Information Theory and Computation

Study of computer science NP problems through an Ising formalism via implementation of quantum optimization and Monte Carlo algorithms

A. Braghetto[*], G. Pompeo[*]

April 19, 2020

Abstract

Context – In [1], several NP (non-deterministic polynomial-time) problems belonging to the domain of Computer Science are analyzed, using Ising-like hamiltonians as a mean of formalizing their description.

Aims – In this paper, we wish to study some of these problems in depth and solve them by finding the ground state of their hamiltonians. In particular, we will focus on the graph partitioning, the vertex cover and the traveling salesman problems; all of them have an underlying description through graph theory.

Methods – These studies will be carried out employing three algorithms written in Fortran90. On one side, we have adiabatic quantum optimization (AQO), which is based on the adiabatic quantum theorem; on the other side, we use Monte Carlo methods in the form of simulated annealing (SA) and simulated quantum annealing (SQA) using the Trotter formula.

Results – We show that AQO has a significant computational time burden and it is also less effective in optimizing the spin systems to ground state. Simulated annealing proves to be effective for simple initializations, but in general its quantum counterpart shows more reliable results at a bearable computational cost.

Keywords: NP problems – Ising model – adiabatic quantum optimization – simulated (quantum) annealing – Fortran

[*]Master Degree in Physics of Data, University of Padua, Italy

Contents

1 Theory

In this Section we summarize the main theoretical aspects of our analysis, particularly focusing on the tackled problems and the analytical aspects of the algorithms.

1.1 The problems

In Computer Science, there exists a class of problems, some of which renowned, commonly referred to as NP problems, which stands for *non-deterministic polynomial-time*. This expression formally describes a class of problems requiring a polynomial time to be solved on a deterministic Turing machine.

In this context, we are simply interested in distinguishing an NP-complete problem, one that has a decision-making approach and hence implies a yes/no question, from an NP-hard one, such as optimization ones. Usually, the same problem can be either, depending on the goal expected from its resolution; throughout our project, we will always prefer the latter formulation, seeking quantitative results.

Further on, we will describe a possible formalism to solve some of the most famous problems which consists of mapping their description within an Ising framework, following what was done in [1]. In particular, we will focus on problems which are also involved in graph theory, just for coherency.

1.1.1 Ising model

As a preliminary step, we consider a basic Ising model on a 2d lattice with a nearest-neighbor interaction and in absence of an external magnetic field. This model will basically only be used as a sort of litmus test to verify the correctness of our algorithms, since the results will be more immediate to compare with theory. Nonetheless, we deem it constructive to summarize its main mathematical aspects, also because its formalism is deeply involved in the implementation of AQO.

A 2d Ising model on a square lattice[1] with nearest-neighbor interaction can be described by the hamiltonian

$$H = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j \tag{1}$$

in its classical form, with $s_i = \pm 1$, J_{ij} the coupling strengths and the symbol $\langle \cdot \rangle$ indicates that the summation is only taken on nearest-neighbors pairs (considered once).

To express Eq. 1 with a quantum formalism, we just need to substitute the spin values with their corresponding quantum operators, that are the Pauli matrices. The hamiltonian itself becomes an operator and this leads to

$$\hat{H} = -\sum_{\langle i,j \rangle} J_{ij} \sigma_i^z \sigma_j^z \tag{2}$$

where $\sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ is the Pauli matrix for the z component. In Eq. 2, the tensor products with identities remain unexpressed for simplicity, but they can be easily traced back also keeping in mind the property that

$$\mathbb{1}_n \otimes \mathbb{1}_n = \mathbb{1}_{n^2} \tag{3}$$

From a physical standpoint, the ground state of this hamiltonian depends on the value of the interaction between the spins: if $J_{ij} > 0 \ \forall i,j$, we have a *ferromagnetic* system, in which spins desire to be aligned; if $J_{ij} < 0 \ \forall i,j$, the system is *anti-ferromagnetic*, with adjacent spins tending to have opposite signs. Of course, in a quantum scenario, there will be a wavefunction ψ describing the ground state system in a probabilistic way, with the aforementioned states having highest probability.

1.1.2 Graph partitioning

Graph partitioning is one of the first problems tackled in the literature with a mapping to the Ising model, especially because of its immediate interpretability.

We consider an undirected graph G = (V,E) with $N = |V|$ vertices and E edges,

[1]A lattice is, in itself, a graph G = (V,E), with $N = |V|$ spins and where each edge represents an interaction.

where the number of nodes must be even. The aim is to divide them into two equally-sized subsets, while also trying to minimize the edges connecting the two subsets. Leaving aside the possible applications of this problem to concrete cases (such as sparse matrix multiplication, but also clique detection in social or biological networks), it is possible to write an Ising-like hamiltonian in the form

$$H \equiv H_A + H_B \tag{4}$$

where

$$H_A = A \left(\sum_{i=1}^{N} s_i \right)^2 \tag{5}$$

penalizes the energy value if the $+1$ spins are not equal in number to the -1 spins (as they distinguish the two groups), while

$$H_B = B \sum_{ij \in E} \frac{1 - s_i s_j}{2} \tag{6}$$

increases the energy every time an edge exists connecting nodes belonging to different subgraphs. We choose $B > 0$ because we wish to minimize this number.

The subdivision of the hamiltonian into two terms, each penalizing a different aspect of the problem, is often the case in this approach. It should be clear that, here and below, the priority is not to violate the constraint imposed by H_A, which in fact provides a harsher penalization on the energy value.

To obtain the quantum counterpart, one needs to formally substitute $s_i \to \sigma_i^z$, while clearly keeping into account the deep mathematical and physical implications of this substitution: the hamiltonian becomes a matrix and the ground state energy needs to be computed via the wavefunction as $\langle\psi| H |\psi\rangle$, assuming ψ normalized.

1.1.3 Vertex cover

Once again, we consider an undirected graph G = (V,E), trying to find the smallest number of vertices that must be *colored* so that each edge is connected at least with one colored node. This problem can be considered as the complementary version of the set packing problem and it consists of a simplification of the better-known graph coloring, albeit still preserving its NP nature.

First off, we need to define an auxiliary vertex-dependent variable x_i whose value is 1 if the $i-$th vertex is colored, 0 if it is not. The transformation from spin values is trivially

$$x_i = \frac{s_i + 1}{2} \tag{7}$$

Also in this case[2], we can write its full hamiltonian as suggested in Eq. 4, with

$$H_A = A \sum_{ij \in E} (1 - x_i)(1 - x_j) \tag{8}$$

to impose that each edge has at least one colored vertex, and

$$H_B = B \sum_{i=1}^{N} x_i \tag{9}$$

to minimize the number of colored vertices (N is also in this case their total number). We need to require $B < A$ for the correct balance of the penalizations: if a mistaken vertex is uncolored, it is clear that at least one edge will not connect a colored vertex any longer.

The transposition to the quantum formalism is to be performed just as described above.

1.1.4 Traveling salesman

The traveling salesman is one of the most famous NP problems also beyond the academic community, mainly due to its exquisite translation into a real, easy-to-picture scenario.

We consider a complete graph G = (V,E) which we suppose to be undirected, although in this case this is not a strict requirement. A salesman needs to visit several cities – represented by the nodes – without returning to

[2]We realize that the notation might be slightly inaccurate, in that we are re-using and will re-use the same symbols in all the problems. However, we find this notation abuse to be negligible compared to the gain obtained in clarity.

any city and while also trying to minimize the costs of his/her journey, embodied by the weights associated to each edge. The salesman is then supposed to get back to its initial position: we consider a cycle over the graph.

To formalize the problem, we again need to employ the binary variable defined in Eq. 7, which will however express two indices $x_{v,i}$, v representing the vertex and i its order in the path: this means that the system will be in the city (spin) v at time step i.

As usual, we write the hamiltonian of the problem as the sum of two terms. Considering n cities, on the first one we have

$$\begin{aligned} H_A =& A\sum_{v=1}^{n}\left(1-\sum_{j=1}^{n}x_{v,i}\right)^2 + \\ & A\sum_{j=1}^{n}\left(1-\sum_{v=1}^{n}x_{v,i}\right)^2 + \\ & A\sum_{uv\notin E}\sum_{j=1}^{n}x_{u,j}x_{v,j+1} \end{aligned} \tag{10}$$

in which the first two terms require that every vertex only appears once in the cycle and that there must be a $j-$th node in the cycle for each j, while the third addendum imposes and energy penalty if $x_{u,j}$ and $x_{v,j+1}$ are both 1 without being connected. The second term, specific to the traveling salesman problem, is

$$H_B = B\sum_{uv\in E}w_{uv}\sum_{j=1}^{n}x_{u,j}x_{v,j+1} \tag{11}$$

to introduce the cost-minimization aspect of the problem. The quantity w_{uv} represents the weight connected to each edge $uv \in E$, which we assume to be all positive; once again, it should never be favorable to violate the constraint imposed by H_A rather than H_B, so we need to impose $0 < B\max(w_{uv}) < A$.

It is fundamental to notice that for n cities, $N = n^2$ total spins are required. Because of the two indices, we will need to account both for the spin labels and their appearance order in the sequence. For this reason, the translation on to a quantum formalism is slightly less trivial than the previous cases.

1.2 The algorithms

We will be solving the problems described above using 3 algorithms, which can be grouped into two families: on the one hand we have adiabatic quantum optimization, which is fully based on quantum concepts and the formalism relying on them; on the other hand we will implement Monte Carlo algorithms in the form of a simulated annealing (simple Metropolis) and quantum simulated annealing (Metropolis with Trotter replicas), which are instead based on the classical hamiltonians and have a stochastic nature.

1.2.1 Adiabatic Quantum Optimization (AQO)

Also referred to as Adiabatic Quantum Computation, this algorithm basically consists of a computer simulation of the inner machinery of a quantum device (such a, for example, a D-Wave).

It is based on the *adiabatic quantum theorem*, which states that if we have an hamiltonian $\hat{H}_P$ whose ground state represents the solution of the chosen NP problem, we can prepare the spin system in the ground state of an *easy* hamiltonian $\hat{H}_0$, evolve it slowly according to the following

$$\hat{H}(t) = \left(1-\frac{t}{T}\right)\hat{H}_0 + \frac{t}{T}\hat{H}_P \tag{12}$$

and the system will remain in the ground state at all time instants. This however holds provided that the two hamiltonians do not commute ($[\hat{H}_0, \hat{H}_P] \neq 0$) and that T is large enough, meaning the time evolution is performed slowly so that it can in fact be adiabatic. This is necessary to avoid *Landau-Zener transitions* into excited states at some intermediate state during the evolution.

Throughout our analysis, we will choose

$$\hat{H}_0 = -h_0\sum_{i}^{N}\sigma_i^x \tag{13}$$

where N will be the total spins in the system considered and $\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. The hamiltonian $\hat{H}_P$ depends from case to case on the problem considered.

The choice of a correct final time T is a delicate matter as well. In [2], an extensive quantitative study is performed on this subject and it is found that one should have

$$T \sim \frac{1}{\min\limits_t \Delta E(t)^2} \tag{14}$$

where $\Delta E(t)$ is the difference between the first excited state and the ground state of $\hat{H}(t)$ considered over the full time evolution. The tuning of this parameter will anyways be matter for analysis later on.

1.2.2 Simulated Annealing (SA)

In a classical formalism, the system is represented by a configuration made of N spins with an explicit value for each site. Within a standard Monte Carlo framework, we implement a Metropolis algorithm that is capable of finding the ground state configuration of the system starting from a randomly and independently initialized state.

This result is achieved by flipping spins iteratively (implementation-related details can be found in the section *Code development*) accepting a new configuration with probability

$$\min\left\{1, e^{-\frac{\Delta E_i^{(k)}}{T_k}}\right\} \tag{15}$$

where we define $\Delta E_j^{(k)} = E_{j'}^{(k)} - E_j^{(k-1)}$ as the energy difference after flipping spin j at iteration k. This means that if the spin configuration has a lower energy than the previous one, it is accepted automatically, otherwise with a probability connected to the Boltzmann law.

