi_x$. As a starting point, a uniform random sample $x$ is generated. Then, new samples ($x'$) are generated from the previous sample using some generation function $g(x'|x)$ and accepted according to their energy differences. If the energy of $x'$ is lower than $x$, it is accepted; otherwise, an acceptance probability is calculated using an evaluation function $f(x)$ as $\alpha = f(x)/f(x')$. This process is similar to a random walk with steps dictated by $W$, preparing a stationary state $\pi_x$. Its stochasticity allows the algorithm to explore a larger search area and avoid getting caught in local energy minima. Figure 1 shows a scheme of this algorithm.

The M-H algorithm is an optimization technique with great utility in domains ruled by a minimization or maximization function and a well-known probability distribution to generate successors. Particularly interesting are problems with uncertainty in the output, in which the solution of the problem is not known, and the task is to find the lowest value/energy state. For example, finding the hyperparameter configuration of a deep learning architecture. In contrast to problems like routing, in which the final point is known and the task is to find the path between the initial and final point given by the problem definition.

The distinguishing advantage of M-H over other heuristic algorithms is the rapid generation of successors due to its stochasticity and its unique dependence on the previous state Zabinsky (2009). On the other hand, that high-speed generation may penalize the M-H algorithm with a poorly guided search far from the optimal, thus the M-H algorithm is better suited for highly complex and unstructured configuration spaces where the stochastic state generation is advantageous.

The process of finding a solution by iteratively trying different steps to refine the current state to an acceptable solution is an old and well-known process. It has been one of the foundations of human reasoning and problem solving. However, the big change comes when there are machines capable of performing thousands of these refinement steps quickly or even at the same time. Using this fast step execution, the M-H algorithm takes the brute-force philosophy and incorporates Markov theory to achieve an algorithm capable of testing many states with a minimum guided search. For this reason, one of the strengths of M-H lies in the ability to evaluate successive states quickly.

3.1 The classical MH method

The performance of the classical M-H algorithm is strongly influenced by two factors: mixing time (MT) of the Markov chain defined in Eq. 1 and Monte Carlo error. These two aspects determine the number of necessary steps to obtain an acceptable solution, and they are key in creating a quantum
version that enhances both. The mixing time of the Markov chain is the time it takes for a Markov chain to reach a stationary distribution. Ref. Boyd et al. (2004) explains the importance of this value optimization. In contrast, the Monte Carlo error is the distance between the calculated distribution at step $N$ and the stationary distribution. In Ref. Wolff and Collaboration (2004), there are examples to reduce the error of the Monte Carlo methods.

The Mixing Time $MT(\epsilon)$ can be interpreted as the minimum number of steps $t$ of the Markov chain that should be applied to any initial distribution $\pi_0$, such that the result is $\epsilon$-close to the stationary distribution $\pi$, under distance $D$ (Eq. 2).

$$MT(\epsilon) = \min\{t|\forall t > t, \forall \pi_0, D(P^t\pi_0, \pi) < \epsilon\}, \quad (1)$$

Distance $D(p, q)$ between probability distributions $p$ and $q$ is in turn defined as the sum of probability differences at each vertex of the Markov chain, under those distributions,

$$D(p, q) = \frac{1}{2} \sum_{v=1}^{N} |p_v - q_v|. \quad (2)$$

In this work, we have focused on an M-H algorithm that converges to the lowest energy state. However, since the successor generation in M-H is governed by a probability distribution function, it can also be used to sample the unknown stationary distribution of a Markov chain. The algorithm can generate samples from the probability distribution, which can be used to approximate a probability density function Yildirim (2012). Specifically, it can be applied to problems that prohibit complete enumeration of all paths Flötteröd and Bierlaire (2013). Again, this M-H sampling can be quantum versioned naturally, since the quantum circuit of the M-H can be executed repeatedly until getting a distribution of the result of each execution shot.

As explained above, the limiting factor to getting a speed-up with the M-H algorithm is the mixing time and the error reduction. Going one step further, it is possible to identify three points that we can optimize in the M-H execution: reduce the number of evaluated states, avoid getting stuck at local minima, and evaluate states faster. Quantum computing can help at these points as the eigenvalue gap of the quantum walk is quadratically smaller than the classical eigenvalue gap as explained in Magniez et al. (2011) with the formula $\Delta = \Omega(\delta^{1/2})$ being $\delta$ the eigenvalue gap of the classical walk and $\Delta$ the phase gap of the quantum walk. Although there are classical approaches to solve this problem Calderhead (2014), Grover’s algorithm and quantum walks add amplitudes instead of probabilities, and their difference ends up showing as a quadratic speed-up Szegedy (2004).
3.2 A quantum version of the MH method

A quantum version of the Metropolis-Hastings algorithm exploits a reduced complexity due to a smaller eigenvalue gap compared to its classical counterparts. This fact helps to infer the minimum energy state quicker. The essence of this advantage comes from the application of the quantum walk operator to an initial uniform superposition of all possible states so that the number of steps to mix the chain is reduced (Table 1).

Quantum walks can be understood as a generalization of Grover’s algorithm Galindo and Martin-Delgado (2000).

With two Grover-like reflections, Szegedy Szegedy (2004) constructs a quantum walk on a bipartite graph. However, in this work, we substitute the bipartite graph with a coin \( |c\rangle \) via an isomorphism Lemieux et al. (2020), which creates an entanglement with the states \( |s\rangle \). This produces a quantum walk \( |\Psi_1\rangle = |s, c\rangle \), where the states are represented as a superposition \( |s\rangle \) of possible states.

\[
|s\rangle = \sum_{x \in \mathbb{N}} \alpha_x |x\rangle \in \mathcal{H}_s, \quad (3)
\]

and the coin \( |c\rangle \) is in the coin space \( \mathbb{C}^2 \)

\[
|c\rangle = \{\alpha_1 |\uparrow\rangle + \alpha_2 |\downarrow\rangle\} \in \mathcal{H}_c. \quad (4)
\]

In Lemieux et al. (2020), a Szegedy quantum walk is used as a basis to construct the circuit of a quantum Metropolis-Hastings algorithm. In this work, we show the implementation complexity of \( W \) in a quantum circuit using unitary operators. The main challenge is the application of the operator and its inverse (unitary operator) in each \( W \) because the inverse of the operator depends on whether the change is accepted or not. In the ideal case, it would be necessary to apply a conditional inverse operator in each step. In the M-H algorithm, after a change is proposed, there are two options: accept or reject a proposed change, and this duality is a key problem in creating a unitary operator. The solution proposed by Lemieux et al. Lemieux et al. (2020) is a different unitary operator for \( W \) that is isomorphic to the original Szegedy walk operator \( U_W \) and is represented as

\[
\tilde{U} = RV^\dagger B^\dagger F B V,
\]

where \( R \) is the reflection operator, \( V \) is the move preparation operator, \( B \) is the coin operator, and \( F \) is the spin-flip operator. The operators \( F, B, \) and \( V \) can be seen as a direct quantization of the classical operations in the M-H algorithm of proposing a candidate and accepting/rejecting it depending on whether it minimizes or maximizes the value function. On the other hand, \( R \) is the only pure quantum operator for the quantum M-H algorithm that is used to generate Grover-like rotations with the potential to exhibit a polynomial advantage. All of these operators simplify the implementation to a circuit, if we compare with Szegedy’s implementation of \( W \).

Table 1 Algorithm of Quantum Metropolis-Hastings, detailing how the new candidate is proposed and the acceptance probability and random number are calculated. This algorithm executes several steps \( W \) and is represented as

### Algorithm 1: Metropolis-Hastings algorithm

| Initialize \( x_t \sim g(x) \) |
| for iteration step = 1, 2, …, \( W \) do |
| Propose: \( x' \sim g(x' | x_t) \) |
| Acceptance Probability: \( \alpha(x' | x_t) = \min(1, \frac{f(x')} {f(x_t)}) \) |
| Random variable \( r: r \in [0, 1] \) |
| if \( r \leq \alpha \) then |
| Accept the proposal: \( x_{t+1} \leftarrow x' \) |
| else |
| Accept the proposal: \( x_{t+1} \leftarrow x_t \) |
| end |

4 QMS: Quantum Metropolis Solver software tool

We present a software framework whose core is the circuit proposed by Lemieux et al. (2020) and build a library around the Metropolis-Hastings quantum algorithm, allowing any user to solve optimization problems with the quantum M-H algorithm. We call this software architecture Quantum Metropolis Solver (QMS). We implement it in a quantum simulator running on a classical computer. QMS pretends to be one step beyond the evolution of quantum walks and quantum M-H algorithm.

There are numerous papers on quantum walks. In our case, we want to continue a development path that we believe has two main previous points (and many intermediate ones), the proposal of quantum walks by Szegedy and the quantization of the M-H quantum algorithm by Lemieux et al. Starting from the proposed \( \tilde{U} \) operators, we have modified this algorithm so that it can solve optimization problems. For this, we have implemented all the circuits in the qiskit library Aleksandrowicz et al. (2019) and modified it to be able to read data from an oracle. We have also modified the ways of initializing the circuit, optimized certain operations and we have included the option of working with different betas that allow testing the algorithm with a quantum walk out-of-equilibrium. We have created an M-H quantum model capable of solving any optimization problem defined as a
set of states and values associated to these states. Our main contributions are as follows:

- **Software tool**: We give the community easy-to-use software to solve problems in which Metropolis-Hastings has a proven advantage.

- **Study of Quantum Metropolis-Hastings to an optimization problem related to Artificial Intelligence**: We provide evidence that the quantum advantage of quantum walks and the QM-H algorithm can be applied to Artificial Intelligence to optimize the search process inherent in any AI technique. The case study to validate this idea is the N-Queen problem, which has been used recurrently and is still used, as a benchmark for new classical AI algorithms.

- **Scaling law**: We analyze the performance of QMS by finding the scaling law for the N-Queen problem. In this way, we add another example of the application of the quantum MH algorithm to the previously analyzed case study of the Protein Folding problem Casares et al. (2022).

- **Comparison of different implementations of quantum walks**: We compared three different implementations of quantum walks and quantum Metropolis-Hastings on the N-Queen problem. The proposal of Szegedy (2004) uses a bipartite graph to search in the state space. This algorithm was later modified in Ref. Lemieux et al. (2020) with discrete operators and substituting the $n$-dimensional search with a binary search performed with a coin. Finally, we also compared the results with an operator ordering different from the qubitization method in Low and Chuang (2019).

- **Quantum walk out-of-equilibrium**: In QMS, we included the option to have different beta $\beta$ schedules. This $\beta$ appears in Eq. 6 and is related to the probability of acceptance of the proposed change. The point of varying the beta value during the quantum walk execution is to have a quantum walk out-of-equilibrium and get the Markov chain mixed faster than a conventional quantum walk.

- **Oracle**: We designed a subroutine to load all energy/value data into the quantum walk. It allows us to execute larger problems with precalculated values.