Annealing comes into play in the fact that the temperature T_k depends on the iteration: in particular, it is decreased linearly at each time step until it gets to 0. This expedient is expected to improve the performance of the algorithm, as it avoids it being trapped in a local minimum configuration by decreasing the temperature value.

1.2.3 Simulated Quantum Annealing (SQA)

There are several variations of quantum Monte Carlo algorithms in the literature; in this project, we implemented a version described in [3][4][5], employing the Suzuki-Trotter replicas approach. Simulated quantum annealing is a classical simulation to approximate quantum annealing by path-integral formulation, in which the quantum fluctuations are based on the presence of an external magnetic field.

Conceptually, the main difference from the "classical" simulated annealing in that its quantum counterpart allows the energy evolution to undergo a *quantum tunneling effect*, as shown in Fig. 1, which in principle should prove more effective in reaching a global minimum configuration.

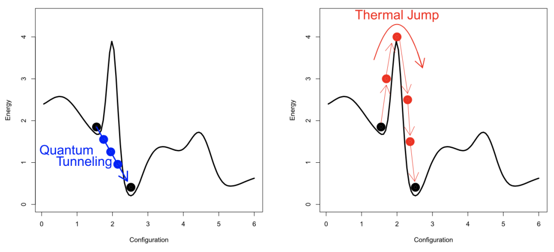


Figure 1: *Schematic exemplification of the differences in the energy minimization as function of iterations between SA and SQA (picture taken from [3]).*

From a theoretical standpoint, the partition function of the quantum hamiltonian is mapped into a classical one through the Trotter formula, obtaining an approximated classical hamiltonian on which to apply the Metropolis algorithm.

The quantum hamiltonian describing the quantum annealing has the general form

$$\hat{H}_Q(t) = A(t)\hat{H}_0 + B(t)\hat{H}_P \tag{16}$$

where $\hat{H}_0$ and $\hat{H}_P$ are the same hamiltonians introduced in Eq. 12 and $A(t)$ and $B(t)$ are time-dependent functions that describes the annealing schedules that in Eq. 12 are set to be *linear*.

The approximated classical hamiltonian obtained through the Trotter formula over Eq. 16 is

$$\begin{aligned} H_C =& B(t) \sum_{m=1}^{M} H_P(s_m) \\ & - J(t) \sum_{m=1}^{M} \sum_{i=1}^{N} s_{i,m} s_{i,m+1} \end{aligned} \tag{17}$$

This approximated classical hamiltonian consists of a set of M copies of N spins called Trotter slices or replicas, where each copy describes a configuration of the considered NP-problem [4]. In particular, the Trotter decomposition maps the original d-dimensional problem into a $(d+1)$-dimension problem where the new dimension corresponds to the *Trotter dimension* containing the replicas that interact among each other with the time-dependent coupling $J(t)$.

With respect to Eq. 17, $s_{i,m}$ refers to the spin i in the m-th replica, s_m refers to the configuration of all N spins describing the NP-problem for replica m while the coupling $J(t)$ is defined to be

$$J(t) = -\frac{MT}{2} \ln \left[\tanh \left(\frac{A(t)}{MT} \right) \right] \tag{18}$$

In particular, the time-dependent functions describing the quantum annealing schedules are chosen to be the same used physically in the D-Wave quantum annealer, while the temperature of the system MT is fixed during the whole process; they are

$$\begin{aligned} A(t) &= \begin{cases} 8t^2 - 9.9t + 2.88, & \text{if } 0 \le t \le 0.6 \\ 0, & \text{if } 0.6 \le t \le 1 \end{cases} \\ B(t) &= 5.2t^2 + 0.2t \end{aligned} \tag{19}$$

It is interesting to observe that, since $A(t)$ is a decreasing function that goes to 0, the coupling $J(t)$ among replicas increases with time (reaching $+\infty$ when $A(t) = 0$), making the interaction among them stronger, obtaining at the end of the process, identical copies of the same configuration: $s_{i,m} = s_{i,m+1}$ $\forall i, m$.

After mapping the quantum $d-$dimensional system into a classical $(d+1)-$dimensional one, it is necessary to apply the Metropolis algorithm with both local and global moves.

The system is initialized randomly and then the moves are performed: the local move consists of the classical independent flip of spin $s_{i,m}$ at site i and replica m, while the global move consists of a simultaneous flip of all the spins at site i in *all* the replicas $m = 1, ..., M$. A local move at all sites i, m and a global move at all sites i is called a *sweep*.

As in the simulated annealing, a move, both local and global, is accepted with a specific probability defined by the variation of the energy induced by the flip. For the local move at iteration k, the variation of the energy induced by the flip of spin $s_{i,m}$ is

$$\begin{aligned} \Delta E_{i,m}^{(k)} =& B(t_k) \left[H_P(s_m^k) - H_P(s_m^{k-1}) \right] \\ & + J(t) [s_{i,m-1}^{k-1} s_{i,m}^{k-1} + s_{i,m}^{k-1} s_{i,m+1}^{k-1} \\ & - s_{i,m-1}^{k} s_{i,m}^{k} - s_{i,m}^{k} s_{i,m+1}^{k}] \end{aligned} \tag{20}$$

where s_m^k denotes the configuration at iteration k within a sweep; this differs from s_m^{k-1} for the flip of spin $s_{i,m}$. The local move is accepted with probability

$$\min \left\{ 1, e^{-\frac{\Delta E_{i,m}^{(k)}}{MT}} \right\} \tag{21}$$

For the global move at iteration k, the variation of the energy induced by the flip of spins at site i in all the replicas is

$$\Delta E_i^{(k)} = B(t) \sum_{m=1}^{M} \left[H_P(s_m^k - H_P(s_m^{k-1})) \right] \tag{22}$$

The global move is accepted with probability

$$\min \left\{ 1, e^{-\frac{\Delta E_i^{(k)}}{MT}} \right\} \tag{23}$$

At each sweep, the value of $A(t)$ (and consequently the value of $J(t)$) and $B(t)$ are updated as described in Eq. 18 and Eq. 19;

after a sufficient number of sweeps, the system is found in the ground state and to determine its energy it is sufficient to compute H_P of an arbitrary replica.

With respect to the simulated annealing, in the simulated quantum annealing the thermal fluctuation is replaced by a quantum one described by the schedules $J(t)A(t)$ and $B(t)$ and the mapping of the original problem into a one with higher dimension makes it necessary to introduce global moves aside from local ones.

2 Code development

In this section, we will focus on describing the code we implemented, which has been developed in Fortran90 and Python. We tried to always keep into perspective those parameters that should constitute the backbone of proper scientific software; in particular, the code was developed in a co-modular way, in order to have different modules perform a specific and well-defined task.

2.1 Modules for initialization

The `initialization` module is dedicated to initializing each problem. Since we always considered NP problems with an underlying graph description, the mathematical tool to do so is an *adjacency matrix*, describing in a_{ij} with 0 and $\neq 0$ respectively the absence or presence of an edge between node i and node j. Because we always consider undirected graphs, the adjacency matrix will always be a symmetric one, except for the Ising model, in which nearest neighbour interaction is instead implemented. Moreover, the matrix gets initialized at random every time the subroutine specific to each problem is called. Saving to file is also handled via a *logical switch* tunable by the user.

To avoid excessive randomness, during the analysis it was decided to set the number of edges as the average between the minimum (to have only one connected component) and maximum value possible; given G = (V,E), this is

$$E = \texttt{floor}\left(\frac{(|V|-1)(|V|+2)}{4}\right) \qquad (24)$$

However, the end user is free to input different values at will.

The `hamiltonian` module is instead devoted to building the hamiltonians; in it, subroutines can be found for each problem tackled, both for a classical and a quantum formalism. In this second case, we make extensive use of some helpers described in Sec. 2.4 to ease the definition of the hamiltonian operators using tensor products.

In order for the overall program to become more flexible, it was decided to implement `interfaces`, as shown in the code snippet below. With this trick, we can distinguish whether we need a quantum or a classical hamiltonian and the kinds or order of arguments will automatically indicate which problem we want initialized.

```fortran
INTERFACE QuantumH
  MODULE PROCEDURE
  IsingModelQuantumHamiltonian,
  GraphPartQuantumHamiltonian,
  VertexCoverQuantumHamiltonian,
  TravSalesmanQuantumHamiltonian
END INTERFACE

INTERFACE ClassicalH
 MODULE PROCEDURE
 IsingModelClassicalHamiltonian,
 GraphPartClassicalHamiltonian,
 VertexCoverClassicalHamiltonian,
 TravSalesmanClassicalHamiltonian
END INTERFACE
```

Listing 1: *Example of `interface` implemented for the `hamiltonians` module; this tool has been used extensively throughout the project to improve the interpretability of the code itself.*

Interfaces are used widely also in other modules, with the same aim to simplify the end-user experience and improving the flexibility of the whole program. Also, it is made an habit to use `allocatable` arrays, again with the same purpose.

2.2 Modules for algorithm implementations

Each algorithm has been developed within a module, together with all the necessary tools to develop it within an analysis framework and to obtain the necessary quantities.

In the module `quantum`, the AQO (or

AQC) algorithm is laid out. The first interesting feature is the creation of a specific `hamiltonian type` to group all the hamiltonians and quantities needed from time to time.

```fortran
TYPE HAMILTONIAN
    integer :: T_final
    real*8 :: E_gs
    real*8, dimension(:,:), allocatable :: h0
    real*8, dimension(:,:), allocatable :: hP
    real*8, dimension(:,:), allocatable :: h_tot
END TYPE HAMILTONIAN
```

Listing 2: *Definition of the new type* `hamiltonian`*, which will be used to define the quantities needed to perform AQO for the spin system considered.*

As it can be seen, among the attributes we have the 3 needed hamiltonians $\hat{H}_0$, $\hat{H}_P$ and $\hat{H}(t)$ (which are all `real*8` because they will always be describing spins systems and due to the necessity of having higher precision during time evolution), the final time T regulating the evolution itself and the energy ground state value computed. Each quantity is defined in a specific subroutine; they are then all grouped together in `AQCAlgorithm`, which lays out all the AQO steps, from parameter initialization and time evolution to energy computation and file savings. The subroutine `PsiToConfig` also extracts the configuration at highest probability and converts it into a spin representation, which will be used later on to allow a visualization of the problem solutions.

We also show the code regarding the wavefunction time evolution, which is a key aspect of AQO.

```fortran
linear = .false.
if (linear) then
    !Time evolution (Crank-Nicolson)
    ssize = size(hamilt%h_tot,1) ! square matrix
    allocate(AA(ssize,ssize), BB(ssize,ssize))

    call Identity(idmat, ssize)
    factor = dcmplx(0.,1.)*dtime*dcmplx(0.5,0.)
    AA = idmat + hamilt%h_tot*factor
    BB = idmat - hamilt%h_tot*factor

    psi_cmplx = matmul(BB, psi_cmplx)
    !Invocation of Lapack subroutine to solve
    !linear system (dir num int)
    call LinSysSolver(AA, psi_cmplx)

else
    !Simpler method (Euler)
    factor = dcmplx(0.,1.)*dtime
    psi_cmplx = psi_cmplx - &
            matmul(hamilt%h_tot*factor,psi_cmplx)

    norm = sqrt(dot_product(psi_cmplx,psi_cmplx))
    psi_cmplx = psi_cmplx/norm

end if
```

Listing 3: *Time evolution of ψ with both implementations; the one employed in the analysis is the Euler method, contained in the* `else` *statement.*

It is interesting to notice that the wavefunction time evolution in `PsiTimeEvolution` is performed with direct numerical integration, via the *Euler method*, which consists of Taylor-expanding the exponential time-evolution operator to avoid the diagonalization of $\hat{H}(t)$ – the boolean `linear` is set to `false` by default. While it is true that this method shrinks and rotates $\psi(t+dt)$ as it does not preserve the norm – which can anyway be easily fixed by normalizing ψ at each time step –, it still proved computationally more efficient than the Crank-Nicolson algorithm, which requires solving a linear system (via the `zsysv` Lapack subroutine) involving very high-dimensional matrices.

As far as the module `simulatedannealing` is concerned, it contains the necessary subroutines to implement a simulated annealing Metropolis, so this time employing a classical formalism.

As shown in Listing 4, the initialization of the spin configuration happens at random through the `random_number` built-in function. To make sure that the values are in fact ± 1, a mathematical trick is employed taking advantage of the `floor` function, as in line 5.

```fortran
integer, dimension(:) :: Spin
Spin = 0
do ii=1,size(Spin)
    call random_number(u)
    Spin(ii) = 2*FLOOR(2D0*u)-1
end do
```

Listing 4: *Code snippet showing how the starting spin configuration is initialized for the Metropolis SA algorithm.*

Again, also in this case, it should be noticed that each subroutine is dedicated to a specific, pinpointed task (spin initialization, temperature scheduling, acceptance or refusal of a configuration and so on) and that afterwards they are all organically combined. The only difference in this context is that each problem is dealt with separately, mainly because of the different parameters needed to be provided as input to each. Each subroutine is built such that several repetitions of the data-taking process can be performed, so as to smooth out the intrinsic stochasticity of Monte Carlo methods.

Moving onward, the `simulatedquantumannealing` module is dedicated to the implementation of the second Monte Carlo algorithm. Many are the similarities with SA, but it is more interesting to focus on some details as its complexity is higher.

In this case, the `spin` quantity, which embodies configurations, is a $2d$-matrix rather than an array, since an extra index is required to run over Trotter replicas. This also implies the presence of extra auxiliary subroutines, like those for the scheduling of the coefficients $A(t)$ and $B(t)$ and the subroutine `TrotterInteraction`, dedicated to computing the energy difference given by the Trotter replicas interaction, an excerpt of which is found below.

```fortran
integer, dimension(:,:) :: Spin
!case of last replica
if (mm == size(Spin,1)) then
    DeltaE = 2*J*(Spin(mm-1,jj)*
    Spin(mm,jj)+0.0)
!case of first replica
else if (mm == 1) then
    DeltaE = 2*J*(0.0+Spin(mm,jj)*
    Spin(mm+1,jj))
else
    DeltaE = 2*J*(Spin(mm-1,jj)*
    Spin(mm,jj)+Spin(mm,jj)*
    Spin(mm+1,jj))
end if
```

Listing 5: *Code implementation of the Trotter interaction between the replicas. The* `if` *statement is needed to deal separately with the case of the first and last replica; indeces* `mm` *and* `jj` *are arguments of the subroutine.*

Each problem finds its specific layout of the algorithm, with the same implementations as above. The key difference with SA is that here both global and local moves need to be performed, but this aspect has been discussed in the section *Theory*. To save precious computational time, the energy variations were computed analytically and directly implemented in the code (Listing 5 is but one of many such examples), instead of computing them as the difference between subsequent configurations, which would have required several nested `do` loops. Explicit analytical formulae can be found in *Appendix A* for each problem.

Finally, the module `energy` contains the necessary subroutines to compute the estimations of the energy values obtained from the algorithms. For AQO, the hamiltonian $\hat{H}_P$ is directly diagonalized using the Lapack `dsyev` subroutine, suitable for double precision symmetric matrices, as it is the case; this will be used as a theoretical estimation of the ground state, to be compared with the computational value.

As far as SA and SQA are concerned, a subroutine computes the GS estimate as the average of the values obtained in the final 10% iterations, where it was seen both algorithms reach stability. A different subroutine computes the residual energy per particle (defined as the *difference between computational and theoretical value*, clearly divided by the number of spins). File saving is automatically managed.

2.3 Modules for debugging

Debugging has had an active part throughout the whole project implementation and several tools are in place also to ease the usage from the end user at analysis time.

The module `checkpoint` is an helper to allow printing of debugging messages and variables particularly useful at implementation time. Several subroutines are needed for this very purpose because we need to distinguish between types; however they are all grouped under the aegis of the same interface `debug`. Since among these functions we also have one for string printing, that specific one will also be employed to print messages directed to the end user with instructions at running time.

The module `debugger` is instead dedicated to more proper debugging, such as checking *pre-conditions* and *post-conditions*, especially for the adiabatic quantum optimization. As an example, the subroutine `MatCommutationCheck` checks on entry that $\hat{H}_0$ and $\hat{H}_P$ in AQO do not commute, which is required by theory; the subroutine `MatEqualCheck`, on the other hand, checks on exit that $\hat{H}(t = T) = \hat{H}_P$, as one would expect, in order to avoid possible numerical instabilities.

Also, every time a Lapack subroutine is applied, we take advantage of the `info` built-in parameter to flag if anything went wrong during the execution of the subroutine itself.

Finally, the subroutines `HamiltSpectrumCheck` and `EigenvecsGSCheck` are present, built to study the spectrum and the eigenvectors respectively for a symmetric matrix in double precision, using the Lapack `dsyev` subroutine. They were created to address a specific problem encountered during implementation regarding the probability trends in the AQO evolution. They made it possible to gain a precious insight on the physical aspects of this algorithm, as it will be shown in the *Results* section.

2.4 Modules for general support

Two general-purpose modules offer the necessary utilities for basic operations that are not specific to the implementation of this project.

In `mathutils` we find all the necessary mathematical tools, such as a subroutine where the tensor product is defined, one to initialize identity matrices, another one to initialize Ising-model hamiltonians, one to perform the solving of a linear system for symmetric double-complex matrices via the `zsysv` Lapack subroutine.

In the `instrutils` we instead collect all the tools to save arrays or matrices to file or to print them on to the terminal. Several subroutines are found depending on the type of the objects (`integer`, `real` and `real*8` are considered for saving, `real*8` and `complex*16` for printing), but once again interfaces come into play to group all of them under unifying names, by category.

The subroutine `PrepareLocation` is needed to prepare the path on which to save results or quantities, receiving as input a properly-manipulated string; this gives great automation in storing our results, as it also creates separate folders with the parameters initialized for a specific problem, so as to have an ordered and systematic storing of the analysis data and plots.

2.5 Modules for general incorporation

Four modules belong to this category, one for each problem studied: `im` (Ising model), `gp` (graph partitioning), `vc` (vertex cover) and `ts` (traveling salesman).

Apart from the obvious differences peculiar to the specific problems, these modules are similarly designed and have a cardinal importance in the project development. They address three goals:

1. They provide the tools for the end user to input some parameters of choice to the terminal in order to initialize a given problem (namely, number of nodes or edges and hamiltonian coefficients). Also, they make it possible to choose what steps are wanted in the analysis of each problem thanks to *yes/no* questions, in order to be able to select only what one actually needs to see; this greatly improves the flexibility at running time.

2. They perform the *parameter tuning* of those quantities that we do not leave to the end user to choose (subroutine `TuningParameters`); in particular, these

are temperature ranges in SA and temperatures and number of replicas in SQA. This means that a grid-search is performed to select the best parameters (meaning those minimizing the ground state energy), privileging a machine learning kind of approach.

3. Most importantly, they incorporate within a single framework the full running of the three algorithms, together with the complementary analysis on each both time-wise and accuracy-wise (subroutine `Performance`). This makes it possible to put all the operations in order, with an outstanding gain in neatness of the main program.

2.6 The main program

In `main.f90`, the main program for the project is found.

The unpacking of the project into several modules bears its fruit in this very context, as we are able to write an extremely simple main, which we are in fact able to copy in its entirety below.

```fortran
PROGRAM PROJECT

USE IM
USE GP
USE VC
USE TS

IMPLICIT NONE

call IsingModel()
call GraphPartitioning()
call VertexCover()
call TravelingSalesman()

STOP
END PROGRAM
```

Listing 6: *Main program of the whole project.*

It should be noted that no parameter at all is required in input: we just have calls to the subroutines specific to the NP problems, each with the list of operations needed to carry out an analysis. However, the possibility to tune the main parameters is left to the end user at running time, thanks to the incorporation modules described above.

2.7 Python coordination script

A Python script is the *launch pad* for the end user to run the analysis. First off, it makes it possible to start it just by calling

```
python analysis.py
```

from terminal, avoiding the user to manually write the compilation command, which requires listing a long number of module files and instructions.

The script then guides the user through the possible macro-steps, asking:

- if the re-execution of the Fortran program is wanted (the program will then itself prompt the user with more specific questions and the initialization of problem-related parameters);
- if it is wanted to show analysis results, which will translate into saving plots and samples from each selected problem.

The script also includes some printing statements to keep the execution status under control.

Plots and graphs are realized using Python. The script in fact also contains all the necessary functions to automate this process (again, specifically for each NP problem) and we also feel we can obtain better visualizations of our results – especially as far as graphs rendering is concerned (thanks to the package `NetworkX`).

3 Results

Throughout our analysis, we decided to focus on the performances of the algorithms, both with respect to accuracy and computational time, in the solution of the chosen NP problems. We aimed at not losing track of the physical background involved in many of these processes, but for simplicity it was decided to impose all side parameters (for example, the couplings J_{ij}, the magnetic field h_0) to be unitary, unless otherwise specified due to mathematical constraints.

Before delving into the core of the analysis, let us briefly define the key quantities that will be employed throughout. The *residual energy per particle* is

$$\varepsilon = \frac{1}{N}\left(E_{gs} - E^{\star}_{gs}\right) \tag{25}$$

where E_{gs} is the energy estimate obtained from the algorithm and $E^{\star}_{gs}$ the "true" value.

In the case of AQO, this true value $E^{\star}_{gs}$ is defined as the lowest eigenvalue obtained from direct diagonalization of $\hat{H}_P$, which is possible to perform because N is low enough. For SA and SQA, given that we consider much higher numbers of particles, it is impossible to compute a ground state by direct diagonalization. Following what was done in [6], we define the true value as the minimum value obtained in 100 iterations of the simulated quantum annealing (the best-performing one between the two), taken before the analysis and without considering averages.

With respect to the final time of the adiabatic quantum optimization, it is observed in all the problems that this time is $T \sim 1$. Thus, considering that the higher the final time is, the lower the residual energy becomes, we decided to increase the final time from 1 to 100 for all the computations in order to observe this expected behaviour.

A similar reasoning over the number of sweeps was done for both the Monte Carlo simulations: we show the behaviour of the residual energy as the number of total sweeps increases. Furthermore, as mentioned in *Code development*, for each problem (both in the analysis of the performances and in the specific sample studied) the best parameters for SA and SQA are searched for and then used in the required analysis; thus all the results are achieved with the best parameters obtained through the tuning.

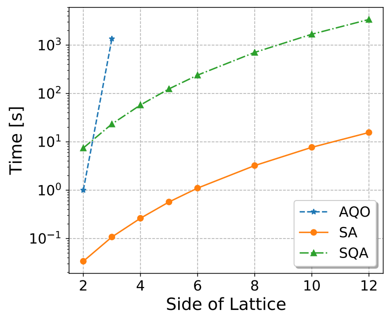
Finally, when showing results for Monte Carlo methods, those are always to be considered as *averages* over 100 repetitions of each, due to the intrinsic stochastic nature of these algorithms.

3.1 Ising model

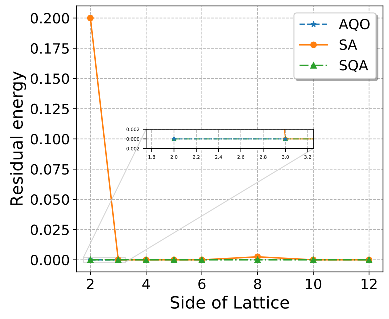
Performance comparison

We remind that this problem should be considered as a sort of *pre-analysis* to perform a global assessment of our algorithms, so not all its aspects will be discussed fully.

We start out by analyzing the performances of our algorithms as far as both computation time and accuracy (measured in terms of ε, residual energy per particle) are concerned.



(A) *Time performance of the 3 algorithms as function of the lattice side for the Ising model.*



(B) *Accuracy performance of the 3 algorithms as function of the lattice side for the Ising model.*

Figure 2

To begin with, we notice that in Fig. 2 we put on the $x-$axis the side of the lattice, meaning the actual number of particles is that value squared; this is the reason why we only have two points for AQO. In general, it is clear how this algorithm performs significantly slower compared to MC methods, mainly due to the operations it needs to compute using matrices with exponentially higher dimensions. On the other hand, residual energies are basically zero because the estimations come from direct calculations. Between SA and SQA, the former appears to be more time-convenient, although SQA is more stable accuracy-wise. This will be especially true in more complex problems.