- **Deltas**: We preprocess all values associated with the states to reduce the complexity of the circuit and the qubits needed to represent them. We calculate the difference between the state values and encode it into ancilla registers of the oracle.

- **Initializations**: We designed our software tool to create quantum circuits for different initializations of the problem. For example, we can start with an equiprobable distribution in the states, even if the number of states is not a power of 2, or with a function defined by the user.

We can work with problems with or without boundary conditions, etc.

The underlying motivation for implementing QMS is to give the scientific and industrial community a test-bed to solve optimization problems with quantum algorithms, including a comparison with its classical counterpart. This software tool abstracts the inherent complexity of implementing quantum algorithms. Users can define a problem to solve with just an input file. In this file, the problem is defined as a set of states and associated costs. Then, QMS will return the state with the least cost as a solution. It is assumed that the states are connected among them in a graph so that a Markov chain algorithm can be applied.

QMS has been designed not only to offer a final result, but also to show statistics that help to understand the performance of the algorithm. There are three possible outputs of QMS: the minimum cost state, the TTS result, or the probability distribution. Stressing the point of a test-bed, QMS allows a TTS comparison between the quantum algorithm and its classical version. In such a mode, the same problem is executed in a classical Metropolis-Hastings algorithm and a quantum one, and both minimum TTS achieved by each is shown. Furthermore, a plot with the TTS curve is generated as a function of $t$ in (9) for both the quantum and the classical M-H algorithms.

The classical M-H module for comparison with quantum M-H has also been implemented by us. It is exactly the counterpart of the quantum Metropolis-Hastings. We implemented a classical random walk following the M-H algorithm to decide if the change is accepted or not. It takes the same input as the quantum walk and performs a search with the same number of steps ($W$s). Then, the classical M-H is executed $n$ times to also get the probabilities of having one state or another as a result. Therefore, we execute the classical M-H $w$ by $n$ times.

Our software tool, detailed in Fig. 2, has been thought of as a user-friendly library that simplifies its use. The user simply defines the problem in an input JSON, a file that stores simple data structures and objects in a standard format. Then, it is possible to configure the QMS execution with a configuration file where some parameters are defined. For example, the number of steps, which is equivalent to the number of times that operator $W$ is applied, can be tuned just by modifying the value of the parameters initial_step and final_step. Any number of steps in between will then be analyzed.

The core of QMS is an implementation of the circuit proposed in Lemieux et al. (2020) with some optimizations and modifications. From Eq. 5, the operators are the same:

- $V$ is move_preparation: This operator creates a superposition of all possible state transitions.
Fig. 2  Scheme of the QMS architecture. It receives a JSON file with a description of the possible states and the values associated with them. Then, the deltas between states ($\Delta_{ij}$) are calculated and sent to the quantum M-H module. This QM-H module constructs an initialization circuit to obtain the initial state $x_t$. In addition, the operator circuit $W$ is generated many times with a fixed parameter. Once the circuit is created, QM-H executes the circuit. The results obtained after execution using raw amplitudes are processed to get a plot with the evolution of the TTS, a numeric TTS value, or the probabilities. In parallel with the quantum M-H execution, QMS has the option to execute a classical M-H, to compare both versions of the M-H algorithm. This classical M-H module has the same structure and connections with input and output as its quantum counterpart.

- $B$ is coin preparation: This operator creates the superposition of the coin and rotates the coin in those states where the change is accepted.
- $F$ is conditional move: This operator performs the change in the states in which the coin has been rotated. This is the M-H acceptance/rejection step.
- $R$ is reflection: Similarly, a Grover reflection is used for amplitude amplification of the marked states.

However, the representation of the states (also called register) done in previous work has been optimized. Previous work used a representation of $N$ qubits to represent $M$ possible registers to move, being $N = M$, so one qubit for each register. With this representation, all registers had a 0 value except the register to move that had a 1. We substitute this unary representation in the move register with a binary representation, using only $\log_2(M)$ qubits, where $M$ is the number of registers. That change allows us to reduce the number of qubits in the whole system. Furthermore, the heuristic that guides the algorithm is based on an inverse temperature parameter $\beta$. The acceptance value comes from the condition:

$$A_{ij} = \min \left( 1, e^{-\beta (C_j - C_i)} \right),$$  \hspace{1cm} (6)

where $A_{ij}$ is the acceptance ratio of the proposed changed, $\beta = 1/T$, $C_j$ is the candidate cost, and $C_i$ is the old state cost. It implies that if the cost of the candidate is lower, the change is accepted. Otherwise, the change is only accepted with exponentially decreasing probability in the inverse temperature.

The input file of QMS is a set of tuples (state, cost). States are represented with a binary notation starting from 0 and cost is a real number. An example tuple could be [(101): 65.53], 101 is the states in binary notation, and 65.53 is the cost.

For internal representation, QMS requires $\lceil \log_2(n) \rceil$ qubits to represent $n$ input states. The cost associated with each state is not directly represented; on the contrary, what is stored in QMS is the cost difference between the connected states, represented as $\Delta$. $\Delta_{ij}$ is set to 1 if the cost of candidate state $j$ is lower than the cost of the state $i$. Otherwise, $\Delta_{ij}$ stores a codification of the cost difference between the states, as shown in this equation:

$$\Delta_{ij} = \begin{cases} 1, & E_j < E_i, \\ e^{-\beta (E_j - E_i)}, & E_j \geq E_i, \end{cases}$$  \hspace{1cm} (7)

this codification corresponds to the probability of change considering the $\beta$ of the step. Then, all probabilities are stored in a QRAM as an oracle that can be accessed in each $W$.