2×2 grid

From Fig. 2B, it can be seen how the classical simulated annealing shows a considerable spike for the case of a lattice with side 2, which clearly is an outlier compared to the other algorithms but also to SA itself.

It was decided to perform a more focused investigation on the reasons why this happens, so we had SA and SQA run in this scenario for $10,000$ iterations.

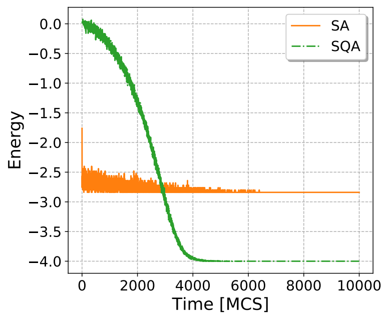


Figure 3: *Energy trend for SA and SQA on the* 2×2 *grid; it is clear how the classical algorithm remains trapped in a configuration leading to geometric frustration.*

As we can see, SA oscillates in a narrow interval of energy values until it remains stuck toward the end of the run. This happens because in this particular scenario, *geometric frustration* takes place: the spin flips are performed in an ordered fashion and the algorithm is not capable of escaping the situation of flipping back and forth between equivalent configurations. A possible immediate improvement would be to randomize that ordering, but it was left it this way to keep the inner structure of the model – also, this is basically the only case where we have a blatant failure of one of our algorithms. SQA appears to be overcoming this issue thanks to the tunneling effect it is capable of undergoing, as it fairly evident in the plot.

3×3 grid

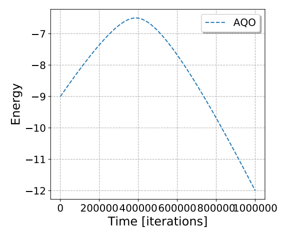
We now consider a 3×3 square lattice and have our algorithms run for more detailed analyses. AQO results have deep physical links, which in a way works as a proof of good implementation of the algorithm.

Firstly, we can see that the energy trend obtained as $\langle\psi(t)|\, \hat{H}(t)\, |\psi(t)\rangle$ (shown in Fig. 4A), so from the computational time evolution of the wavefunction, is the same as the trend of the first eigenvalue of $\hat{H}(t)$ retrieved from direct diagonalization using the proper Lapack subroutine, shown in Fig. 4B. This plot also shows the different degeneracy of the ground state when evolving the system from $\hat{H}_0$ to $\hat{H}_P$.

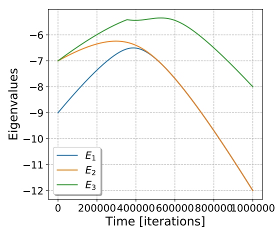
It is also very interesting to study the behaviour of the probability to find our system at ground state, computed as $|\langle\psi(t)|v_{gs}(t)\rangle|^2$ with $v_{gs}(t)$ representing the ground state eigenvectors obtained from diagonalization of $\hat{H}(t)$ – if they are degenerate, that value corresponds to the the sum of all degenerate states. It is interesting to see how the probability is very close to 1 throughout the evolution, meaning no transition to excited levels takes place. A more considerable dip (0.3%) is spotted around iteration $400,000$ but from Fig. 4D a very straightforward explanation can be given: around the very same time step, the energy difference between first excited state and ground state reaches its lowest. Where the two states are closest is exactly where we would expect to see a decrease in the probability, as it is where the wavefunction has the highest chance of escaping ground state.

We now study the residual energy trend, in order to have a quantity by which we can compare the accuracy performance of all the 3 algorithms. We show it as a function of the total number of iterations (which for AQO is the product between the final time T and the time step dt chosen for evolving $\psi(t)$).

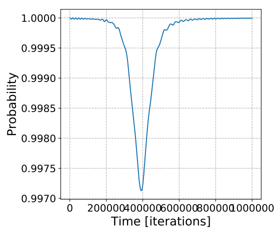
AQO is highly dependent on the total number of iterations (Fig. 4E), given that the residual energy spans 6 orders of magnitudes when going from $10,000$ to 1 million iterations; this comes from the inverse quadratic dependence of T from ε. SA and SQA (Fig. 4F) appear to be more stable in this regard, with the former showing much higher values at low iterations: as already noted, SQA tends to outperform classical annealing but it requires a higher computational time. It should be pointed out that already with 1000 iterations the residual en-



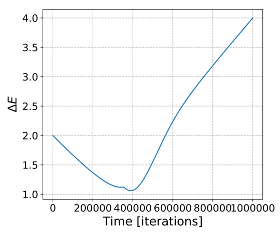
(a) *Energy trend from AQO during time evolution computed as $\langle\psi|\,\hat{H}(t)\,|\psi\rangle$.*



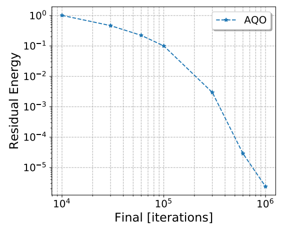
(b) *Ground states and excited states during time evolution of AQO from $\hat{H}_0$ to $\hat{H}_P$ obtained by direct diagonalization of $\hat{H}(t)$.*



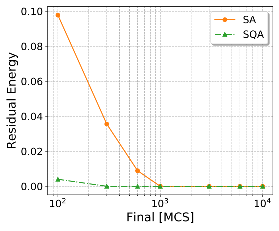
(c) *Probability that the wavefunction finds itself in ground state during the time evolution of AQO – points have been plotted every 1,000 iterations for better file management.*



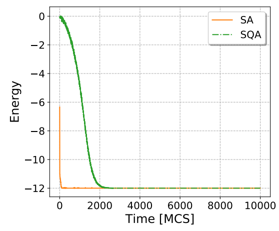
(d) *Difference between ground state and first excited state during time evolution of AQO.*



(e) *Residual energy as function of total time steps for AQO.*



(f) *Residual energy as function of total time steps for SA and SQA.*



(g) *Energy trends for SA and SQA over the highest number of time steps considered. In this simple scenario the "tunneling" effect proper to SQA cannot be spotted.*

Figure 4: *Results of the Ising model on a 3×3 grid.*

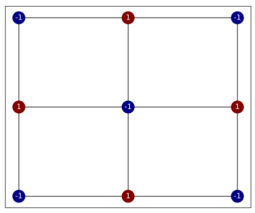
ergies go to 0 for both (even before for SQA, thanks to the different replicas), which implies a significant outperformance compared to AQO, as that requires more iterations and reaches a lower accuracy.

Configuration display

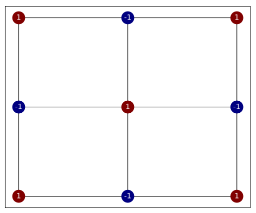
As a final result for this problem, we show actual final configurations of spins given from the algorithms.

We select an antiferromagnetic case, so we expect contiguous spins to be misaligned. From Fig. 5, we can see that this is exactly what happens for all algorithms (we chose SQA to be representative for both MC methods in the 3×3 grid), as they show results in accordance with expectations.

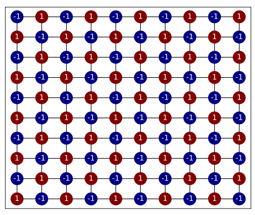
The bottleneck for AQO is once again given by the maximum system size that can be studied due to insurmountable memory constraints. Here we also show results obtained on a 10×10 lattice, but obviously only using SA and SQA: the configurations are correctly initialized.



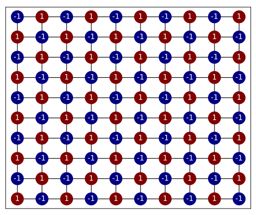
(a) *Spin configuration obtained with AQO for an antiferromagnetic system on a 3×3 grid.*



(b) *Spin configuration obtained with SQA for an antiferromagnetic system on a 3×3 grid – the one with SA was equal.*



(c) *Spin configuration obtained with SA for an antiferromagnetic system on a 10×10 grid.*



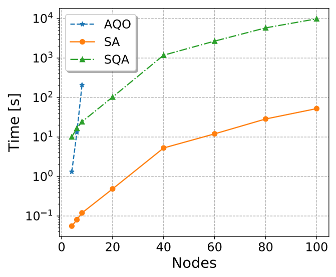
(d) *Spin configuration obtained with SQA for an antiferromagnetic system on a 10×10 grid.*

Figure 5

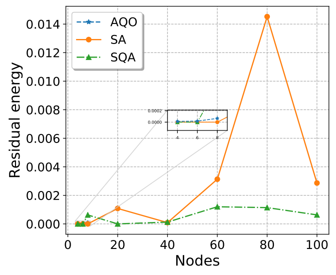
3.2 Graph partitioning

Performance comparison

As before, we start the analysis of this NP problem by considering a direct comparison of the performance between all algorithms as function of the problem size.



(A) *Time performance of the 3 algorithms as function of the number of nodes.*



(B) *Accuracy performance of the 3 algorithms as function of the number of nodes.*

Figure 6

Also for this problem, we notice how Monte Carlo methods perform better than AQO, except for very low values of nodes (where AQO accuracy is comparable but SA still shows better time performance by at least one order of magnitude). The increase in time for AQO is much steeper and its range of applicability remains very limited. It should also be pointed out how SA residual energies are higher by a full order of magnitude when compared to SQA, which once again proves to be the algorithm of choice when it comes down to a high number of particles (of course at a computational cost).

Few-spin configuration: $G = (8, 17)$

The first configuration that we come to consider is one having $N = 8$ nodes and $E = 17$ edges (from formula 24). We can again appreciate from Fig. 7A-B the physical connection between the energy trend obtained computationally and the one retrieved from direct diagonalization of the time-dependent hamiltonian (which also shows the multiple degeneracy of the ground state of $\hat{H}_P$). Also, in Fig.7C-D, we can see the mentioned connection between the dip in probability and the energy difference ΔE between GS and excited states; in this problem the probability shows a strong oscillatory behaviour, which is hard to interpret and which could very well be due to numerical instability; its effect is however of order 10^{-4}, fairly negligible.

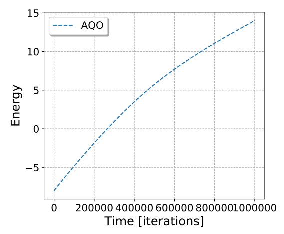
Monte Carlo methods have an energy trend across iterations which is dependent on their numerical aspects rather than on the physics behind the problem. In particular, SA appears to be faster in reaching convergence, but, as we will see later on, this is limited to simple problems. Moreover, residual energies are lower by a 3-fold in SQA for less total iterations and for this problem the value of 0 is reached after 3,000 steps. As far as AQO is concerned, the residual energies tend to be higher especially considering that many more iterations are needed (the plot in Fig. 7G starts on the $x-$axis where the one in Fig. 7F ends).

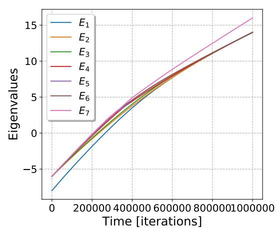
Many-spin configuration: $G = (50, 637)$

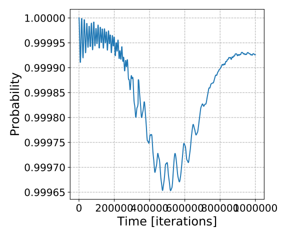
If we now move onto considering a more challenging scenario, with $G = (50, 637)$, we again only need to focus on MC methods.

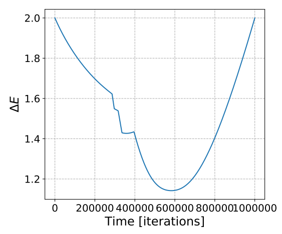
In this case, it is much clearer how the tunneling effect shows up in SQA (around iteration 2,500 in Fig. 8A). This algorithm starts at a higher energy value compared to SA, but thanks to the replicas interaction it manages to drastically reduce it, outperforming its classical counterpart.

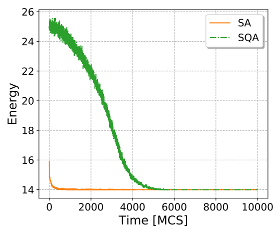
As far as residual energies are concerned, we see that for lower values SA performs better: in fact, it takes more computational time to SQA to reach optimal behavior because of the multiple replicas it has to deal with. Compared to the few-spin configuration, the values are higher by about $30 - 50\%$, but we feel confident in saying that this is an effect of the increased complexity of the problem, which causes the ground state to be harder to reach.

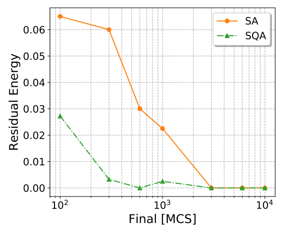

(A) *Energy trend from AQO during time evolution computed as $\langle\psi|\,\hat{H}(t)\,|\psi\rangle$.*


(B) *Ground states and excited states during time evolution of AQO from $\hat{H}_0$ to $\hat{H}_P$ obtained by direct diagonalization of $\hat{H}(t)$.*


(C) *Probability that the wavefunction finds itself in ground state during the time evolution of AQO – points have been plotted every 1,000 iterations for better file management.*


(D) *Difference between ground state and first excited state during time evolution of AQO.*


(E) *Energy trend of MC methods over the highest number of iterations considered.*


(F) *Residual energy as function of total time steps for SA and SQA.*

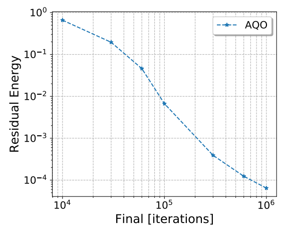
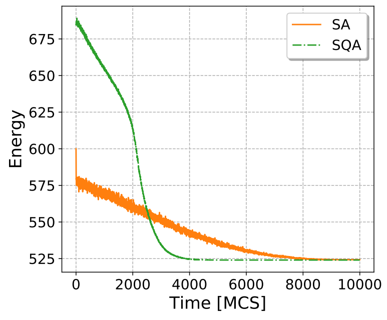

(G) *Residual energy as function of total time steps for AQO.*

Figure 7: *Results of the graph partitioning problem with 8 nodes.*


(A) *Energy trends for SA and SQA over the highest number of time steps considered.*

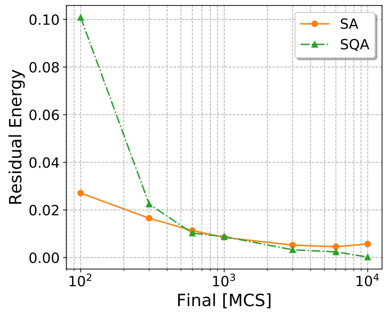
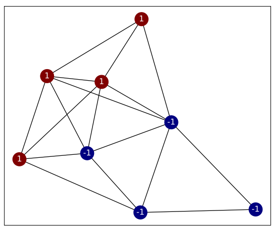
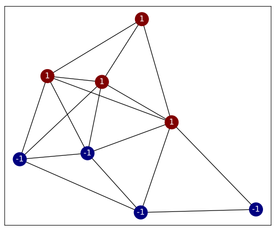

(B) *Residual energy as function of total time steps for SA and SQA.*

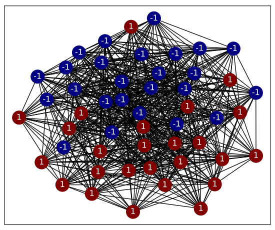
Figure 8: *Results of the graph partitioning problem with 50 nodes.*

Configuration display

As before, we now show the configurations obtained with the optimization processes from all algorithms in the two cases studied.


(A) *Spin configuration obtained with AQO for $G = (8, 17)$.*


(B) *Spin configuration obtained with SQA for $G = (8, 17)$.*


(C) *Spin configuration obtained with SA for $G = (50, 637)$.*

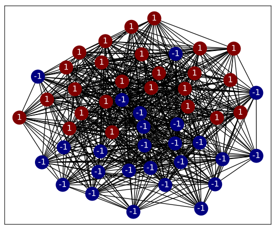

(D) *Spin configuration obtained with SQA for $G = (50, 637)$.*

Figure 9

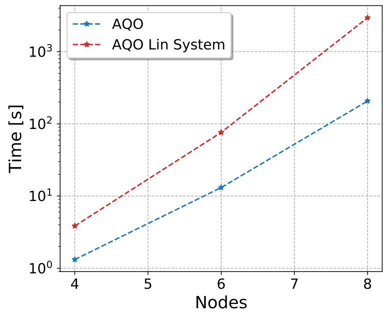
Even for equal initializations, no configuration is exactly equivalent to the other, due to the degeneracy of the ground state. This means that there are several ways to label the spins in order to have the edges connecting the two subsets minimized. Also, it is possible to consider each configuration with its spins flipped, because this problem is invariant under parity.

Considerations on AQO time evolution

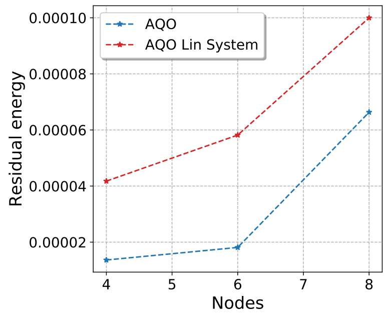
A major debugging problem was encountered at this stage of the analysis, involving the time evolution of the wavefunction; it was already discussed in the *Code development* section but we now show some quantitative analysis performed to better clarify its implications.

We had noticed great numerical instability introduced when trying to evolve the wavefunction applying the Crank-Nicolson algorithm. This reflected itself especially in the accuracy of the algorithm and it was spotted by analyzing the spectrum of $\hat{H}(t)$ and the aforementioned probability $|\langle\psi|v_{gs}\rangle|^2$ to find the evolving state in the GS: this quantity showed a sudden and dramatic drop toward the end of its evolution (at times by 40%).

It was hence tried to have ψ evolve by direct numerical computation, with the only foresight to normalize it at each iteration. As it can be seen from Fig. 10A, this proved to be a rather conspicuous gain in the computing time for the algorithm, which is expected as there is no need to solve a highly-dimensional linear system at every time step. Also accuracies appear to be improved, even if there appears to be some sort of convergence at higher values (higher numbers of steps were not tried due to the computational burden). It should be noted that there exists a trade-off: the method we employed is intrinsically less stable, and it shows so for higher values of iterations compared to those tried out here. For this values, however, the computation time becomes difficult to manage anyway – so it was decided to proceed with the simpler and more immediate algorithm for the rest of the problems tackled.



(A) *Time performance of AQO with and without the implementation of the Crank-Nicolson algorithm for the wavefunction time evolution of the GP problem.*

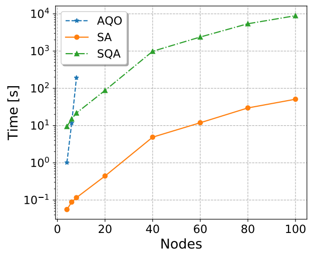


(B) *Accuracy performance of AQO with and without the implementation of the Crank-Nicolson algorithm for the wavefunction time evolution of the GP problem.*

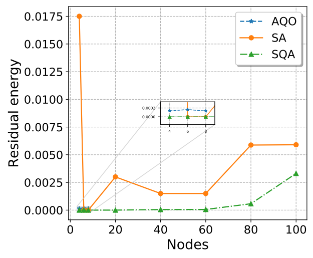
Figure 10

3.3 Vertex cover

Performance comparison



(A) *Time performance of the 3 algorithms as function of the number of nodes.*



(B) *Accuracy performance of the 3 algorithms as function of the number of nodes.*

Figure 12

As usual, we start out by general considerations on the performances of the 3 algorithms on computational time and accuracy. We select $A = 2$ and $B = 1$ for all cases.

From the plots in Fig. 12A-B, we can see that the time values are comparable to those seen for the graph partitioning problem and tend to be higher than the Ising model, as expected; very similar considerations about the trends apply also here.

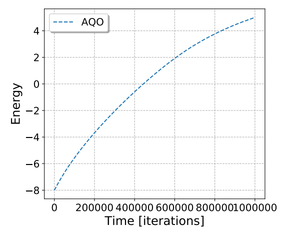
Accuracies do not differ significantly either: the gap in MC methods is likely due to stochastic oscillations. However, in this NP problem we notice how the accuracies tend to increase for SQA at the highest values of nodes considered, even if they remain lower than SA. This could potentially indicate that more nodes imply a noticeable increment in the complexity of the problem optimization. Simulated annealing shows a peak by a factor 4 at the very beginning which could be a signal of geometric frustration, similarly as what happen with the Ising model for the same number of nodes (here however there might be a dependence also on the randomness of the edge initialization).

Few-spin configuration: $G = (8, 17)$

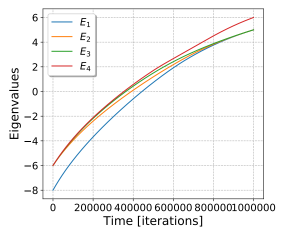
Figs. 11A-B show the accordance between theory and computation in that the trend and scale of the energies from diagonalization are comparable to the one obtained from AQO; in particular, in B we can appreciate the triple degeneracy that the chosen configuration displays.

As far as the probability is concerned, Fig. 11C shows an oscillating behavior which is more accentuated than the previous cases, spanning a range of about 0.2-0.3% in the initial iterations. We also notice a dip around iteration $700,000$ but as usual Fig. 11D provides a solid physical explanation, manifesting that around the same point the difference between first excited state and GS reaches its minimum. We can see that this time the probability never goes back up to 1 fully, but also this fact is linked to ΔE being half its initial value (so the gap between first excited and ground state remains narrow).

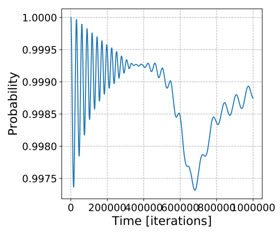
Comparing the residual energies between Monte Carlo methods and AQO (Figs. 11F-G), we see how also here the first two are capable of reaching 0 in the order of 10^3 it-



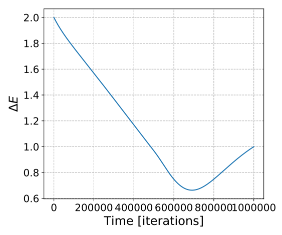
(A) *Energy trend from AQO during time evolution computed as $\langle\psi|\,\hat{H}(t)\,|\psi\rangle$.*



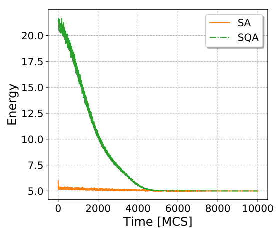
(B) *Ground states and excited states during time evolution of AQO from $\hat{H}_0$ to $\hat{H}_P$ obtained by direct diagonalization of $\hat{H}(t)$.*



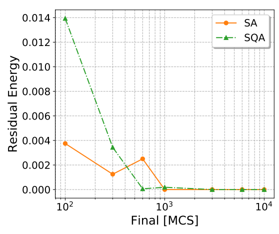
(C) *Probability that the wavefunction finds itself in ground state during the time evolution of AQO – points have been plotted every 1,000 iterations for better file management.*



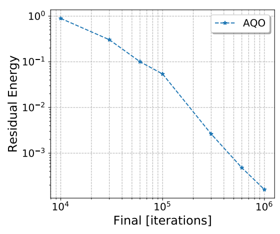
(D) *Difference between ground state and first excited state during time evolution of AQO.*



(E) *Energy trend of MC methods over the highest number of iterations considered.*



(F) *Residual energy as function of total time steps for SA and SQA.*



(G) *Residual energy as function of total time steps for AQO.*

Figure 11: *Results of the vertex cover problem with 8 nodes.*

erations, while AQO only manages to reduce this value to about 10^{-4} and with a number of iterations higher by 3 orders of magnitude, which is a very considerable amount. Actually, it also appears that AQO performs worse by a full order of magnitude with respect to graph partitioning, which could signal how this algorithm is also prone to the increasing complexity of the NP problem itself.