We define four registers to store all information in QMS:

- **States** $|\cdot\rangle_S$: Contains each possible state affected by operator move preparation ($V$).
- **Move** $|\cdot\rangle_M$: Contains the candidate move state affected by operator move preparation ($V$).
- **Coin** \( |\cdot \rangle_C \): Coin affected by operator coin preparation \((B)\).
- **Oracle** \( |\cdot \rangle_O \): Contains the acceptance probability between all connected states affected by the conditional move of the operators \((F)\) and the reflection \((R)\).

The move register \( |\cdot \rangle_M \) can be divided into two registers: move id register \( |\cdot \rangle_M^{|i} \) to know which state is going to be changed and move value register \( |\cdot \rangle_M^{|v} \) that indicates how the register is to be changed (+1, -1, swap left, swap right, etc.).

In any optimization problem, it is critical to choose the initial state distribution. The gap between the initial state and the goal state determines the execution time or the quality of the solution. QMS receives an input set of states from which it has to find the least energetic one. QMS takes an initial point and applies the W operator the number of times defined by the user. The state reached after the application of the W operators is the solution returned by QMS. For that reason, our library allows different initial states to leave to the user the freedom of selecting at which point the search should start. It is even possible not to select just a point but rather a probability distribution that will determine the initial point for the search. The preparation of such a probability distribution is carried out by a process called initialization.

- **Fixed:** QMS starts the search in a state selected by the user. This option is useful for refinement problems in which there is a first approximate solution, and the optimization consists of a search for a more quality solution close to the initial approximation.
- **Random:** QMS starts in a uniform superposition of states that is equivalent to a random state because no one has more probability than others to be selected.

QMS initialization can also be classified by how states are generated, as follows:

- **Sequential:** This mode generates candidates sequentially, simply adding or subtracting one unity from the coordinate. For example, if the problem is a pawn moving onto a chess board, the states could be represented as the row and column occupied by the pawn (column, row). So, if the pawn is in \((4,5)\), one possible candidate is to add 1 position in the column, resulting in a state \((5,5)\).
  
  - **Circular:** This option allows connecting frontier states with periodic boundary conditions. For example, if the previous pawn is in position \((5,7)\), so it is in the upper row, it would be possible to add one row to place the pawn in the lower row, state \((5,0)\).
- **Non-circular:** This option does not allow connecting frontier states, so it is configured with non-periodic boundary conditions. For example, if the pawn is in the row frontier \((5,7)\), it is not allowed to sum 1 in rows.
- **Swap:** This mode generates candidates exchanging coordinates, and it does not allow collisions. For example, if there are 3 queens placed on the board and each queen is in one column, such that there are no two queens in the same column. The state will represent the row of queens. One possible state is \((1,0,2)\), first queen in row 1, second queen in row 0, etc. The candidates are generated by exchanging queen positions; for example, a candidate switching the first and second queens would result in \((0,1,2)\), which means the first queen in row 0, the second queen in row 1, etc.

QMS is a software tool with a white-box design and a modular architecture. It means that it is made up of an aggregation of modules that receive input, process it, and serve the result to other modules. These modules can be analyzed, modified, or even replaced by other modules with the same data interface following similar input/output format rules. This open and flexible architecture is an advantage that can enhance the use of the tool by the community because it is easy to understand, including new functionalities that may be added in the future or fixing bugs. QMS architecture is detailed in Fig. 2.

The whole architecture has been implemented using Python3 because it is an open-source language and very used by the developer community. The quantum module calls Qiskit and Qiskit-Aer for the quantum simulations on the classical computer. Qiskit is also an open source Python module that simplifies circuit creation and simulation, and it has proven good performance with large circuits with more than 25 qubits and multi-thread execution, as can be seen in Fig. 11 of Suzuki et al. (2021).

### 5 Case study: Quantum Artificial Intelligence

Any Machine Learning (ML) algorithm modifies its internal state and the world representation following a deliberative process. This process is based on reasoning about the input data to obtain a knowledge model of the problem. During the reasoning process, the system generates different hypotheses to explain the environment and execute the correct action. For example, adjusting connection weights in an artificial neural network or modifying the value of the action in a Reinforcement Learning agent.

The hypothesis generation and selection steps require a fast search in the state space (hypothesis space) to evaluate
which of all hypotheses available fits better with the problem to be solved. This search process is one of the bottlenecks of any ML algorithm, and quantum computing can speed it up. For that reason, we consider QMS to help in the Artificial Intelligence domain, improving the performance of the search process. We decided to validate our tool on a problem that has been extensively used as an AI benchmark, the N-Queen problem. Since this problem is in essence a search problem with multiple similarities with the search of an AI algorithm, any algorithm that gets a good performance in N-Queen can be easily adapted to other AI problems based on search, as a search of hypothesis in an ML algorithm.

The N-Queen problem is a spin-off of the classic chess problem Bowtell and Keevash (2021). The N-Queen goal state is a chess board of \( n \) rows and \( n \) columns with \( n \) queens such that no queen attacks the others. Therefore, there are no two queens in the same row, column, or diagonal as shown in Fig. 3. Many solutions have been proposed to the N-Queen problem Luria and Simkin (2021); Simkin (2021).