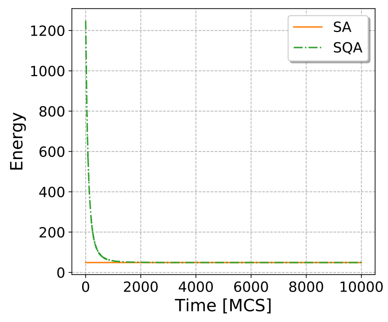
As a general note, the considerations expressed here are similar to those drawn for the graph partitioning, so we start to see a trend inherent to the algorithms themselves more than on the specifics of the problems (which, it needs to be reminded, do fall under the same NP category).

Many-spin configuration: $G = (50, 1225)$

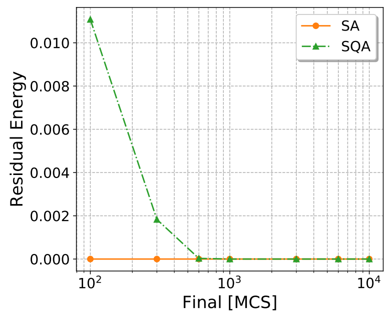
For the many-spin configuration we decided to initialize a 50-node graph but for this problem the number of edges is not computed following Eq. 24; instead, we build the graph so that it is fully-connected (a quick calculation gives $E = 1225$). This configuration should give the intuitive scenario that all nodes but one should be colored (of course this state will be $N-$degenerate), so other than the results, we will be able to verify if this is actually what happens when configurations are displayed.

In this part, we just show the performance results of the Monte Carlo methods. Compared to Fig. 11E and the corresponding many-spin configuration for graph partitioning (Fig. 8A), the tunneling effect proper to SQA here does not appear. This is likely due to the relative simplicity of this specific problem: while it is true that the number of nodes is high, it is also easy to see that a global minimum configuration can be achieved quite easily, also because of its high degeneracy value.

As a further confirmation of this fact, we see that the residual energies promote the SA algorithm in this case and it has been shown how this algorithm struggles with problems that are hard to optimize. In this scenario, SA reaches 0 even after 100 iterations, while SQA takes 600 (which is still few compared to previous cases) mainly due to the greater complexity of the algorithm itself (both in the replicas and the need to perform local and global moves).



(A) *Energy trends for SA and SQA over the highest number of time steps considered.*



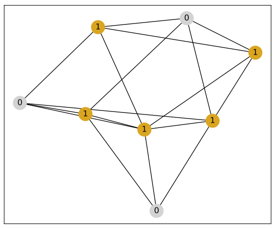
(B) *Residual energy as function of total time steps for SA and SQA.*

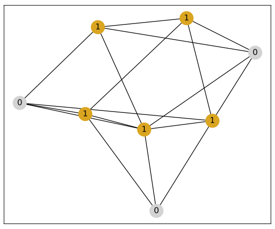
Figure 13: *Results of the vertex cover problem with 50 nodes.*

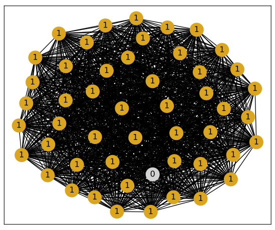
Configuration display

We show the spin configurations obtained with the optimization processes from all algorithms in the two cases studied.

Degeneracy can be appreciated also in Fig. 14A-B, albeit coming from two different algorithms. In particular, the fully connected configuration in Fig. 14C-D shows the expected result both for SA and SQA: all the nodes are colored but a single one in gray. Of course, this node is not the same between the two displays due to the already discussed high degeneracy of this non-generic ground state.


(A) *Spin configuration obtained with AQO for* $G = (8, 17)$.


(B) *Spin configuration obtained with SQA for* $G = (8, 17)$.


(C) *Spin configuration obtained with SA for* $G = (50, 1225)$ *– fully connected graph.*

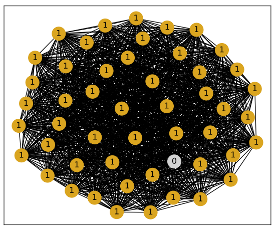
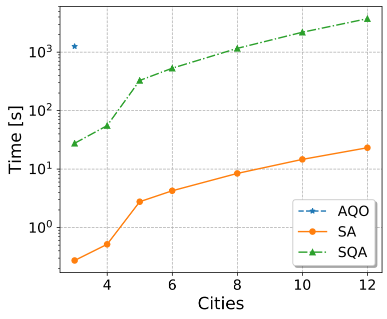

(D) *Spin configuration obtained with SQA for* $G = (50, 637)$ *– fully connected graph.*

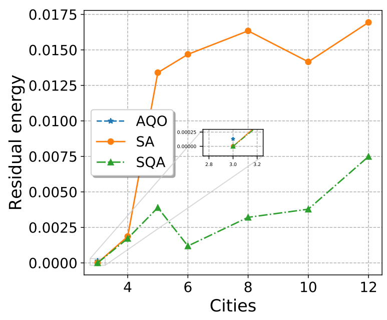
Figure 14

3.4 Traveling Salesman

Performance comparison



(A) *Time performance of the 3 algorithms as function of the number of nodes.*



(B) *Accuracy performance of the 3 algorithms as function of the number of nodes.*

Figure 15

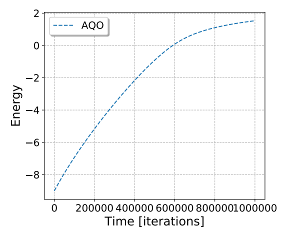
At first, the performances of the three algorithms are analyzed by comparing their cpu-time and residual energy for different number of cities. As explained above, in the traveling salesman problem the number of spins needed to solve the problem is the square of the number of cities meaning that for the adiabatic quantum optimization we can only study the scenario with 3 cities. Furthermore, in order to satisfy the condition over the coefficients of the hamiltonian, the analysis is performed with $A = 2$ and $B = 1$, unless otherwise declared.

As shown in Fig. 15, the problem with 2 cities is not analyzed because it is meaningless. Furthermore, we again notice how the SQA algorithm shows better results in terms of residual energy, even if it slower than SA. In particular, unlike the other problems, with the increase of the complexity (represented by the number of cities) there is a systematic increase not only in the time needed to perform the computation but also in the residual energies. In fact, the traveling salesman problem is, in general, a very complex problem to optimize: the hamiltonian to minimize has many components and it is hard to reach the ground state.

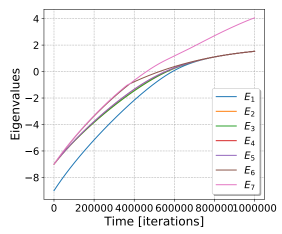
Few-spin configuration: 3 cities

The first configuration we show refers to the problem with 3 cities, the only one that is possible to solve also with the adiabatic quantum optimization. Unluckily, this problem is not very impressive due to the fact that the distance of the cycle (component H_B) does not need to be minimized since just one path is possible; this means the solution of the problem coincides with the minimization of just the hamiltonian H_A in Eq. 10.

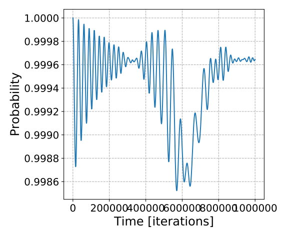
In Fig. 16B it is possible to notice that the ground state has degeneracy equal to 6, corresponding to the possible orders in which the three cities appear in the cycle. It is also clear that the behaviour of the instant energy shown in Fig. 16A coincides with the ground state E_1 of $H(t)$ in 16B, showing that the system clearly stays in the ground state avoiding jumps to the excited ones. The same behaviour can be observed in 16C where the probability oscillates a lot (more noticeably



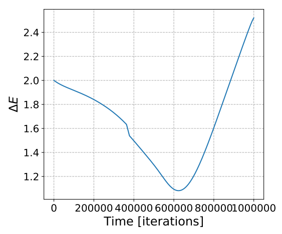
(A) *Energy trend from AQO during time evolution computed as $\langle\psi|\,\hat{H}(t)\,|\psi\rangle$.*



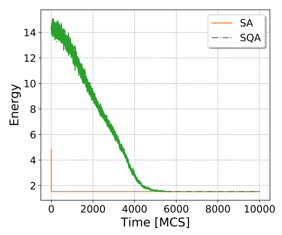
(B) *Ground states and excited states during time evolution of AQO from $\hat{H}_0$ to $\hat{H}_P$ obtained by direct diagonalization of $\hat{H}(t)$.*



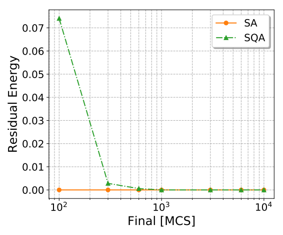
(C) *Probability that the wavefunction finds itself in ground state during the time evolution of AQO – points have been plotted every 1,000 iterations for better file management.*



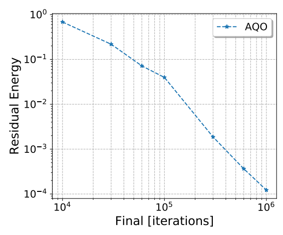
(D) *Difference between ground state and first excited state during time evolution of AQO.*



(E) *Energy trend of MC methods over the highest number of iterations considered.*



(F) *Residual energy as function of total time steps for SA and SQA.*



(G) *Residual energy as function of total time steps for AQO.*

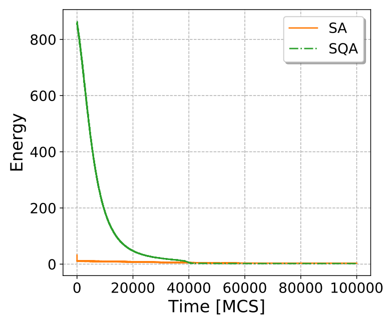
Figure 16: *Results of the traveling salesman problem with 3 cities.*

than the other problems, due to the increased complexity), especially around the minimum gap between the ground state and the first excited one shown in 16D, but stays well above 0.99.

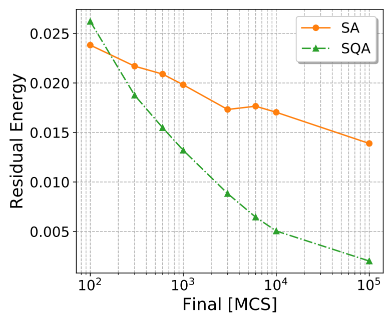
Again, by increasing the final time the residual energy obtained through AQO decreases, as shown in Fig. 16G. As expected, since the considered problem with 3 cities is relatively *easy* to solve, the ground state is immediately reached by simulated annealing, while the simulated quantum annealing, as observed also for the problems already analyzed, requires more sweeps, as displayed in 16E-F.

Many-spin configuration: 10 cities

As observed, the traveling salesman problem with 3 cities is not very informative, therefore we show the results obtained with 10 cities. Obviously the results refer only the the Monte Carlo algorithms because the number of spins required ($10x10$) is too high for the AQO to handle.



(A) *Energy trend of MC methods over the highest number of iterations considered for 10 cities.*



(B) *Residual energy as function of total time steps for SA and SQA for 10 cities.*

Figure 17: *Results of the traveling salesman problem with 10 cities.*

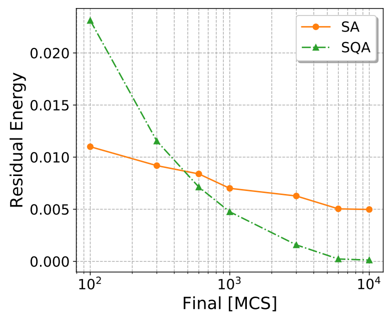
It is interesting to observe in Fig. 17 that as the complexity of the problem is increased, the simulated annealing cannot solve the problem while the simulated quantum annealing is faster in reaching the minimum. In particular, since the complexity is huge, we decided to increase the highest number of iterations to 100,000, that are still not enough to reach the minimum as observed in Fig. 17B.

Dependence on coefficients A and B

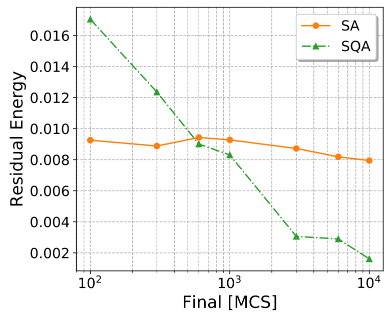
During the analysis, we observed a strong dependence of the results on the coefficients considered for the AQO hamiltonian A and B. In order to investigate this behaviour we performed the simulations with 5 cities and B kept fixed to 1 and A set to 1 and 2: in both cases we checked the condition $B \max(w_{uv}) < A$ that is always assured due the fact that $\max(w_{uv}) < 1$.