The N-Queen problem has been studied by many authors as a NP-complete problem Khan et al. (2009); Crawford (2016); Güldal et al. (2016). However, this categorization has not yet been clearly demonstrated Gent et al. (2017). The study that has the most detailed description of the complexity of the N-Queen problem places it in beyond the P class Hsiang et al. (2004). However, Gent et al. (2017) showed that N-Queen completion, a variation of N-Queen in which some queens are preplaced and the challenge is to place the others, is a NP-complete problem. This work was used by Torggler et al. (2019) to propose a quantum solution for a different variant of N-Queen, excluded diagonals, which is also NP-complete.

This kind of reasoning, searching for a hypothesis that explains and generalizes input data, can be seen as an abstraction of general knowledge from concrete examples, and it is known as inductive reasoning. As it is explained in Ref. Russell and Norvig (2010), the inductive reasoning goal is to find the hypothesis with the best balance between the classification of the existing examples and the generalization of the new examples. The search of this hypothesis with the best reward generalization/classification is equivalent to the N-Queen search, and, for this reason, N-Queen is a typical problem used as a benchmark in classic AI papers Gent et al. (2017). For example, this problem was used to introduce the backtracking search Walker (1960).

During environment interaction, any rational agent generates a set of hypotheses or associations from input data, which is similar to human learning from the environment or describing a problem. Then, it searches for which hypothesis is the best to explain the world. This process is known as a search in a decision tree. Sometimes, the agent selects one hypothesis and adapts it to new input data, but this is similar to a search of variations around the fixed point performed by QMS. Both symbolic learning (e.g., using first-order logic) and non-symbolic (e.g., using weights in a neural network) require the hypothesis space search to find the best next decision to take.

That process is similar to most machine learning algorithms. As Mitchell describes it in Ref. Mitchell (1997), the process of learning can be understood as a searching task in a large space of hypotheses and the search for the hypothesis that best fits the training and upcoming examples. This is the main reason why if it is possible to speed up the search process using quantum computing, it is possible to get advantages in AI algorithms. The reason is that what is commonly called the “learning process” is just the process of searching for the best explanation (hypothesis) for the data received, trying to be as ready as possible for future data entering the system.

Since the final state of the problem solved by QMS is unknown because it has uncertainty in the output (the task is to find an unknown configuration with minimum cost), the QMS algorithm makes a state-space search guided by the objective of minimizing a cost function. As Knuth describes, there is a type of searching based on comparing keys (values) of states Knuth (1998). In our case, we perform an informed

---

Fig. 3  N-Queen problem explanation with 8 queens in a chessboard of \( 8 \times 8 \). On the left is a valid solution for the problem because no queens are attacking one another as is explained for queen in D3. On the right is an example of the same chess board but with 4 queens attacking each other (A4, C5, E7, F6, and G4)

---
search (guided by a heuristic) that compares the different heuristic values of the states and a sequential search because it is necessary to perform the search moving the queens in a certain order. Of course, the sequential concept is just a formalism to describe the search because quantuminly it is done in superposition. These are the reasons why we view QMS applied to N-Queen as a search algorithm.

An AI agent does a similar search to optimize its interaction with the environment to speed up its goal attainments. Due to the similarities between the search in hypothesis space and the QMS search around an initial state (initial hypothesis), it is possible to understand QMS as a technique that implements one of the most fundamental steps in any Artificial Intelligence agent.

Another feature of the N-Queen problem solved with QMS is the transition probabilities. Since we are solving the problem with a technique based on Metropolis-Hastings, the probability of transitioning to one state or another from the current state is of particular relevance. According to Eq. 6 and Eq. 7, the probability will always be one that the heuristic value is reduced (the number of attacks between queens is lower) and a value between 0 and 1, otherwise. This probability will be close to zero. In addition, this probability can be regulated using a coefficient of $\beta$.

As explained in Section 4, QMS requires an input file which is a set of tuples (state, cost). In the N-Queen problem, the states are all possible combinations of boards that can appear for a given $n$. To reduce the size of the state space, we represent each state as a list with $n$ positions (one per queen), and each position represents the row of the queen. Assuming that it is not possible to have two queens in the same column, a movement is just a permutation in the position (row) of two queens. The scheme shown in Fig. 4 will be represented with the list (0, 1, 2, 3) indicating that the first queen is in row 0, the second one in row 1, etc.

This representation is an efficient codification of the problem that reduces the number of states. Possible movements are swaps between positions. For example, in Fig. 4, the first and second queens are swapped between them. Once the codification of the board has been explained, it is necessary to explain the heuristic value associated with each position on the board. The heuristic that we designed counts the number of attacking queens and penalizes them with extra cost if one queen attacks more than one other queen. Heuristic ($H$) is defined by Eq. 8, and the objective is to minimize the heuristic value. If the heuristic is 0, this board configuration is a solution.

$$H = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} [\delta_{\text{row}_i, \text{row}_j} + \delta_{\text{diag}_i, \text{diag}_j}] \cdot \gamma, \quad (8)$$

where the first sum counts the heuristic value for all queens, and the second sum counts the heuristic value for each queen. $\delta_{\text{row}_i, \text{row}_j}$ is 1 if $\text{queen}_i$ and $\text{queen}_j$ are in the same row (or same for diagonal), else it is 0. $\gamma$ is an accumulative value that counts the number of queens that $\text{queen}_i$ is already attacking. It is a multiplicative factor, which is increased by 1 for each extra attacked queen. $\gamma = 1$ for the first attacked queen by $\text{queen}_i$, $\gamma = 2$ for the second attacked queen by $\text{queen}_i$, etc. This heuristic returns a high penalty for boards in which a queen is very badly placed, such that, it is attacking multiple other queens, which it is something it is necessary to avoid.

In the N-Queen problem, the number of solutions for each size $n$ determines its complexity. A higher number of solutions implies more goal states and a more guided search because the gradient between states is bigger and the transition to the goal state is faster. Results in faster solution generation. In Table 2 extracted from Number solutions of the n-queen problem (2022), it is possible to see that $n = 6$ has significantly fewer solutions than the previous case, $n = 5$, and the next case, $n = 7$. This affects the complexity of the problem. As can be seen from the simulations plot in Fig. 5, the TTS obtained for $n = 6$ and $n = 7$ is similar, despite the fact that for $n = 7$, the state space is much larger than for $n = 6$.

6 Simulation results

To validate the QMS tool with the N-Queen problem, we execute different simulations with both classical and quantum Metropolis-Hastings algorithms. These simulations were
Table 2  Table of number of solutions for N-Queen problem by n. In general, the number of solutions is increased with the number of n, except in n = 6 that it is reduced which affects the complexity

| n  | Number solutions |
|----|------------------|
| 4  | 2                |
| 5  | 10               |
| 6  | 4                |
| 7  | 40               |

executed using the framework Qiskit Aleksandrowicz et al. (2019) and the included free noise simulator QASM. The N-Queen problem requires many qubits to represent its states, and the actual simulator has reduced capacities to represent large amounts of data. For that reason, we calculated the number of qubits that our codification needs and the memory consumption in the QASM simulator for the number of qubits. In Table 3, the number of qubits necessary for each register in the QMS software tool is detailed.

The list of necessary registers is as follows:

- **Coordinates** register represents each state. It is necessary to codify each state in binary form, so the number of qubits is the number of registers multiplied by the qubits necessary for the binary representation.
- **Move id** register represents the coordinate to move, so it is an index that requires a binary representation of the number of registers.
- **Move value** register indicates whether the movement is up or down (left or right in the swap case). It only requires one qubit.
- **Coin** register represents the binary decision of acceptance or rejection of the proposed candidate.
- **Ancilla** register is used to store the change probability of the proposed candidate. It can be represented with 3 or more qubits depending on the selected precision in probability.

Table 3  The number of qubits for each register in the QMS software tool

| N-Queen codification | qubits | memory RAM |
|----------------------|--------|------------|
| Coordinates          | n * \([\log_2(n)]\) |
| Move id              | \([\log_2(n)]\) |
| Move value           | 1      |
| Coin                 | 1      |
| Ancilla              | 3      |
| Total                | \(n * [\log_2(n)] + [\log_2(n)] + 5\) |

Table 4 represents the number of qubits for each size n in the N-Queen problem. This table is useful to understand the complexity of the circuit and the necessary resources to execute it on a classical computer. In our case, we have 128 GB of RAM available, but we only have results of \(n = 7\) due to the execution times that are around 2 weeks per instance of the problem with \(n = 7\).

We execute the simulations using the TTS metric, explained in Eq. 9, as a figure of merit. We test QMS for \(n = 4, 5, 6, 7\). The decision to stop at \(n = 7\) is directly related to the time consumption of each execution. To get more cases of the problem with different initial configurations, we slightly modify the N-Queen rules. In each instance, we fixed one queen in one position, considering that this queen is stuck in the position for any reason. It is common to have this kind of restriction in an ML problem as, for example, a mandatory point to visit in a route generated with Deep Learning. This new rule gives us extra points to evaluate our problem.

Table 4  The number of qubits and memory RAM consumption of QASM simulator for each size n instance problem

| QMS  |         |         |
|------|---------|---------|
| N    | Qubits  | RAM Memory |
| 4    | 15      | 0.1 GB   |
| 5    | 23      | 0.8 GB   |
| 6    | 26      | 1.5 GB   |
| 7    | 29      | 8 GB     |
| 8    | 32      | 65 GB    |

Fig. 5  Comparison between the classical and quantum TTS for N-Queen problem with n=4, 5, 6, and 7 with 74 samples of the problem. The dashed gray line separates the spaces of the classical advantage (upper triangle) and the quantum advantage (lower triangle). The key aspect to notice in this figure is that for n = 4, most points are in the classical advantage region, but when the problem increases its difficulty, most points are in the quantum advantage region. The scaling exponent is 0.939
Without this N-Queen modification, we would only have 4 different samples of the problem, one per \( n \) value. In the simulation, we have 74 different instances of the N-Queen problem with 9 instances for \( n = 4 \), 42 instances for \( n = 5 \), 21 instances for \( n = 6 \), and 2 instances for \( n = 7 \). The reduced number of instances for \( n = 7 \) results in 4 weeks of execution.

The metric proposed and used by Lemieux et al. (2020) is called Time To Solutions and denoted as TTS. It is a figure of merit that measures the expected number of steps required to find a solution. It is helpful to compare procedures that need to be repeated in case of failure, like this sampling algorithm. TTS strikes a balance between probability increase and the number of steps in each execution, which means that lower TTS implies less expected execution time.