In Figs. 18A-B it is possible to study the residual energy obtained for different number of sweeps for the two Monte Carlo algorithms in case of $A = 1$ and $A = 2$ respectively. The residual energy is smaller in case of $A = 1$ while the decrease toward 0 is more accentuated in case of $A = 2$, particularly for the simulated annealing. This behaviour is due to the fact that the higher A is, the higher the weight on the minimization of H_A becomes making it more difficult to minimize the distance (H_B). In fact the variation of the energy, involved in the acceptance of the spin flip, depends directly on both coefficients and, once the minimization of H_A is reached, it is really difficult to escape from the possible local minimum and reach the global one.

In simulated annealing, the escape can happen just through the thermal fluctuation introduced by the temperature T_k but, as we lower A, the variation of the energy decreases, making the move more likely to be accepted: we in fact observe oscillating decreases of the residual energy with the number of sweeps, faster for $A = 1$ and slower for $A = 2$. In simulated quantum annealing, the escape from the local minimum can happen both because of the global move and of the low thermal fluctuation introduced by the temperature MT: the improvement obtained through the simulated quantum annealing is well visible in the figure, where the residual energy goes toward 0 with less difficulty compared with the one of the simulated annealing.



(A) *Energy trend of MC methods over the highest number of iterations considered with A=1.*



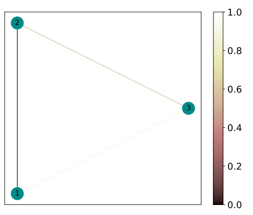
(B) *Energy trend of MC methods over the highest number of iterations considered with A=2.*

Figure 18: *Residual energy for Traveling Salesman problem with 5 cities.*

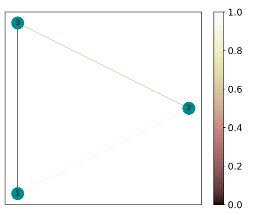
In conclusion of this brief side analysis, it is possible to conclude that the best A is the smallest one that satisfies the constraint $B \max(W_{uv}) < A$.

Configuration display

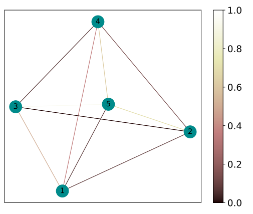
Again, we show the results of the cycles obtained for the different problems considered in Fig. 19. In particular, for the problem with just 3 cities just the results for AQO Fig. 19A and SA Fig. 19B are shown because the result obtained with SQA is the same as AQO: the two different orders observed in AQO and SA are a result of the degeneration of the ground state. In Figs. 19C-D two examples are shown with SQA for the problem with 5 and 10 cities respectively.



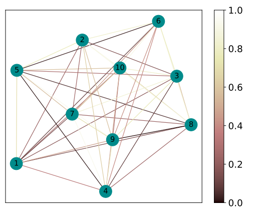
(A) *Order of the cycle for AQO for 3 cities.*



(B) *Order of the cycle for SA for 3 cities.*



(C) *Order of the cycle for SQA for 5 cities and A=1.*



(D) *Order of the cycle for SQA for 10 cities.*

Figure 19: *Order of the cycle of the Traveling Salesman problem.*

4 Self-evaluation

Overall, the results obtained throughout this project, both from the code implementation and the subsequent analysis, are very satisfactory. We managed to implement all the chosen NP problems using Ising-like hamiltonians, obtaining results that appear correct and coherent. We now move on to trying to self-evaluate the developed software as a whole.

4.1 Parameter-based evaluation

Five parameters exist for a well rounded assessment of scientific software: we try to self-evaluate our project according to each.

- **Correctness**

  The results obtained from all algorithms appear to be correct and in accordance with theory. We can strengthen this statement thanks to the fact that the Ising model problem tackled at the beginning was chosen with the specific purpose of testing its results (which have a strong and known theoretical background) to verify the correctness of all the 3 algorithms we built; its results were reliable.

  Throughout the implementation, *incremental testing* was used so as to make sure that all the quantities were initialized and computed correctly; checkpoint statements were also employed to guarantee correctness, with a systematic exploitation of automatic debugging subroutines. As noted in the *Code development* section, pre- and post- conditions are in place to check on any errors that might arise throughout the algorithms and to verify that their theoretical requirements are satisfied.

  Finally, step-by-step guidance is provided through the analysis and key quantities are printed to terminal, so that it is also possible to spot unexpected results on the fly; moreover, this helps the user to be more conscious about the status of the ongoing analysis.

- **Stability**

  Our code shows coherent behaviour across all problems: it withstands a wide range of parameters, it is built in such a way that allows for automatic tuning to search the best parameter configuration for each problem and it works as expected in its domain of application. The AQO algorithm is certainly the most prone to instability issues, especially those caused by the wavefunction time evolution (more details are also discussed in the following point).

  On more practical terms, we made sure to implement stability by carefully allocating and deallocating variables; moreover, stress was put into never comparing quantities with exact equalities: rather, we choose a threshold fixed at 10^{-5}, in order to avoid finite architecture effects.

  Lastly, over the various testings and runs, four flags were raised by the compiler: `IEEE_INVALID_FLAG`, `IEEE_DIVIDE_BY_ZERO`, `IEEE_UNDERFLOW_FLAG` and `IEEE_DENORMAL`. They gave no detectable sign of issues whatsoever in the results, but they make us conscious that something might be (potentially) wrong.

- **Accurate discretization**

  Working with discrete values such as spins, the problem of accurate discretization was not a cardinal one. However, it was still encountered especially in the context of AQO, where a proper window dt had to be chosen for a correct time evolution of the wavefunction ψ. Several trials were made before choosing a value of $dt = 10^{-4}$, which is in a way a sort of compromise between accuracy and computational time.

  In the same context, it was also necessary to employ double precision and double complex variables to attenuate effects of instability due to the finite-precision architecture. We noticed that the probability of staying in the ground state decreased suddenly after a certain amount of iterations, probably due to effects in the norm of the wavefunction. Moving on to double precision, we managed to control this effect; nonetheless, we still acknowledge the possibility of it arising, as said probability trend shows a pronounced oscillating behaviour.

- **Flexibility**

  Major efforts were spent into making this program as flexible as possible: operations have been segmented into multiple subroutines and modules so that they could be reused and combined for different problems. The user is allowed to choose not only what portions of the analysis to perform at each run, but also to directly input some parameters of his/her choice to initialize a problem at will.

  The code has been commented thoroughly in all its steps, especially the least intuitive ones. Moreover, a detailed documentation can be found for all subroutines, explaining the input parameters needed (their function and the type they need to be) and the operations performed.

  A possible limitation is the one inherently present in the solutions of NP problems: as the number of particles/nodes or the number of iterations increases, the computation time explodes. This is especially true for AQO, which requires matrices whose size grows exponentially (which often only makes it possible to study the problems for $N < 12$), but the time becomes challenging to manage also for the other two algorithms when $N > 100$.

  Furthermore, the computation of the ground state obtained through the AQO is problem independent: the code is the same for all the problems since the specific structure of the input hamiltonian is not involved in the time evolution. On the other hand, due the inner structure of the Monte Carlo methods, each problem requires its own subroutine in order to take into account the correct types and parameters, and the specific variation of energy involved in the flips. The general structure of the subroutines is preserved, making the possible addition of new problem really straightforward.

- **Efficiency**

  The whole program takes a very considerable amount of time to run in its entirety (possibly more than a full day), but, as discussed right above, the user can choose what problems and aspects to tackle at each run.

  The choice of algorithms was somewhat bounded to the aims of our project analyses, but it was still tried to make the overall code more efficient where possible. For example, quantities were initialized as `allocatable`, carefully managing their allocations and deallocations to avoid segmentation faults. Also, Lapack functions were used in order to optimize the resolution of some linear algebra matters.

  Finally, the optimization flag `O3` provided by the compiler `gfortran` was used, which slightly increases compilation time with a gain performance of the generated code.

4.2 Conclusions and future insights

We now present some brief conclusive remarks on the comparison between the performance of the three algorithms developed.

There is little doubt that SA and SQA globally outperform AQO: they are usable in a much wider range of particles (AQO necessarily deals with matrices of size 2^N, so $N_{max} \approx 12$ due to memory constraints), their computation time is considerably less and the accuracy tends to be higher. Between SA and SQA, the former implies less computation burden, but the latter has shown to be less dependent on the initial configuration and on the overall complexity of the NP problems.

Future developments of this analysis might start from considering global properties of other NP problems, perhaps using different kinds of algorithms or more computationally-performing resources. The beauty and deepness of this common Ising-like formalism surely appears to be worthy of further investigation.

References

[1] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, p. 5, 02 2014.

[2] G. Passarelli, "Adiabatic quantum annealing of simple np-complete problems," *Scuola Politecnica e delle Scienze di Base - Area Didattica Scienze MM. FF. NN.*, vol. 2, 2017.

[3] Y. Wang, S. Wu, and J. Zou, "Quantum annealing with markov chain monte carlo simulations and d-wave quantum computers," *Statistical Science*, vol. 31, pp. 362–398, 08 2016.

[4] O. Titiloye and A. Crispin, "Quantum annealing of the graph coloring problem," *Discrete Optimization*, vol. 8, pp. 376–384, 05 2011.

[5] A. Das and B. Chakrabarti, "Colloquium: Quantum annealing and analog quantum computation," *Reviews of Modern Physics - REV MOD PHYS*, vol. 80, pp. 1061–1081, 09 2008.

[6] T. Kadowaki, "Study of optimization problems by quantum annealing," 06 2002.

A Variation of the energy due to the spin flip

In order to implement efficiently both the simulated annealing and simulated quantum annealing, the variation of the energy due to a spin flip is computed for each problem.

The results shown refer just to the case of simulated annealing but can be easily converted into the simulated quantum annealing case by including also the index of the replica m. Furthermore, to speed up the computation, each flip is assumed *a priori* and the computation is performed only in case of acceptance of the move.

A.1 Ising model

The original configuration of spin is $\{s_i\}$ and suppose to flip spin i' obtaining the new configuration $\{s'_i\}$: $s'_{i'} = -s_{i'}$.

The hamiltonian is

$$H = -\sum_{\langle i,j\rangle} J_{ij} s_i s_j \tag{26}$$

and the variation of the energy due to the spin flip is

$$\Delta H = 2 s_{i'} \sum_j J_{ij} s_j \tag{27}$$

A.2 Graph partitioning

The original configuration of spin is $\{s_i\}$ and suppose to flip spin i' obtaining the new configuration $\{s'_i\}$: $s'_{i'} = -s_{i'}$.

The hamiltonian is

$$H = A\left(\sum_{i=1}^{N} s_i\right)^2 + B\sum_{ij\in E} \frac{1 - s_i s_j}{2} \tag{28}$$

and the variation of the energy due to the spin flip is

$$\Delta H = A\,4 s_{i'}(s_{i'} - \sum_j s_j) + B\,2 s_{i'} \sum_j s_j \tag{29}$$

A.3 Vertex cover

The original configuration of spin is $\{s_i\}$ and suppose to flip spin i' obtaining the new configuration $\{s'_i\}$: $s'_{i'} = -s_{i'}$.

In order to perform the spin flip, the hamiltonian is rewritten as a function of s_i and hence becomes

$$H = \frac{A}{4}\sum_{uv\in E}(1 - s_u)(1 - s_v) + B\sum_u \frac{s_u + 1}{2} \tag{30}$$

and the variation of the energy due to the spin flip is

$$\Delta H = A s_{i'} \sum_{u:\, ui'\in E}(1 - s_u) - B s_{i'} \tag{31}$$

A.4 Traveling Salesman

The original configuration of spin is $\{s_{v,j}\}$ and suppose to flip spin v', j', obtaining the new configuration $\{s'_{v,j}\}$: $s'_{v',j'} = -s_{v',j'}$.