\[
TTS(t) := t \frac{\log(1 - \delta)}{\log(1 - p(t))},
\]

where \( t \) is the number of steps executed, \( \delta \) is the success probability, and \( p(t) \) is the probability of hitting the ground state after \( t \) steps. With this metric and a scaling law exponent analysis, Lemieux et al. got a polynomial speedup of 0.75, e.g., classical TTS = \( O(\text{quantum TTS}^{0.75}) \), arguing that their proposal scales better than the classical Metropolis-Hastings and can thus be advantageous in bigger problem instances. The exponent indicates how the relationship between the classical and quantum algorithms scales. Lower than 1 means that quantum complexity scales more favorably than classical. We can estimate this exponent with a linear least-square fitting in the logarithmic scale for both classical and quantum minimum TTS. Since we want to see the scaling law exponent of quantum TTS against classical TTS, we follow the equation \( y = bx^a \), \( x \) and \( y \), classical (cTTS) and quantum (qTTS) TTS, respectively. In logarithmic scale:

\[
\log(qTTS) = \log(b) + a \log(cTTS),
\]

being \( a \) the exponent to define the scaling between \( qTTS \) and \( cTTS \). This exponent defines three regions:

\[
a = \begin{cases} 
> 1 & \text{quantum TTS > classical TTS}, \\
1 & \text{quantum TTS = classical TTS}, \\
< 1 & \text{quantum TTS < classical TTS}.
\end{cases}
\]

The results are shown in Fig. 5. This plot shows the relationship between classical and quantum TTS. It is divided into two triangles by a gray dashed line. The upper triangle shows the classical advantage region where the majority of \( n = 4 \) points are and the lower triangle with quantum advantage where all the points of \( n = 6 \) and \( n = 7 \) are. The plot shows that the resulting points present a tendency to move toward the region of quantum advantage as the problem size \( n \) increases, and we can conclude that there is a possible quantum advantage of quantum M-H against classical M-H. We can quantify it using a linear least-squares fitting with an exponent \( a \) (defined in Eq. 11) of 0.939.

We also test the core of QMS, quantum walks, to find the best performing algorithm. Due to the discretization carried out by Lemieux et al. in the quantum M-H unitary operator \( \tilde{U} \), we test whether the sorting of the operators could affect the results. We also include two other alternative sorting options in the comparative. We define Preparation with the operators \( VB \), Selection with the operator \( F \), Inverse Preparation with \( B^\dagger V \), and Reflection with \( R \).

- **Lemieux et al.** Prepared-Selection-Inverse Preparation-Reflection. Namely, it corresponds to the sorting:

\[
\tilde{U} = RV^\dagger B^\dagger F BV.
\]

- **Qubitization**. Explained in Low and Chuang (2019). Inverse Preparation-Reflection-Preparation-Selection. Namely, it corresponds to the sorting:

\[
\tilde{U} = F V B R B^\dagger V^\dagger.
\]

- **Alternative**. Selection-Inverse Preparation-Reflection-Preparation. Namely, it corresponds to the sorting:

\[
\tilde{U} = V B R B^\dagger V^\dagger F.
\]

To compare these three different sorting options, we execute the N-Queen problem with \( n = 4 \), 5, and 6 including several instances of fixed queens. For each sorting and each \( n \), we get the mean and standard deviation. Using these parameters, it is easy to compare whether the TTS have significant differences between them. We show the results in Fig. 6. The operator sorting election selected by Lemieux et al. achieves a lower TTS value for all the problem sizes tested. In addition, it is possible to observe a similar tendency between the different sorting options, but separated by a gap. Thus, these simulations show that the Lemieux et al. sorting works better.

### 6.1 Time complexity analysis

In order to prove the superiority of the quantum Metropolis-Hastings algorithm, we need to carry out a detailed analysis of several factors. It is possible to see two clearly differentiated point clouds in Fig. 5, one corresponding to the instances of 4 and 5 queen that are small size instances of the problem and the other corresponding to 6, 7, and 8 queen corresponding to medium size instances. We observe that quantum algorithms have a small computational overhead over classical ones, which penalizes them with small optimization problems.
to the 4-queen problem are in the classical advantage region (above the dashed line) and points in the 5-queen problem are in a transition region between the classical and quantum advantage region (over the dashed line). This penalization due to computational overhead is compensated with the bad classical scalability for larger problems (6, 7, and 8 queen instances) for which all points fall in the quantum advantage region.

Since we are conducting simulations with small and medium size problems, we run the risk that the quantum advantage will be hidden by the quantum computational overhead of small problems. To avoid that, we perform a scaling exponent study. This exponent $\alpha$ is a linear least squares fitting on the logarithmic scale of both values (quantum and classical TTS). In Eq. 11, the fitting exponent analysis shows how it is expected that QMS behaves with larger instances of the problem ($\alpha < 1$).

Theoretically, an algorithm based on quantum walks has a quadratic advantage as an upper bound. Before the simulations, we calculated a range of the exponent value between $0.5$ and $1$. After the simulations, we observed an exponent of value $0.939$ which being less than $1$ indicates that there is a quantum advantage that will become larger as the size of the problem increases.

It is important to understand that we get this exponent based on our figure of merit, TTS. Therefore, we want to show that we are able to observe a quantum advantage by analyzing TTS. The first remark is that, if the exponent trend is sustained for larger problems, there are polynomial advantages; in fact, we observed that the exponent is improved (lowered) each time we test a larger problem.

The maximum problem size allowed by the quantum simulator is $2^{12}$ (8-Queen). However, using classical computers, N-Queen has been solved for much larger instances. For example, if we want to solve the 60-Queen problem, the state space using our representation would be $2^{180}$. If we want to analyze the difference using TTS between classical and quantum algorithms, we need to know how TTS grows in both cases according to the problem size ($\log$ (problem size) vs $\log$ (TTS)). For classical TTS, the scaling factor is 7.89034, and for quantum, it is 7.13895, so the relation between them is 0.904. Extrapolating these values to the 60-Queen problem, we get a difference of TTS equivalent to 40.551 faster the quantum over the classical M-H solution.