In order to perform the spin flip, the hamiltonian is rewritten as a function of $s_{v,j}$ and hence becomes

$$\begin{aligned} H =& A\sum_v \left(1 - \sum_j \frac{s_{v,j}+1}{2}\right)^2 + \\ & A\sum_j \left(1 - \sum_v \frac{s_{v,j}+1}{2}\right)^2 + \\ & A\sum_{(uv)\notin E}\sum_j \frac{s_{v,j}+1}{2}\frac{s_{u,j+1}+1}{2} + \\ & B\sum_{uv\in E} w_{uv}\sum_j \frac{s_{v,j}+1}{2}\frac{s_{u,j+1}+1}{2} \end{aligned} \tag{32}$$

and the variation of the energy is

$$\begin{aligned} \Delta H =& A\, s_{v',j'}\left(2 + s_{v',j'} - \sum_j (1 + s_{v',j})\right) + \\ & A\, s_{v',j'}\left(2 + s_{v',j'} - \sum_v (1 + s_{v,j'})\right) - \\ & A\,\frac{s_{v',j'}}{2}\sum_{u:\, uv'\notin E}(2 + s_{u,j'+1} + s_{u,j'-1}) - \\ & B\,\frac{s_{v',j'}}{2}\sum_{u:\, uv'\in E}(2 + s_{u,j'+1} + s_{u,j'-1}) w_{uv'} \end{aligned} \tag{33}$$

B Exemplificative code

The most relevant extracts of the code are shown below for the **graph partitioning** problem.

At first the problem has to be initialized by randomly building the adjacency matrix of the graph with an arbitrary number of nodes N and edges E as shown in Listing 7.

```fortran
!allocate adjacency matrix: NxN
allocate(ADJ(N,N))

!initialize randomly the adjacency matrix
ADJ = 0 !no connections
do ii=1,E
    !draw x and y
    !u in [0,1] -> floor(2*u) = {0,1}
    call random_number(u)
    xx = floor(N*u+1)
    call random_number(u)
    yy = floor(N*u+1)

    !If edge already present or self-interaction,
    !random number is re-drawn
    do while (xx == yy .or. ADJ(xx,yy) == 1 &
                        .or. ADJ(yy,xx) == 1)
        call random_number(u)
        xx = floor(N*u+1)
        call random_number(u)
        yy = floor(N*u+1)
    end do

    !draw an edge, imposing the adj mat to
    !be symmetric, meaning undirected edges
    ADJ(xx,yy) = 1
    ADJ(yy,xx) = 1
end do

!save the matrix in a file if Analysis is .true.
if (Analysis) then
    !write ADJ
    call Save(ADJ, FILEADJ)
end if
```

Listing 7: *Problem initialization*

After the initialization of the problem, the hamiltonian of the problem is computed. In particular, in Listing 8,9 both quantum and classical ones are defined to perform all the computations.

The quantum hamiltonian is built through all the utils subroutines that allow to build each spin matrix σ_i^z. Furthermore, once the hamiltonian is defined, it will be used to compute the true ground state energy and to perform the adiabatic quantum optimization when the number of nodes is sufficiently small.

```fortran
!allocate hamiltonian matrix: 2^Nx2^N
allocate(HP(2**N,2**N),HA(2**N,2**N),HB(2**N,2**N))

!build first part of the hamiltonian HA
HA = 0
do ii=1,N
```

```
        !define sigma_z_i
        call Element(ii,2,N,sigma_z,sigma_i)
        !sum
        HA = HA + sigma_i
        !deallocate sigma_z_i
        deallocate(sigma_i)
    end do
    !compute the square
    HA = matmul(HA,HA)

    !build second part of the hamiltonian HB
    HB = 0
    do ii=1,size(ADJ,1)
        do jj=1,size(ADJ,2)
            !check the connection
            if (ADJ(ii,jj) == 1) then
                call Element(ii,2,N,sigma_z,sigma_u)
                call Element(jj,2,N,sigma_z,sigma_v)
                !retrieve identity
                call Identity(id,size(sigma_u,1))
                !compute the sum
                HB = HB + (id-matmul(sigma_u, sigma_v))/2d0
                !deallocate
                deallocate(sigma_u, sigma_v, id)
            end if
        end do
    end do

    !total hamiltonian
    HP = A*HA + B*HB

    if (Analysis) then
        !write HP
        call Save(HP, FILEHP)
    end if

    !deallocate HA and HB
    deallocate(HA,HB)
```

Listing 8: *Quantum hamiltonian*

The classical hamiltonian is used to determine the *true* ground state energy for the bigger problems and to estimate the GS energy for the simulated annealing and simulated quantum annealing.

```
    !build first part of the hamiltonian HA
    HA = sum(SPIN)**2
    !debug
    call debug(debham,'HA computed!')

    !build first part of the hamiltonian HB
    HB = 0
    do ii=1,N
        do jj=1,N
            !check the connection
            if (ADJ(ii,jj) == 1) then
                !compute the sum
                HB = HB + (1-SPIN(ii)*SPIN(jj))/2d0
            end if
        end do
    end do

    !debug
    call debug(debham,'HB computed!')

    !total hamiltonian
    HP = A*HA + B*HB
```

Listing 9: *Classical hamiltonian*

The main relevant code to perform the adiabatic quantum optimization is shown in Listing 10 where, at each time step, the hamiltonian $H(t)$ is computed and the wavefunction $\psi(t)$ is evolved through the subroutines `HTotInitialization` and `PsiTimeEvolution`, respectively. Furthermore, if required, a physical analysis on the evolution is performed and then saved in external files.

```
    do tt = 0, hamilt%T_final
        call HTotInitialization(hamilt, tt)
        call PsiTimeEvolution(psi_cmplx, hamilt, dtime)

        if (spectrum_analysis .and. mod(tt,1000).eq.0) then
            !diagonalization
            call EigenvecsGSCheck(hamilt%h_tot, psi_cmplx, &
                                  eigvals(index, :), prob(index),deg)
            psi_hamilt = matmul(psi_cmplx, hamilt%h_tot)
            !energy
            energies(index) = dot_product(psi_hamilt, psi_cmplx)
            index = index+1
        end if

    [...]

    end do
```

Listing 10: *Adiabatic Quantum Optimization*

The code containing the computation of each Monte Carlo sweep for the simulated annealing is shown in Listing 11. As defined above, at each sweep all the spin flips are checked by analyzing the variation of energy: each move is accepted or refused by the subroutine `AcceptOrRefuse` that permits to actually flip or not the considered spin. Then, after each sweep, the energy defined by the classical hamiltonian is computed.

```
        do time=1,MaxTime

            do jj=1,n
                !Suppose the spin flip at index j and
                !compute the variation of the energy
                tmp1 = 0
                tmp2 = 0
                do ii=1,N
                    tmp1 = tmp1 + Spin(ii)
                    tmp2 = tmp2 + 2.0*Jij(ii,jj)* &
                                    Spin(ii)*Spin(jj)
                end do
                tmp1 = 4d0*Spin(jj)*(Spin(jj)-tmp1)
                DeltaE = A*tmp1+B*tmp2

                !Accept or refuse the move
                call AcceptOrRefuse(DeltaE, T0, Sign)
                !Flip or not the spin jj
                Spin(jj) = Sign*Spin(jj)

            end do

            !compute the energy of the system
            call ClassicalH(N,Jij,Spin,A,B,tmp1)
            AverageEnergy(time) = AverageEnergy(time)+tmp1

            !reduce temperature
            call Tt(T0, DeltaT)

        end do
    end do
```

Listing 11: *Simulated Annealing*

In the simulated quantum annealing code, shown in Listing 12, it is possible to observe the introduction of the replica index and of the global move. At first, the coefficients that rule the schedules of the annealing are updated, then both local and global moves are performed. As done for the simulated annealing, after each sweep, the energy defined by the classical hamiltonian is computed.

```
        do time=1,MaxTime

            !SCHEDULERS

            !scheduler for H0
            call ACoeffs(As, Time, MaxTime)
            !Scheduler fo HP
            call BCoeffs(Bs,Time, MaxTime)
            !define trotter interaction strenght J(t) at time t
            call Jt(J,As, M, Temperature)


            !LOCAL MOVE

            !loop on the replicas
            do mm=1,M
                do jj=1,n
                    !Suppose the spin flip at index j and compute the
                    !variation of the energy

                    !Variation of the energy for the interaction
                    call TrotterInteraction(J, Spin, mm, jj, DeltaE)
                    !Add variation of the energy on the potential
                    tmp1 = 0
                    tmp2 = 0
                    do ii=1,N
                        tmp1=tmp1 +Spin(mm,ii) !HA
                        tmp2=tmp2 +2d0*Jij(ii,jj)* & !HB
                                    Spin(mm,ii)*Spin(mm,jj)
                    end do
                    tmp1= 4d0*Spin(mm,jj)*(Spin(mm,jj)-tmp1)
                    DeltaE = DeltaE +Bs*(A*tmp1+B*tmp2)

                    !Accept or refuse the move
```

```fortran
                    call AcceptOrRefuse(DeltaE, M, Temperature, Sign)
                    !Flip or not the spin mm,jj
                    Spin(mm,jj) = Sign*Spin(mm,jj)
                end do

            end do

            !GLOBAL MOVE

            do jj=1,n

                !Suppose all the spins in columns jj flip and
                !compute the variation of the energy

                !Variation of the energy on the potential
                tmp1 = 0
                tmp2 = 0
                do mm=1,m
                    tmp3=0
                    do ii=1,N
                        tmp3 = tmp3 +Spin(mm,ii) !HA
                        tmp2 = tmp2 +2d0*Jij(ii,jj)* & !HB
                                    Spin(mm,ii)*Spin(mm,jj)
                    end do
                    tmp3 = 4d0*Spin(mm,jj)*(Spin(mm,jj)-tmp3)
                    tmp1 = tmp1+tmp3
                end do

                DeltaE = Bs*(A*tmp1+ B*tmp2)

                !Accept or refuse the move
                call AcceptOrRefuse(DeltaE, M, Temperature, Sign)
                !Flip or not the spin mm,jj for all mm
                do mm=1,M
                    Spin(mm,jj) = Sign*Spin(mm,jj)
                end do

            end do

            !ENERGY

            !Compute the average energy of the system
            do mm=1,M
                call ClassicalH(N,Jij,Spin(mm,:),A,B,EnergyM(mm))
            end do
            AverageEnergy(time)=AverageEnergy(time)+sum(EnergyM)

        end do
    end do
```

Listing 12: *Simulated Quantum Annealing*

Finally, all the necessary subroutines to study the considered problem, in this case the graph partitioning, are collected inside a dedicated module (`GP`); in particular, the subroutine `GraphPartitioning` contains all the main passages as reported in Listing 13. At first, it is chosen if the study of the performance has to be run and then, if the study of a particular sample has to be performed as well. In case of positive answer to any question, the necessary subroutines are launched and the computation is performed.

```fortran
    print *, ''
    print *, 'GRAPH PARTITIONING'
    print *, 'Do you want to study the performance? (y/n)'
    read (*,*) choice

    if (trim(choice) .eq. 'y' .or. trim(choice) .eq. 'Y') then
        print *, 'Analyzing the performance...'
        !analysis on the performance
        call Performance()
    end if

    print *, ''
    print *, 'Do you want to analyze a sample? (y/n)'
    read (*,*) choice

    if (trim(choice) .eq. 'y' .or. trim(choice) .eq. 'Y') then
        !Initialize the parameters
        call InitializationParameters()

        !Tuning the parameters
        call TuningParameters(.True., .True.)

        !COMPARISON AMONG THE 3 METHODS FOR LESS THAN 12 NODES

        !QUANTUM ANNEALING (performed just for less than 12 nodes)
        if (N <= 12) then
            call QAResolution()
        else
            call GSSQA()
            !print ground state energy
            print *, 'Ground state energy: ', E0
        end if

        !SIMULATED ANNEALING
        call SAResolution()

        !QUANTUM SIMULATED ANNEALING
        call SQAResolution()

        !Finally deallocate all
        deallocate(ADJ, MaxTime)
    end if
```

Listing 13: *Graph partitioning*

Once the results are stored in the specific directories , the required plots are produced through the Python script.