7 Conclusions and outlook

We have studied quantum optimization algorithms to test how they could challenge existing classical algorithms for industrial problems. Classical optimization algorithms have been contributing to find solutions to hard problems that are otherwise impossible to solve by computers with brute force. However, classical optimization algorithms have limitations in scalability and can be optimized with quantum computing. Specially, we have focused on the Metropolis-Hastings algorithm that has many applications for industrial problems and its quantum counterpart, the quantum Metropolis-Hastings.

A quantum version of M-H was proposed by Szegedy (2004) and modified to obtain an implementable version by Lemieux et al. (2020). We use both works to construct a software tool that has the M-H algorithm at the core and which can solve any optimization problem given in simple format as a list of state-cost tuples. This tool is an easy-to-use Python module that receives a description of the problem and returns the minimum cost position. Besides, QMS evaluates to return a Time To Solution (TTS) value of the search to find the solution. This TTS metric is a figure of merit in evaluating the performance of the tool and comparing it with the classical algorithm.

As we have shown, it is possible to use QMS with any combinatorial optimization problem that has uncertainty in the output. Thus, the goal is to find an unknown state with some properties. This requirement is met in most optimization problems at the industrial level (knapsack problem, TSP, routing, etc.). The search process to find the state with the minimum cost in these problems converts them into an NP-complexity problem. It is in this family of problems in which quantum computing polynomial advantages can be the most useful, that is the reason why we selected them. The works by Szegedy and Lemieux et al. show a polynomial advantage using quantum walks, which we extrapolate to a general-purpose tool for any optimization problem.
We validate our QMS quantum software tool in the Artificial Intelligence domain. It is well known that one of the bottlenecks in Machine Learning algorithms is the search process that the algorithm performs to find the explanation that best fits the input data. This search is very similar to the process used to solve an optimization problem, as has been explained in the literature Russell and Norvig (2010); Mitchell (1997). Therefore, we consider that QMS could be applied to speed up some processes of an Artificial Intelligence algorithm.

Since we want to show how QMS can help AI algorithms using quantum search, we identify the N-Queen problem as a good benchmark for this task. The N-Queen problem is considered a benchmark for AI Gent et al. (2017); Russell and Norvig (2010) and can also be solved by a quantum algorithm, as we show in this work. In the simulations, we observe that the quantum algorithm gets better results than its classical counterparts. We also perform an analysis of the scaling with a linear least-squares fitting, getting an exponent of 0.939. Although the classical Metropolis-Hastings algorithm is not the state-of-the-art procedure to solve the N-Queen problem, we emphasize that our goal is to show the quantum advantage that QMS can get in a search problem. This N-Queen problem case study for QMS is added to the case study we proposed in a previous paper Casares et al. (2022), also with quantum advantage.

Another study we carried out was to understand why the discrete operators were sorted in a non-standard way Lemieux et al. (2020), and we have also compared the TTS results for different sorting options. We again used the N-Queen problem as a benchmark to test the $\hat{U}$ operator defined by Szegedy Szegedy (2004), Lemieux et al. Lemieux et al. (2020), and Low et al. Low and Chuang (2019).

Finally, future work should include more case studies to test QMS tools and possible applications. It would be interesting to analyze in multiple domains if the exponent is always above 0.5, which is the value required for a quadratic advantage.

Acknowledgements R.C. and P.A.M.C contributed equally to this work. We acknowledge support from the CAM/FEDER Project No.S2018/TCS-4342 (QUITEMAD-CM), Spanish MINECO grants MINECO/FEDER Projects. PGC2018-099169-B-I00 FIS2018. PID2021-122547NB-I00 FIS2021, the “MADQuantumCM” project funded by Comunidad de Madrid and by the Recovery, Transformation, and Resilience Plan - Funded by the European Union - NextGenerationEU and Ministry of Economic Affairs Quantum ENIA project. M. A. M.-D. has been partially supported by the U.S. Army Research Office through Grant No. W911NF-14-1-0103. P. A. M. C. thanks to the support of a MEC grant FPU17/03620, and R.C. the support of a CAM grant IND2019/TIC17146.

Author Contributions R.C. and P.A.M.C contributed equally to this work. M. A. contributed and supervised the work.

Funding Described in the acknowledgments.

Data availability The code has been placed on the GitHub platform (https://github.com/roberCO/QMS-OSS).

Declarations

Ethics approval and consent to participate Not applicable.

Human and animal ethics Not applicable.

Consent for publication This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Conflict of interest The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Bretthauer KM, Shetty B (2002) The nonlinear knapsack problem—algorithms and applications, European Journal of Operational Research, vol. 138, no. 3, pp. 459–472
Hoffman KL, Padberg M, Rinaldi G, et al (2013) Traveling salesman problem. Encyclopedia of operations research and management science, vol. 1, pp. 1573–1578
Bektaş T, Laporte G (2011) The pollution-routing problem. Transportation Research Part B: Methodological 45(8):1232–1250
Bellman R (1956) Dynamic programming and lagrange multipliers. Proceedings of the National Academy of Sciences 42(10):767–769
Kumar SN, Panneerselvam R (2012) A survey on the vehicle routing problem and its variants, Intelligent Information Management, 2012
Boyd S, Diaconis P, Xiao L (2004) Fastest mixing Markov chain on a graph. SIAM review 46(4):667–689
Bretthauer KM, Shetty B (2002) The nonlinear knapsack problem—algorithms and applications. European Journal of Operational Research 138(3):459–472
